

《FastGS: Training 3D Gaussian Splatting in 100 Seconds》论文研读与实验报告

1. 引言

1.1 研究背景

新视角合成（Novel View Synthesis, NVS）是计算机视觉与计算机图形学领域的重要研究方向，其目标是利用有限视角采集的图像信息重建三维场景，并生成任意视角下的逼真图像。该技术在增强现实（AR）、虚拟现实（VR）、数字孪生以及自动驾驶等领域具有广泛应用价值。

近年来，神经辐射场（Neural Radiance Field, NeRF）通过使用神经网络表示连续场景，实现了高质量的新视角合成效果，在学术界和工业界受到广泛关注。然而，NeRF需要对每个场景进行长时间优化训练，通常需要数小时甚至更长时间才能完成场景重建，难以满足实时应用需求。为解决NeRF训练效率较低的问题，Kerbl等人于2023年提出了3D Gaussian Splatting（3DGS）方法。该方法采用显式三维高斯表示场景，并结合基于Tile的光栅化渲染技术，在保持高质量渲染效果的同时显著提升了训练和渲染速度。得益于其优异的性能，3DGS迅速成为当前三维场景重建与新视角合成领域的重要研究方向，并被广泛应用于动态场景重建、表面重建以及同步定位与地图构建（SLAM）等任务。

1.2 3DGS训练瓶颈

尽管3DGS相比NeRF已经实现了数量级上的速度提升，但其训练过程仍然需要十几分钟甚至几十分钟才能完成一个场景的优化。在实际应用中，这种训练开销仍然限制了其进一步推广和部署。

FastGS论文指出，传统3DGS训练流程主要存在两个方面的问题：一是自适应密度控制（Adaptive Density Control, ADC）过程中会产生大量冗余高斯；二是随着高斯数量不断增长，渲染阶段的计算开销持续增加。近年来已有大量研究针对光栅化流水线进行了优化，因此高斯数量控制逐渐成为影响训练效率的关键因素。

在原始3DGS中，ADC主要包括高斯增殖（Densification）和高斯剪枝（Pruning）两个阶段。训练过程中，高斯数量会随着增殖操作不断增长，最终达到数百万级别。虽然更多高斯能够提高场景表达能力，但也会显著增加优化和渲染成本。

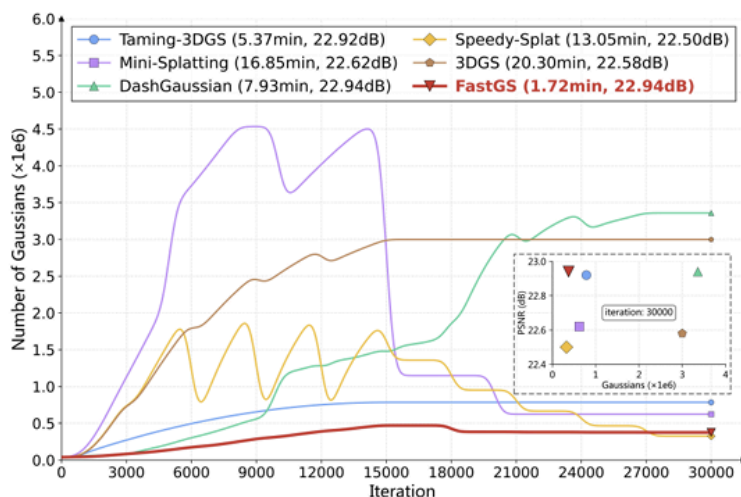


图1 FastGS与其他方法在训练过程中Gaussian数量变化对比

如图1所示，传统3DGS及部分后续改进方法在训练过程中会持续生成大量高斯，而FastGS能够在整个训练阶段保持较低的Gaussian数量。这说明大量新增高斯并不一定能够带来对应的质量提升，其中存在较多冗余表示。随着Gaussian数量增加，优化阶段需要更新更多参数，同时光栅化阶段需要处理更多Gaussian-Tile Pair，从而导致训练时间显著增长。因此，如何有效控制Gaussian数量成为提升3DGS训练效率的重要研究方向。

1.3 现有加速方法及其不足

为了降低训练成本，研究人员提出了多种3DGS加速方法。

一类方法通过限制高斯增长数量来减少训练开销。例如，Taming-3DGS采用预算约束机制控制高斯数量增长；DashGaussian通过动态预算调度策略实现持续增殖控制。这类方法能够在一定程度上降低高斯数量，但通常需要预先设定预算规模，并可能影响最终重建质量。

另一类方法则重点优化高斯剪枝策略。例如，Speedy-Splat通过软剪枝与硬剪枝相结合的方法删除大量高斯，从而加快训练速度。然而，过于激进的剪枝往往会导致场景细节缺失和渲染质量下降。

FastGS作者认为，上述方法的共同问题在于没有充分利用多视角信息来判断高斯的重要性。部分方法仅依据高斯自身属性进行增删决策，而另一些方法虽然引入了多视角信息，但仍采用间接指标衡量高斯贡献，因此容易产生冗余高斯或导致重建质量下降。

图2展示了FastGS的完整算法流水线，并横向对比了Taming-3DGS、Speedy-Splat与本文FastGS三种高斯自适应控制策略的核心差异。

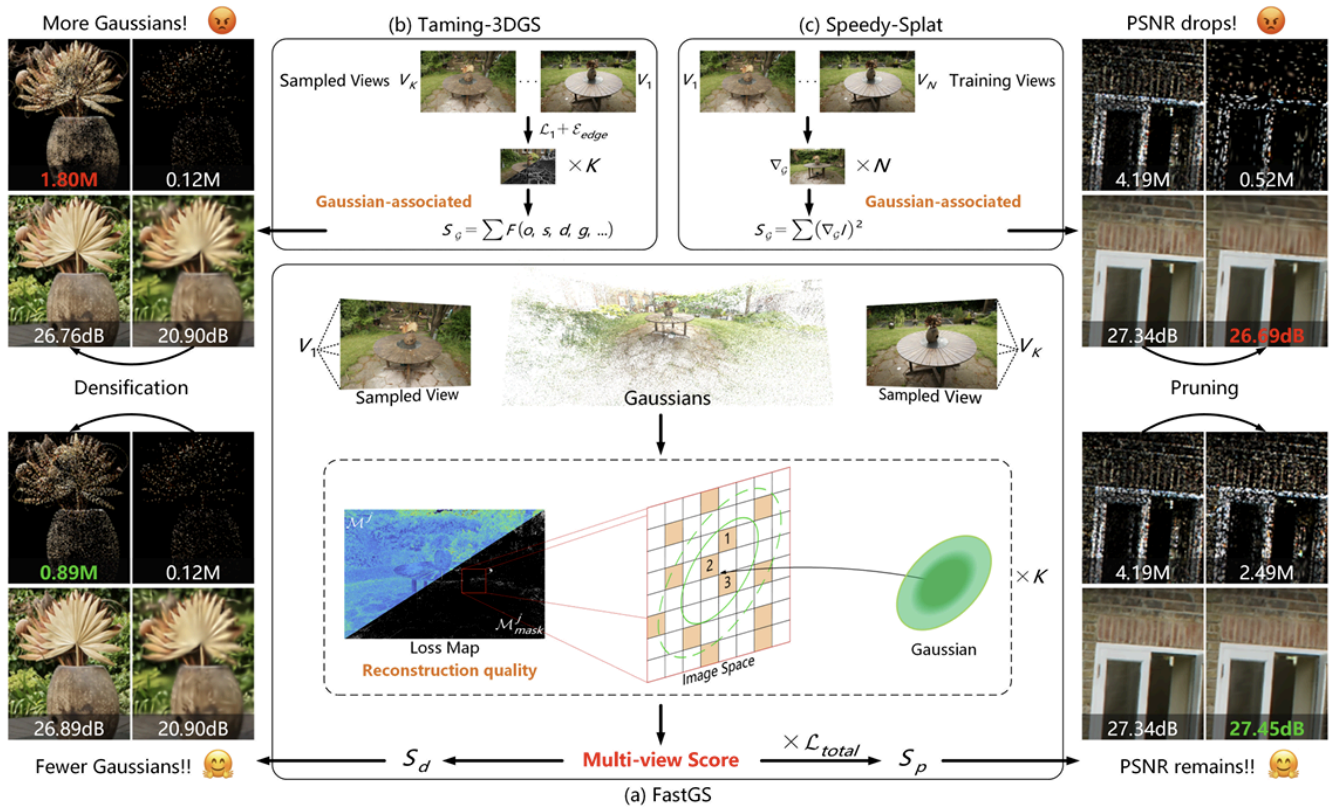


图2 FastGS技术路线

如图2所示，Taming-3DGS虽然考虑了多视角信息，但其重要性评估主要依赖高斯自身属性，缺乏对实际重建质量贡献的直接衡量，因此容易产生冗余高斯。Speedy-Splat利用多视角梯度信息指导剪枝，但其重要性评估方式较为间接，在加速训练的同时可能造成渲染质量下降。相比之下，FastGS直接基于多视角重建误差评估每个Gaussian对场景质量的贡献，并同时指导增殖与剪枝过程，从而在保持渲染质量的同时减少冗余Gaussian。

1.4 FastGS研究动机

在真实场景中，一个三维高斯通常会同时参与多个视角下同一区域的渲染。因此，一个Gaussian是否重要，不应仅由单一视角的信息决定，而应从多个视角共同评估其对场景重建质量的贡献。

基于这一观察，FastGS提出了一种基于多视角一致性的高斯管理策略。其核心思想是利用多个训练视角下的重建误差评估每个Gaussian对场景质量的实际贡献，并据此指导高斯的增殖与剪枝过程。

具体而言，FastGS提出了多视角一致性致密化（Multi-view Consistent Densification, VCD）、多视角一致性剪枝（Multi-view Consistent Pruning, VCP）以及Compact Box（CB）三个关键模块。其中，VCD用于筛选真正需要增殖的高斯，VCP用于删除对多视角重建贡献较小的冗余高斯，而CB通过缩小二维高斯覆盖范围减少光栅化计算量。通过上述设计，FastGS能够在保持渲染质量的同时显著减少训练时间，实现约100秒完成场景训练的目标。

2 3DGS 技术概述

2.1 3DGS场景表示

3D Gaussian Splatting采用显式点云表示方式对三维场景进行建模。与NeRF使用神经网络隐式表示场景不同，3DGS直接使用一组各向异性三维高斯（Anisotropic 3D Gaussian）描述场景结构。

设场景由 N 个三维高斯组成，则每个高斯可以表示为：

$$G_i(x) = \exp\left(-\frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1}(x - \mu_i)\right)$$

其中：

- μ_i ：高斯中心位置；
- Σ_i ：协方差矩阵；
- x ：空间中的三维点。

场景初始化阶段通常利用Structure from Motion（SfM）生成的稀疏点云作为高斯中心。

每个高斯包含位置（Position）、旋转（Rotation）、尺度（Scale）、不透明度（Opacity）以及球谐函数颜色系数（Spherical Harmonics Coefficients）等参数。其中旋转参数与尺度参数共同决定高斯的协方差矩阵：

$$\Sigma = R S S^T R^T$$

通过上述参数，高斯能够同时表达场景的几何结构与外观信息，从而实现了对复杂三维场景的高效表示。

2.2 Gaussian投影与渲染

在渲染过程中，需要将三维高斯投影到二维图像平面。

首先利用相机外参与投影矩阵将三维高斯变换到相机坐标系，并计算其对应的二维协方差矩阵：

$$\Sigma' = J W \Sigma W^T J^T$$

其中：

- W 表示相机变换矩阵；
- J 表示投影变换的Jacobian矩阵。

经过投影后，三维高斯被表示为二维椭圆高斯。

随后按照深度顺序进行排序，并采用Alpha Blending完成颜色累积：

$$C(p) = \sum_i c_i \alpha_i \prod_{j < i} (1 - \alpha_j)$$

其中：

- $C(p)$ 为像素颜色；
- c_i 为高斯颜色；
- α_i 为高斯透明度贡献。

通过这种方式，3DGS能够实现高质量且高效率的场景渲染。

2.3 Tile-Based Rasterization

为了进一步提高渲染效率，3DGS采用了Tile-Based Rasterization机制。与传统体渲染方法需要对大量采样点进行积分计算不同，3DGS利用高斯的显式表示特点，将三维高斯直接投影到二维图像平面后进行光栅化渲染，从而显著提高渲染速度。

在给定相机视角下，首先将场景中的三维高斯投影到图像平面，得到对应的二维椭圆高斯。随后，根据二维高斯在图像中的覆盖区域，将其分配到对应的图像块（Tile）中。图像被划分为若干固定大小的Tile，每个Tile仅处理与其存在重叠关系的高斯，而无需遍历全部高斯，因此能够减少大量无效计算。

在每个Tile内部，系统根据高斯与相机之间的深度关系进行排序，并按照从前到后的顺序执行Alpha Blending操作，将多个高斯的颜色和透明度逐步累积到最终图像中。由于不同Tile之间相互独立，因此该过程能够充分利用GPU并行计算能力，实现高效渲染。

Tile-Based Rasterization是3DGS获得实时渲染能力的重要原因之一。然而，随着训练过程中Gaussian数量不断增加，高斯与Tile之间的匹配关系（Gaussian-Tile Pairs）也会迅速增长，从而导致光栅化阶段的计算开销显著增加。因此，控制Gaussian数量并减少无效Gaussian成为提升训练效率的重要途径，这也是FastGS重点优化的方向之一。

2.4 Adaptive Density Control

为了提高场景表达能力，3DGS在训练过程中引入了自适应密度控制（Adaptive Density Control，ADC）机制。ADC主要包括高斯增殖（Densification）和高斯剪枝（Pruning）两个部分。

2.4.1 Densification

当场景中某些区域存在较大的重建误差时，系统会根据高斯位置梯度判断是否需要增加新的高斯。具体方式主要包括两种：

- Clone：复制当前高斯；
- Split：将当前高斯分裂为多个新的高斯。

通过不断增加高斯数量，系统能够更加精细地表示复杂场景结构，从而提高重建质量。

2.4.2 Pruning

为了避免高斯数量无限增长，3DGS会周期性执行剪枝操作。剪枝过程中主要删除以下两类高斯：

- Opacity较低的高斯；

- Scale过大的高斯。

这些高斯通常对最终渲染贡献较小，因此可以安全移除，从而降低模型规模和计算开销。

2.5 Vanilla 3DGS的局限性

虽然ADC机制有效提升了场景表达能力，但其仍然存在明显不足。

首先，增殖过程主要依据单视角梯度信息进行判断，无法准确反映高斯在多个视角下的真实贡献。其次，高斯数量会随着训练持续增长，容易产生大量冗余高斯，导致优化和渲染成本不断增加。最后，高斯数量膨胀会进一步增加Gaussian-Tile Pair数量，从而降低光栅化效率，延长整体训练时间。

因此，需要一种能够从多视角重建质量角度评估高斯重要性的策略，在保证渲染质量的同时减少冗余高斯数量。FastGS正是在这一背景下提出的一种高效3DGS加速框架，其具体方法将在下一章节中进行介绍。

3. FastGS 方法原理

本章聚焦于 FastGS 论文的方法部分，系统分析VCD、VCP与CB三个核心模块的设计动机、算法细节及其相互关系，此外，还将对 FastGS 与目前主流的几类 3DGS 加速方法进行对比，为后续实验设计提供方法层面的依据。

3.1 研究动机

前文已经介绍了3DGS的ADC机制，从训练效率角度来看，原始 3DGS 主要问题在于：

1. 单视图梯度驱动的Densification容易产生冗余Gaussian；
2. 基于Opacity和Scale的Pruning无法准确衡量Gaussian对重建质量的真实贡献；
3. Gaussian数量增长进一步导致Gaussian-Tile Pair增加，从而降低光栅化效率。

因此，需要一种能够直接从多视图重建质量出发评估Gaussian重要性的策略。

3.2 核心 Insight：多视图一致性

FastGS 的核心思想来源于传统三维重建中的Bundle Adjustment概念——每一个 3D 点应当对来自多个视角的观测都有正面的贡献，而不应仅仅服务于某个特定视角。

现有的加速方法在利用多视图信息时存在两种不足：

- **间接约束**：Taming-3DGS 在 Densification 中虽然采样了多个视角，但其重要性分数完全依赖于高斯自身属性（opacity、scale、depth、gradient 等）的跨视角聚合，而非直接评估该高斯在多个视角下的渲染表现。这种间接方式使得多视图约束较弱，难以精确控制冗余。
- **单向应用**：Speedy-Splat 仅在 Pruning 环节利用跨视角的 Hessian 近似来评估高斯重要性，但没有在 Densification 阶段利用多视图信息。同时，其 Hessian 近似仍然是高斯属性的函数，而非对渲染质量的直接度量。

对于这两种缺陷，FastGS 提出的解决方案为直接评估每个高斯在多个采样视角下对重建质量的实际贡献，从而精确判断哪些高斯需要被致密化来填补重建盲区，哪些高斯可以被删除而不损害多视图重建质量。

3.3 VCD：多视图一致性致密化

VCD（Multi-view Consistent Densification）的设计目标是：仅对那些“在多个视角下都处于高重建误差区域”的高斯进行致密化，从而避免因单视角偶然高误差而引入冗余高斯。

VCD 的具体流程如下：

第一步：采样多视角并计算误差图

从训练视角中随机采样 $K = 10$ 个相机视角 $V = \{v_1, v_2, \dots, v_K\}$ ，对每个视角 v_j ，计算其渲染图像 r_j 与真值图像 g_j 之间的逐像素 L1 误差：

$$e_{u,v}^j = \frac{1}{C} \sum_{c=1}^C |r_{u,v}^{j,c} - g_{u,v}^{j,c}| \quad (6)$$

其中 C 为颜色通道数， (u, v) 为像素坐标。将所有像素的误差聚合为 loss map $M_j \in \mathbb{R}^{W \times H}$ ，并进行 min-max 归一化：

$$M_j = \mathcal{N} \left(\{e_{u,v}^j\}_{u=0, v=0}^{W-1, H-1} \right) \quad (7)$$

第二步：识别高误差像素

设定阈值 $\tau = 0.2$ ，将归一化后的 loss map 二值化，生成高误差像素掩码：

$$M_j^{\text{mask}} = \mathbb{I}(M_j > \tau) \quad (8)$$

其中 $M_j^{\text{mask}}(u, v) = 1$ 表示该像素在当前视角下重建质量差。

第三步：统计每个高斯的高误差覆盖

对每个 3D 高斯原语 G_i ，将其投影到 2D 图像空间得到其 2D 脚印（footprint） Ω_i 。统计在所有采样视角中，落在 Ω_i 内的高误差像素数量，取平均作为该高斯的 densification score：

$$s_i^d = \frac{1}{K} \sum_{j=1}^K \sum_{p \in \Omega_i} \mathbb{I}(M_j^{\text{mask}}(p) = 1) \quad (9)$$

第四步：致密化决策

仅当高斯 G_i 的重要性分数 s_i^d 超过预设阈值 θ_d 时，才对该高斯执行 clone 或 split 操作。这一严格的多视图校验机制确保新增的高斯只会出现在对多视角渲染质量都有改善潜力的区域。

需要注意的是，VCD 可以直接在渲染的前向传播过程中高效地统计高误差像素数量，几乎不带来额外的计算开销。

3.4 VCP：多视图一致性修剪

VCP（Multi-view Consistent Pruning）的设计目标是：基于多视图重建质量的直接评估，删除那些对整体渲染质量贡献微小的冗余高斯，同时保留对关键区域重建有实质作用的高斯。

VCP 的具体流程如下：

第一步：计算各视角的 Photometric Loss

对于同样的 K 个采样视角，计算每个视角的完整 photometric loss，该损失包含 L1 误差和 SSIM 两个分量：

$$E_j^{\text{photo}} = (1 - \lambda)L_1^j + (1 - L_{\text{SSIM}}^j) \quad (10)$$

其中 L_1^j 和 L_{SSIM}^j 分别表示视角 v_j 上全图的平均绝对误差和结构相似性损失。Photometric loss 为重建保真度提供了全局维度的可靠度量。

第二步：计算每个高斯的 Pruning Score

将高误差像素覆盖统计与全局 photometric loss 结合，得到每个高斯 G_i 的 pruning score：

$$s_i^p = \mathcal{N} \left(\sum_{j=1}^K \sum_{p \in \Omega_i} \mathbb{I}(M_j^{\text{mask}}(p) = 1) \right) \cdot (-E_j^{\text{photo}}) \quad (11)$$

第三步：修剪决策

当高斯 G_i 的 pruning score s_i^p 超过阈值 θ_p 时，将其标记为冗余并删除。

VCP 的修剪逻辑可以这样理解：

- 若高斯覆盖了很多高误差像素 → 该高斯位于重建质量差的区域 → 它对改善重建有关键作用 → **保留** (pruning score 低)
- 若高斯覆盖的高误差像素很少 → 该高斯位于已经重建良好的区域 → 同时整个图像的 photometric loss 又很高 → 说明重建差的区域不包含该高斯 → 即它不是对质量有贡献的高斯 → **删除** (pruning score 高)

这种机制确保 VCP 在删除冗余高斯的同时，不会误伤对场景重建至关重要但 opacity 偏低或 scale 偏大的高斯。

VCD 与 VCP 的协同关系：

二者共用同一套多视角误差统计框架（公式 6-8 的高误差像素检测），但在使用方式上形成互补：VCD 用其发现“需要增加高斯来修补”的区域，VCP 用其发现“加了也帮不上忙”的冗余高斯。两者的协同作用使得 FastGS 不需要任何 Gaussian Budget 机制（即人为预设高斯数量上限），也能自然地收敛到一个紧凑且高质量的表示。VCD 负责精准添加，VCP 负责精准删除，共同形成了一个自适应的高斯数量调控回路。

3.5 CB：紧凑盒

CB（Compact Box）直接作用于光栅化预处理的效率优化，与 VCD 和 VCP 面向不同的加速维度。

在光栅化预处理阶段，原始 3DGS 使用 3σ 准则（即以 2D 高斯投影中心为圆心、3 倍标准差为半径）来确定每个高斯覆盖的像素范围。这种方式简单但产生的 2D 椭圆覆盖区域过大，导致每个 Tile 需要处理的高斯数量远超实际有贡献的高斯数量，造成大量无效的 Gaussian-tile 对计算。

Speedy-Splat 提出了精确的 Tile 求交策略，在一定程度上减少了这些冗余对。FastGS 在此基础上进一步引入 CB：利用马氏距离（Mahalanobis distance）判断像素相对于高斯中心的偏离程度，剪枝掉那些马氏距离过大（即高斯对该像素的实际 alpha 贡献已接近零）的 Gaussian-tile 对。

具体而言，对于 2D 投影协方差矩阵为 Σ^{2D} 的高斯，像素 p 到该高斯中心 μ^{2D} 的马氏距离为：

$$d_M(p) = \sqrt{(p - \mu^{2D})^\top (\Sigma^{2D})^{-1} (p - \mu^{2D})}$$

当 $d_M(p)$ 超过一定阈值时，该高斯对该像素的 alpha 贡献已接近为零（参考公式 5 中的高斯函数衰减特性），可以从该 Tile 的处理列表中移除。

从效果上看，如论文 Fig. 4 所示，CB 进一步缩减了每个 2D 高斯的有效覆盖范围，减少了 Gaussian-tile 对的总数。这在不改变渲染质量的前提下，直接降低了正向渲染和反向传播的计算量。CB 是利用 Speedy-Splat 中已有的 `mult` 参数来实现的。

3.6 与同类方法的对比

下表展示了 FastGS 与几类 3DGS 加速方法代表性方法进行对比，如表1所示：

表1 FastGS与同类方法的对比

维度	原始 3DGS	Taming-3DGS	Speedy-Splat	DashGaussian	FastGS
Densification 策略	单视图 梯度	高斯属性分数 (opacity/scale/depth/grad) 跨视图聚合	—（不改 Densification）	分辨率引导的 渐进式调度	多视角高误差 像素直接统计
Pruning 策略	低 opacity / 大 scale	沿袭 vanilla 3DGS	跨视图 Hessian 近似累积	—（使用 Budget 约束）	多视角重建贡 献度直接评估
多视图一致性	无	弱（通过高斯属性间接约束）	弱（通过 Hessian 间接约 束）	无	强（直接评估 渲染质量）
是否需要 Budget	否	是	否	是	否
光栅化优化	3 σ 准 则	无	精确 Tile 求交	无	精确 Tile 求交 + CB
加速机制	基准	Densification 收缩 + 优化策 略	Pruning + 光栅 化优化	Densification 调度	Densification + Pruning + 光栅化全面优 化
核心局限	冗余高 斯多， 训练慢	多视图约束弱，仍产生冗余	间接分数导致 质量下降	需 Budget，高 斯仍百万级	极稀疏视角场 景（< 3 张） 可能需额外先 验

从该表可以看出，FastGS 的核心优势在于：它是目前唯一同时在 Densification 和 Pruning 环节都直接利用多视图渲染质量来决策的方法，因此不需要 Budget 机制即可自然收敛到一个高效的高斯表示。这一特性也使得 FastGS 具有极强的通用性，其 VCD 和 VCP 框架可以被直接接入各类基于 3DGS 的下游任务如动态场景、表面重建、SLAM 等。

3.7 小结

FastGS 的三个模块从不同维度协同加速 3DGS 训练：

- 1. **VCD** 解决了"在哪里添加高斯"的问题，通过多视角渲染误差的直接评估，确保新增高斯精准投放到需要改善的区域；
- 2. **VCP** 解决了"哪些高斯可以删除"的问题，通过对多视角重建贡献的量化，在删除冗余的同时避免损伤关键区域的渲染质量；
- 3. **CB** 解决了"如何让光栅化更快"的问题，通过剪枝无效的 Gaussian-tile 对，降低每次迭代的计算开销。

三个模块的叠加效果在原始论文的 Ablation Study 中得到了充分验证：在 3DGS-accel baseline 上，VCD 将训练时间缩短约 50%、高斯数量降低 80%；VCP 减少 26% 高斯且不损失质量；CB 进一步加速 14%。完整 FastGS 以 baseline 不到 1/3 的训练时间（1.93 min vs 7.10 min）实现了相当甚至更优的渲染质量。

4. FastGS 代码分析

FastGS 的代码主要可以分为训练、渲染和评估三部分：

- `train.py`：读取数据集，初始化 Gaussian 模型，执行优化、增密和剪枝；
- `render.py`：加载训练好的模型并生成测试视角图像；
- `metrics.py`：计算 PSNR、SSIM 和 LPIPS 等指标。

4.1 训练模块

训练模块是 FastGS 的核心，主要入口是 `train.py`，主要流程如下：

1. 解析命令行参数，例如数据路径、迭代次数、增密频率等。
2. 创建 `GaussianModel` 和 `Scene`。
3. 循环训练：
 1. 随机选择一个训练相机，用当前 Gaussian 集合，从当前相机视角进行渲染，得到渲染图片。
 2. 将渲染图片和真实图像进行比较，计算 L1 + SSIM 损失。
 3. 反向传播，计算 Gaussian 参数的梯度。
 4. 根据迭代次数和梯度信息，对 Gaussian 集合进行 densification 或 pruning。
 5. 调用优化器更新 Gaussian 参数。
4. 保存训练好的 Gaussian 模型 `point_cloud.ply`。

4.1.1 `GaussianModel` 和 `Scene`

训练开始时，程序首先创建 `GaussianModel`：

```
gaussians = GaussianModel(dataset.sh_degree, opt.optimizer_type)
```

`GaussianModel` 表示整个场景中的所有 Gaussian 的参数。创建 `GaussianModel` 时传入了两个参数，`sh_degree` 表示球谐函数的最高阶数，用于控制颜色表达能力，`opt.optimizer_type` 表示优化器类型。

初始化时，`GaussianModel` 会先创建若干空张量，存储所有 Gaussian 的位置、颜色特征、尺度等参数。

```
def __init__(self, sh_degree, optimizer_type="default"):
    self.active_sh_degree = 0
    self.optimizer_type = optimizer_type
    self.max_sh_degree = sh_degree
    self._xyz = torch.empty(0)
    self._features_dc = torch.empty(0)
    self._features_rest = torch.empty(0)
    self._scaling = torch.empty(0)
    self._rotation = torch.empty(0)
    self._opacity = torch.empty(0)
    ...
```

创建好空的 `GaussianModel` 后，程序会创建 `Scene`：

```
scene = Scene(dataset, gaussians)
```

`Scene` 首先会读取不同格式的场景数据集，并将场景数据保存到 `scene_info` 中。场景数据包括初始点云、相机、场景归一化尺度等。

```
if os.path.exists(os.path.join(args.source_path, "sparse")):
    scene_info = sceneLoadTypeCallbacks["Colmap"](...)
elif os.path.exists(os.path.join(args.source_path, "transforms_train.json")):
    scene_info = sceneLoadTypeCallbacks["Blender"](...)
```

随后，`Scene` 会构造训练和测试相机，这些相机对象会在训练中提供相机位姿、投影矩阵、图像尺寸和真实图像。

```
self.train_cameras[resolution_scale] = cameraList_from_camInfos(...)
self.test_cameras[resolution_scale] = cameraList_from_camInfos(...)
```

最后，如果不是加载已有模型，`Scene` 会把初始点云中的每个点转换成一个 `Gaussian`，点的位置作为 `Gaussian` 中心，点的颜色转换成球谐颜色特征，尺度根据点之间的距离估计，旋转和不透明度则按默认方式初始化。

```
self.gaussians.create_from_pcd(scene_info.point_cloud, self.cameras_extent)
```

4.1.2 可微渲染

训练中的渲染由 `gaussian_renderer/__init__.py` 中的 `render_fastgs` 实现。

给定一个相机视角和当前 `Gaussian` 集合，`render_fastgs` 会把三维 `Gaussian` 渲染成一张二维图像，并返回训练所需的辅助信息。

```
render_pkg = render_fastgs(viewpoint_cam, gaussians, pipe, bg, opt.mult)
```

首先，创建和 `Gaussian` 数量对应的屏幕空间变量，并保留每个 `Gaussian` 在屏幕空间位置上的梯度。后面训练时会用这个梯度判断哪些 `Gaussian` 对图像误差更敏感。


```

screenspace_points = torch.zeros((pc.get_xyz.shape[0], 4), dtype=pc.get_xyz.dtype,
requires_grad=True, device="cuda") + 0
try:
    screenspace_points.retain_grad()
except:
    pass

```

然后，将当前相机的视角、图像大小、投影矩阵等相机参数整理为 `GaussianRasterizationSettings`，并根据这个配置创建CUDA 光栅化器 `GaussianRasterizer`。

```

tanfovx = math.tan(viewpoint_camera.FoVx * 0.5)
tanfovy = math.tan(viewpoint_camera.FoVy * 0.5)

if metric_map==None:

metric_map=torch.zeros(int(viewpoint_camera.image_height)*int(viewpoint_camera.image_width), dtype=torch.int, device='cuda')

raster_settings = GaussianRasterizationSettings(...)
rasterizer = GaussianRasterizer(raster_settings=raster_settings)

```

随后，从 `GaussianModel` 中取出位置、颜色、尺度、旋转、不透明度，并调用光栅化器进行渲染。

```

rendered_image, radii, accum_metric_counts = rasterizer(
    means3D=means3D,
    means2D=means2D,
    dc=dc,
    shs=shs,
    colors_precomp=colors_precomp,
    opacities=opacity,
    scales=scales,
    rotations=rotations,
    cov3D_precomp=cov3D_precomp
)

```

最后返回渲染图像，以及训练中需要的辅助信息。

```

return {"render": rendered_image, # 当前视角的渲染图像
        "viewspace_points": screenspace_points, # 屏幕空间点，用于保留梯度
        "visibility_filter" : (radii > 0).nonzero(), # 当前视角可见的 Gaussian
        "radii": radii, # Gaussian 在屏幕空间的半径
        "accum_metric_counts" : accum_metric_counts} # FastGS 多视角评分统计量

```

4.1.3 densification 和 pruning

FastGS 相比普通 3DGS 的关键改动集中在 densification 和 pruning。

当 $\text{iteration} < 15000$ ，代码会进入 densification 相关阶段：

1. 记录当前可见 Gaussian 的最大屏幕半径，该信息后续可用于判断 Gaussian 是否在屏幕空间中过大。
2. 累积当前可见 Gaussian 的屏幕空间梯度，该梯度用于判断哪些 Gaussian 对图像误差更敏感，从而作为 densification 依据。
3. 从 $\text{iteration} > 500$ 开始，每隔 `densification_interval` 次执行一次 densification 与 pruning：
 1. 从训练相机中随机采样多个相机。
 2. 用这些相机计算多视角 `importance_score` 和 `pruning_score`。
 3. 调用 `densify_and_prune_fastgs` 修改 Gaussian 集合：
 1. 对需要增强的区域进行 clone 或 split，增加 Gaussian。
 2. 同时删除部分低质量或过大的 Gaussian。
4. 每隔 3000 次重置 opacity，避免部分 Gaussian 过早变得过透明或过不透明，影响后续优化。

```
importance_score, pruning_score = compute_gaussian_score_fastgs(camlist, gaussians,
pipe, bg, opt, DENSIFY=True)
gaussians.densify_and_prune_fastgs(max_screen_size = size_threshold,
                                   min_opacity = 0.005,
                                   extent = scene.cameras_extent,
                                   radii=radii,
                                   args = opt,
                                   importance_score = importance_score,
                                   pruning_score = pruning_score)
```

当 $15000 < \text{iteration} < 30000$ ，且 $\text{iteration} \% 3000 == 0$ 时，代码会进入 final pruning 阶段：

1. 从训练相机中随机采样多个相机。
2. 用这些相机计算多视角 `pruning_score`。
3. 调用 `final_prune_fastgs` 对 Gaussian 集合进行最终剪枝：
 1. 删除 `opacity < 0.1` 的 Gaussian。
 2. 删除 `pruning_score > 0.9` 的 Gaussian。

```
if iteration % 3000 == 0 and iteration > 15_000 and iteration < 30_000:
    my_viewpoint_stack = scene.getTrainCameras().copy()
    camlist = sampling_cameras(my_viewpoint_stack)

    _, pruning_score = compute_gaussian_score_fastgs(camlist, gaussians, pipe, bg,
opt)
    gaussians.final_prune_fastgs(min_opacity = 0.1, pruning_score = pruning_score)
```

4.2 渲染模块

渲染模块的主要入口是 `render.py`。它的作用是加载训练阶段保存的 Gaussian 模型，并使用训练集或测试集相机视角生成渲染图像。

渲染模块的主要流程如下：

1. 解析命令行参数，例如模型路径、指定迭代次数、是否跳过训练集或测试集渲染等。

2. 加载训练好的 Gaussian 模型。
3. 构造 `Scene` 和相机列表。
4. 调用 `render_fastgs` 进行渲染。
5. 保存渲染结果。

4.2.1 模型加载

渲染入口位于 `render.py`:

```
render_sets(  
    model.extract(args),  
    args.iteration,  
    pipeline.extract(args),  
    args.skip_train,  
    args.skip_test,  
    args  
)
```

在 `render_sets` 中, 程序首先创建 `GaussianModel` 和 `Scene`, 加载训练好的高斯模型:

```
gaussians = GaussianModel(dataset.sh_degree, optimizer_type="default")  
scene = Scene(dataset, gaussians, load_iteration=iteration, shuffle=False)
```

4.2.2 渲染训练集和测试集视角

加载模型后, `render_sets` 会根据参数决定渲染哪些相机视角:

```
if not skip_train:  
    render_set(dataset.model_path, "train", scene.loaded_iter,  
               scene.getTrainCameras(), gaussians, pipeline, background, args)  
  
if not skip_test:  
    render_set(dataset.model_path, "test", scene.loaded_iter,  
               scene.getTestCameras(), gaussians, pipeline, background, args)
```

`train_base.sh` 中的渲染命令指定了 `--skip_train`, 因此只渲染测试集视角, 用于后续 `metrics.py` 计算指标。

4.2.3 单个视角的渲染

逐视角进行渲染的函数是 `render_set`。

遍历所有相机视角进行渲染, 每个视角渲染完成后, 程序会保存渲染图像和真实图像, 用于后续 `metrics.py` 进行评估。

```

for idx, view in enumerate(tqdm(views, desc="Rendering progress")):
    start_time = time.time()
    rendering = render_fastgs(view, gaussians, pipeline, background, args.mult)
    ["render"]
    end_time = time.time()
    total_time += (end_time - start_time)
    gt = view.original_image[0:3, :, :]
    torchvision.utils.save_image(rendering, os.path.join(render_path,
'{0:05d}'.format(idx) + ".png"))
    torchvision.utils.save_image(gt, os.path.join(gts_path,
'{0:05d}'.format(idx) + ".png"))

```

4.3 评估模块

评估模块的主要入口是 `metrics.py`。它的作用是读取渲染阶段生成的图像和对应真实图像，计算评估指标，定量评价模型的渲染质量。

评估模块的主要流程如下：

1. 解析命令行参数，读取待评估的模型路径。
2. 进入模型目录下的 `test/` 文件夹，查找不同渲染方法或不同迭代结果。
3. 读取 `renders/` 中的渲染图像和 `gt/` 中的真实图像。
4. 逐张图像计算 SSIM、PSNR 和 LPIPS。
5. 输出平均指标，并保存整体结果和逐视角结果。

4.3.1 图像读取

`metrics.py` 会遍历模型目录下的 `test/ours_xxx/` 文件夹，读取其中的 `renders/` 和 `gt/`。程序会把图像转换为 PyTorch Tensor，并放到 GPU 上计算指标。

```

for fname in os.listdir(renders_dir):
    render = Image.open(renders_dir / fname)
    gt = Image.open(gt_dir / fname)
    renders.append(tf.to_tensor(render).unsqueeze(0)[: , :3, :, :].cuda())
    gts.append(tf.to_tensor(gt).unsqueeze(0)[: , :3, :, :].cuda())

```

4.3.2 指标计算

代码逐张图像计算 SSIM、PSNR 和 LPIPS。PSNR 主要反映像素级重建误差，SSIM 关注结构相似性，LPIPS 则更接近感知质量评价。三种指标结合起来，可以从不同角度衡量渲染结果的质量。

```

for idx in tqdm(range(len(renders)), desc="Metric evaluation progress"):
    ssims.append(ssim(renders[idx], gts[idx]))
    psnrs.append(psnr(renders[idx], gts[idx]))
    lpipss.append(lips(renders[idx], gts[idx], net_type='vgg'))

```

4.3.3 结果保存

评估完成后，程序会输出每个场景的平均指标，并保存 `results.json` 和 `per_view.json`。

`results.json` 保存整个场景的平均 SSIM、PSNR 和 LPIPS，`per_view.json` 保存每张测试图像的具体指标。这样既可以比较整体效果，也可以定位某些表现较差的视角。

4.4 小结

总体来说，FastGS 可以概括为三个阶段：训练阶段从数据集中学习得到 Gaussian 模型，渲染阶段把模型转换为测试视角图像，评估阶段用真实图像计算质量指标。它的核心创新主要体现在训练模块中，即通过多视角一致性评分指导 Gaussian 的增密和剪枝，让模型更关注重建误差较大的区域，同时减少冗余 Gaussian，从而在保持渲染质量的同时提升训练效率。

5. 数据情况分析

本次实验分为两部分：

- 1. 标准公开数据集实验
- 2. 自采 `badge_demo` 展示实验

5.1 标准公开数据集

本次标准公开数据集实验使用了能够公开下载到的 11 个场景，分别来自三个数据来源：

- **MipNeRF360**：7 个场景
`bicycle`、`garden`、`stump`、`room`、`counter`、`kitchen`、`bonsai`
 - 同时包含室外自然场景和室内多物体场景；
 - 更适合观察大范围视角变化、全局几何稳定性和复杂背景下的纹理表现。
- **Tanks and Temples**：2 个场景
`truck`、`train`
 - 以更大尺度、几何结构更突出的室外环境为主；
 - 更适合考察模型的几何表达和紧凑表示能力。
- **Deep Blending**2 个场景
`playroom`、`drjohnson`
 - 室内纹理丰富、遮挡复杂；
 - 更适合观察局部细节、结构保持与感知质量。

为了更清楚地说明数据规模，下面给出 11 个公开场景的数据规模表，如表2所示：

表2 11个公开场景数据规模表

场景	数据集	本地大小	图像数量	场景特点
<code>bicycle</code>	MipNeRF360	2.4G	194	室外自然场景，视角覆盖较广
<code>garden</code>	MipNeRF360	2.9G	185	室外自然/植被场景，背景复杂
<code>stump</code>	MipNeRF360	1.5G	125	室外细碎结构场景
<code>room</code>	MipNeRF360	1.4G	311	室内多物体场景，纹理丰富
<code>counter</code>	MipNeRF360	1.2G	240	室内静物与桌面结构场景

场景	数据集	本地大小	图像数量	场景特点
kitchen	MipNeRF360	1.5G	279	室内复杂纹理与遮挡场景
bonsai	MipNeRF360	1.4G	292	室内小物体与背景混合场景
truck	Tanks and Temples	183M	251	室外大场景，几何结构明显
train	Tanks and Temples	215M	301	更大尺度场景，远近结构复杂
playroom	Deep Blending	151M	225	室内高纹理细节场景
drjohnson	Deep Blending	202M	263	室内遮挡与细节较多场景

从表格可以看出：

1. MipNeRF360 的单场景数据规模明显大于 Tanks and Temples 和 Deep Blending；
2. MipNeRF360 不仅场景更多，而且单场景的图像数据更大，因此它对训练时间与资源占用的影响更明显；
3. Tanks and Temples 和 Deep Blending 尽管体量更小，但由于场景结构和纹理复杂度不同，仍然会对不同技术路线产生明显区分作用。

从数据规模和内容上看，这 11 个场景已经足以覆盖 FastGS 所面对的主要重建类型。因此，本次标准实验包括：11 个场景、3 类公开数据来源、4 条技术路线，每条路线在每个场景上均完成 30000 iter 训练、渲染和评测。

5.2 自采 badge_demo 数据

为了增强展示性并验证方法在真实自采数据上的可用性，本次额外构建了一个自采演示场景 badge_demo。

该场景数据特点如下：

- 总图像数：26 张
- 拍摄方式：手机横屏，围绕目标物体环绕一圈拍摄
- 视角关系：图像按顺序记录相邻视角，首张和末张也互为相邻视角
- 数据来源：自采真实场景

针对这一拍摄方式，后续又对 train/test 划分做了专门调整：不使用默认的固定步长划分，而是从 26 张有序环绕图像中挑选间隔较大的图像作为测试视角，从而让测试视角更具代表性。

经测试，26 张图像全部成功进入 COLMAP 建图流程，所有图像在特征匹配图中都保持连通。因此，这份自采数据在“可用于三维重建”的意义上是有效的。此外，从实验角度看，这份数据集与标准数据集相比不追求大规模 benchmark，而更偏向于方法迁移验证、小尺度真实静物的可视化展示和艺术化与个性化效果补充。

6. 技术路径与实验版本

本次实验一共比较了四个技术路线：

1. baseline
2. axis
3. curriculum
4. improved

6.1 Baseline

baseline 是原始 FastGS 方法，不添加额外结构改动和训练调度改动，作为最基本的对照组。

需要强调的是，这里的 **baseline** 并不是原始3DGS，而是**原论文提出的 FastGS 原始版本**。本次实验中的所有改进，都是在 FastGS 原始论文方法基础之上的进一步调整。

FastGS 原方法的核心思想主要包括：

1. 多视图一致性 densification

在 densify 时，不仅依赖梯度候选，还通过多视图误差统计进一步筛选需要分裂或克隆的 Gaussian。

2. 多视图一致性 pruning

在训练中后期，通过多视图重建一致性分数进一步裁剪冗余 Gaussian。

3. compact box 机制

用更紧凑的覆盖控制减少 rasterization 的无效开销。

因此，**baseline** 本身已经是一个以训练效率和表示控制为目标的增强版 3DGS。

6.2 Axis

axis 版本引入了**主轴定向分裂**。

原始 FastGS 在 split 时采用的是基于 Gaussian 尺度分布的随机扰动采样。改进后的 **axis** 版本则对各向异性更强的 Gaussian 沿其最大尺度主轴进行确定性分裂，这种分裂方式理论上更符合几何结构方向，尤其可能对细长、方向性强的结构更有利。

其核心目标是：

- 提高结构化分裂能力
- 降低随机 split 带来的不稳定性
- 在尽量保持质量的同时减少冗余 Gaussian

从原始 FastGS 论文的角度看，**axis** 改进主要提升了**结构表达的可解释性、分裂方向的确定性、潜在的紧凑表示能力**，也就是说，它试图把 FastGS 的 split 从“更统计驱动”进一步推向“更几何驱动”。

6.3 Curriculum

curriculum 版本引入了**分阶段多视图打分策略**。

原始 FastGS 在多视图一致性打分时固定使用 10 个视角；改进后的 **curriculum** 版本采用：

- 前期：6 个视角
- 中期：8 个视角
- 后期：10 个视角

这样做的动机是：训练早期更关注快速粗筛，不必使用过多视角；训练后期模型逐渐稳定时，再增加视角数量以提升 densify/prune 判定稳定性。这一策略的核心目标是提升优化稳定性、调节辅助评分的力度，并考察训练调度对最终效果的影响。

相对于原始 FastGS 固定 10 视角打分的方式，**curriculum** 的本质提升在于：

- 使辅助打分过程更适应不同训练阶段；
- 在训练早期更强调粗筛和速度；
- 在训练后期更强调判定稳定性和一致性。

6.4 Improved

improved 是组合改进版本，同时启用：

- 主轴定向分裂
- 分阶段多视图打分

它的目标是观察结构级改进与训练调度改进能否形成协同效果，同时比较其在公开标准数据集和自采数据上的行为差异。

该版本最重要的实验意义体现在两个方面：一是验证两种改进是否具有叠加增益效果，二是判断它们在训练过程中的相互影响是互补协同还是存在冲突。

7. 评价指标说明

本次实验主要采用以下评价指标：

7.1 PSNR

Peak Signal-to-Noise Ratio (PSNR) 用于衡量重建图像与真实图像之间的像素级误差，其单位为 dB。PSNR 值越高，表示重建结果与真实图像越接近。该指标主要反映模型在像素层面对场景细节的还原能力，是评价图像重建质量的常用指标之一。

7.2 SSIM

Structural Similarity Index Measure (SSIM) 用于衡量重建图像与真实图像在亮度、对比度和结构信息上的相似程度，其取值范围通常为 0~1。SSIM 值越高，表示两幅图像在结构层面越接近。相比于 PSNR，SSIM 更关注图像的结构保持能力，能够更好地反映边缘和局部纹理的重建效果。

7.3 LPIPS

Learned Perceptual Image Patch Similarity (LPIPS) 是一种基于深度神经网络特征的感知质量评价指标，用于衡量两幅图像在视觉感知层面的差异。LPIPS 值越低，表示重建图像与真实图像在视觉感知上越接近。该指标能够较好地反映纹理细节和图像自然度，因此被广泛应用于新视角合成和神经渲染任务的评价。

7.4 Training Time

Training Time 表示模型从训练开始到完成预设迭代次数所需的总时间，用于衡量模型的训练效率。在保证重建质量的前提下，较短的训练时间意味着更高的计算效率和更好的实际应用价值。

7.5 Gaussian Number

Gaussian Number 表示训练结束后场景中保留的高斯基元数量，用于衡量场景表示的紧凑程度。较少的高斯数量通常意味着更低的存储和渲染开销，但过度压缩可能导致场景细节损失。因此，该指标需要结合 PSNR、SSIM 和 LPIPS 等重建质量指标进行综合分析。

8. 标准公开数据集实验结果

8.1 整体平均结果

在 11 个标准场景上，四个版本的平均质量对比如下图3、4、5所示：

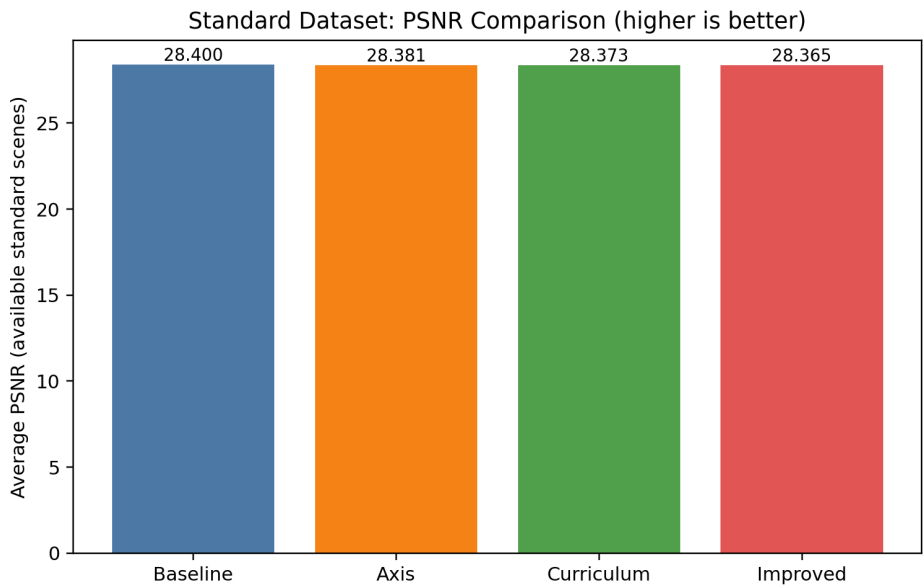


图3 标准集平均 PSNR结果

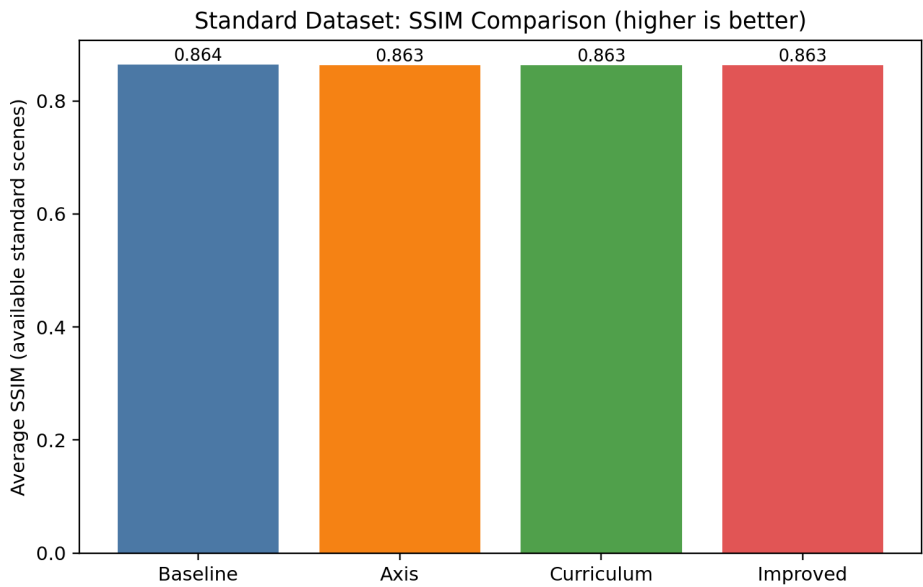


图4 标准集平均 SSIM结果

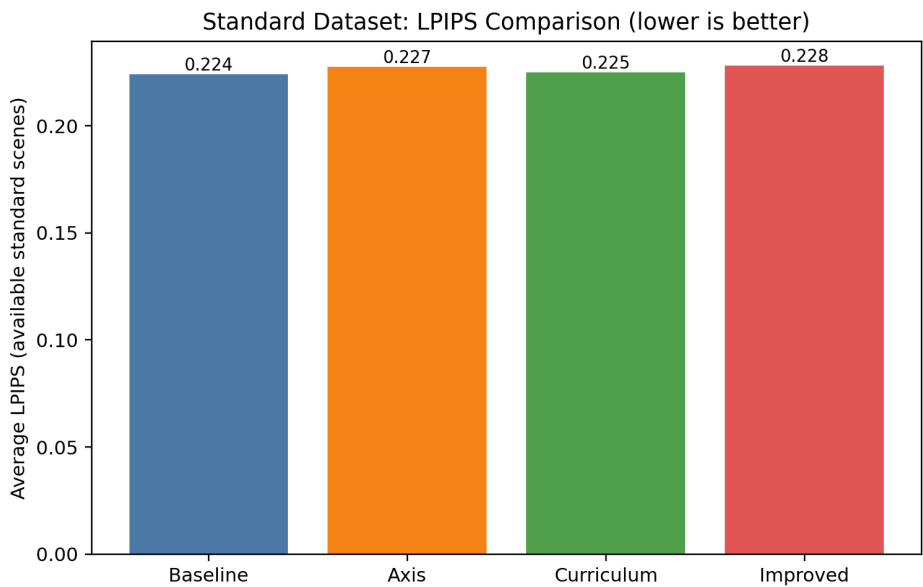


图5 标准集平均 LPIPS结果

从结果可以看出：

- **baseline** 的平均质量最好；
- **curriculum** 与 **baseline** 最接近；
- **axis** 和 **improved** 的 LPIPS 相对更高，说明感知质量有一定退化；
- **improved** 并没有在平均质量上超过 **baseline**。

结合具体定量结果，**baseline**平均 PSNR 最高，为 **28.4004**；平均 SSIM 最高，为 **0.8641**；平均 LPIPS 最低，为 **0.2241**。而**curriculum** 的表现非常接近，平均 PSNR为**28.3730**，平均 SSIM为**0.8632**，平均 LPIPS为 **0.2250**。这说明 **curriculum** 是本次实验中最接近原始方法质量表现的单项改进。

进一步按照数据集来源分别计算指标，评估结果表格如下表3所示：

表3 各数据集平均指标汇总表

数据集	技术路径	平均 PSNR	平均 SSIM	平均 LPIPS
MipNeRF360	baseline	29.1386	0.8584	0.2164
MipNeRF360	axis	29.1445	0.8583	0.2193
MipNeRF360	curriculum	29.1050	0.8577	0.2176
MipNeRF360	improved	29.1146	0.8578	0.2200
Tanks and Temples	baseline	24.1250	0.8423	0.2083
Tanks and Temples	axis	24.1044	0.8398	0.2125
Tanks and Temples	curriculum	24.1890	0.8422	0.2082
Tanks and Temples	improved	24.0902	0.8403	0.2125
Deep Blending	baseline	30.0920	0.9057	0.2666
Deep Blending	axis	29.9876	0.9045	0.2712
Deep Blending	curriculum	29.9951	0.9034	0.2677
Deep Blending	improved	30.0152	0.9047	0.2720
All 11 scenes	baseline	28.4004	0.8641	0.2241
All 11 scenes	axis	28.3814	0.8633	0.2275
All 11 scenes	curriculum	28.3730	0.8632	0.2250
All 11 scenes	improved	28.3648	0.8632	0.2281

这个分组结果表明：

1. MipNeRF360 上，**axis** 的平均 PSNR 略高，但 LPIPS 更差，说明其带来的优势更多体现在像素误差层面，而非感知质量；
2. Tanks and Temples 上，**curriculum** 平均最好，说明分阶段多视图打分在大场景几何优化中更有潜力；
3. Deep Blending 上，**baseline** 仍是最稳，说明室内复杂纹理场景更依赖原方法的稳定性；
4. 整体 11 场景汇总后，**baseline** 仍然保持最优，**curriculum** 最接近 baseline。

8.2 分场景结果

为了更全面地比较不同方法在各个场景上的性能表现，本节进一步统计了所有测试场景的定量评价结果，并汇总于下表4中。表中同时给出了训练时间、高斯数量（Gaussian Number）、PSNR、SSIM 和 LPIPS 等指标，从重建质量、模型紧凑性以及训练效率三个方面对各方法进行综合评价。

表4 全部场景详细实验结果表

场景	技术路径	训练时间(s)	Gaussian 数	PSNR	SSIM	LPIPS
bicycle	baseline	792.68	536,574	24.8313	0.7149	0.3089
bicycle	axis	238.84	436,135	24.8478	0.7153	0.3160
bicycle	curriculum	430.38	512,621	24.8302	0.7141	0.3107
bicycle	improved	427.53	418,810	24.8513	0.7151	0.3168
garden	baseline	882.48	741,275	27.2263	0.8364	0.1733
garden	axis	291.29	677,905	27.2275	0.8359	0.1774
garden	curriculum	589.97	739,105	27.2157	0.8363	0.1738
garden	improved	589.32	672,625	27.2033	0.8356	0.1784
stump	baseline	710.08	386,968	26.6684	0.7558	0.2972
stump	axis	222.85	338,590	26.6603	0.7563	0.3007
stump	curriculum	382.89	367,763	26.6090	0.7539	0.2995
stump	improved	382.11	321,994	26.6768	0.7552	0.3035
room	baseline	337.50	208,259	31.9140	0.9207	0.2170
room	axis	224.19	197,040	31.9814	0.9207	0.2179
room	curriculum	377.01	205,892	31.8867	0.9205	0.2169
room	improved	373.11	196,281	31.9726	0.9205	0.2178
counter	baseline	391.97	208,104	29.1531	0.9079	0.2028
counter	axis	245.84	199,154	29.1646	0.9074	0.2048
counter	curriculum	450.87	206,763	29.1367	0.9076	0.2035
counter	improved	444.51	198,076	29.1595	0.9077	0.2044
kitchen	baseline	575.92	380,544	31.8978	0.9295	0.1258
kitchen	axis	298.89	362,935	31.9213	0.9296	0.1267
kitchen	curriculum	575.32	366,937	31.8124	0.9289	0.1277
kitchen	improved	575.96	354,531	31.8381	0.9292	0.1276
bonsai	baseline	236.91	276,728	32.2793	0.9434	0.1900
bonsai	axis	234.02	263,573	32.2088	0.9429	0.1914

场景	技术路径	训练时间(s)	Gaussian 数	PSNR	SSIM	LPIPS
bonsai	curriculum	420.58	269,731	32.2444	0.9429	0.1910
bonsai	improved	417.55	259,203	32.1005	0.9417	0.1912
truck	baseline	497.40	253,519	25.8163	0.8770	0.1764
truck	axis	510.80	230,619	25.8050	0.8758	0.1800
truck	curriculum	510.07	251,376	25.7315	0.8763	0.1774
truck	improved	495.97	227,945	25.8083	0.8756	0.1807
train	baseline	488.83	229,347	22.4337	0.8076	0.2402
train	axis	524.42	222,336	22.4038	0.8038	0.2449
train	curriculum	516.19	228,478	22.6466	0.8080	0.2391
train	improved	509.13	222,178	22.3722	0.8051	0.2444
playroom	baseline	388.29	182,481	30.5337	0.9106	0.2621
playroom	axis	490.28	172,396	30.4455	0.9091	0.2670
playroom	curriculum	493.83	176,824	30.3560	0.9058	0.2638
playroom	improved	492.38	167,820	30.5045	0.9086	0.2688
drjohnson	baseline	398.02	253,385	29.6503	0.9008	0.2710
drjohnson	axis	488.70	243,410	29.5297	0.8999	0.2754
drjohnson	curriculum	486.53	241,311	29.6342	0.9010	0.2716
drjohnson	improved	484.44	230,622	29.5259	0.9007	0.2751

根据详细分场景评估指标结果，可以得出以下结论：

- 1. **baseline** 在平均指标上最好，但并不是每个场景都最优；
- 2. **curriculum** 在 **train** 上明显最优，说明它对特定大场景有帮助；
- 3. **axis** 和 **improved** 在几乎所有场景上都更紧凑；
- 4. **MipNeRF360** 的部分场景（如 **bicycle, garden, room, kitchen**）中，结构改进确实能带来非常接近 **baseline** 的表现；
- 5. **Deep Blending** 场景更敏感，说明复杂室内纹理对改动更苛刻。

分场景结果如下图6、7、8所示：

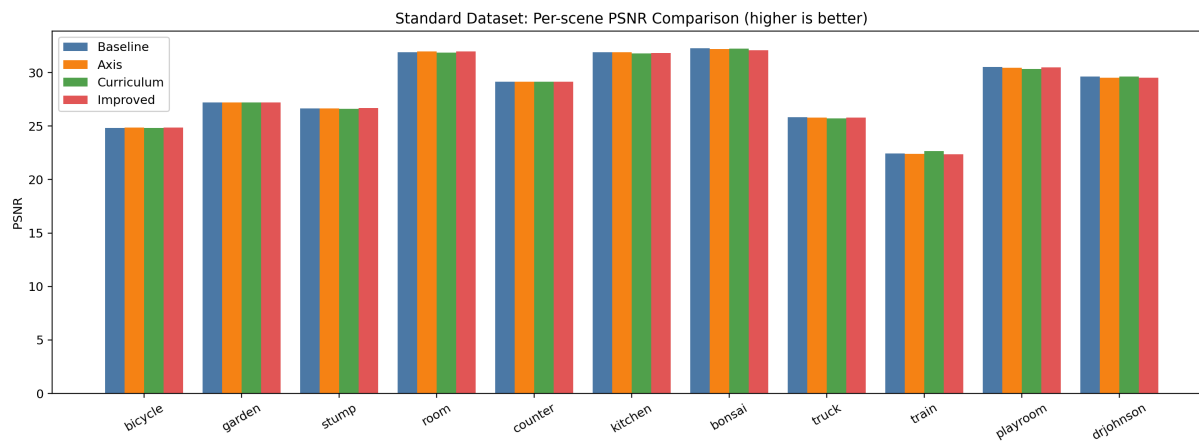


图6 标准集分场景 PSNR结果

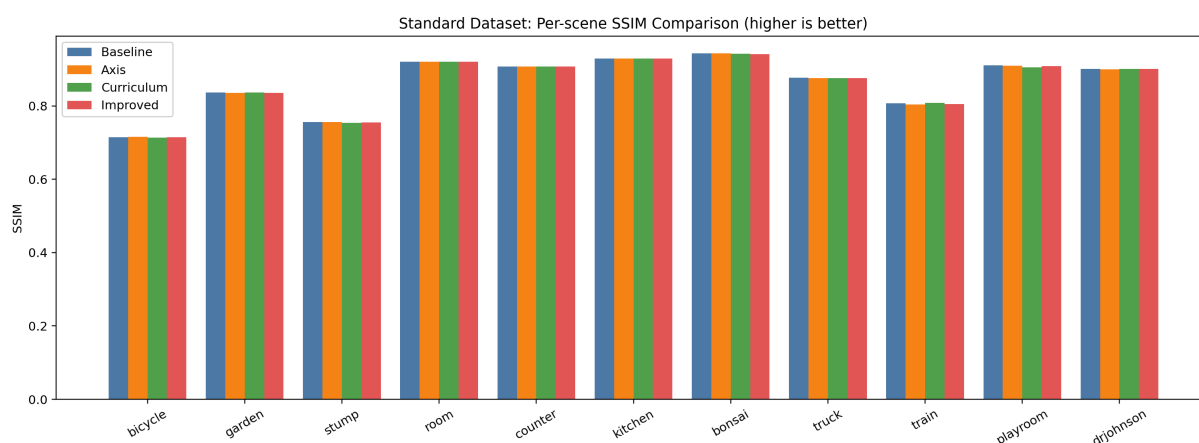


图7 标准集分场景 SSIM结果

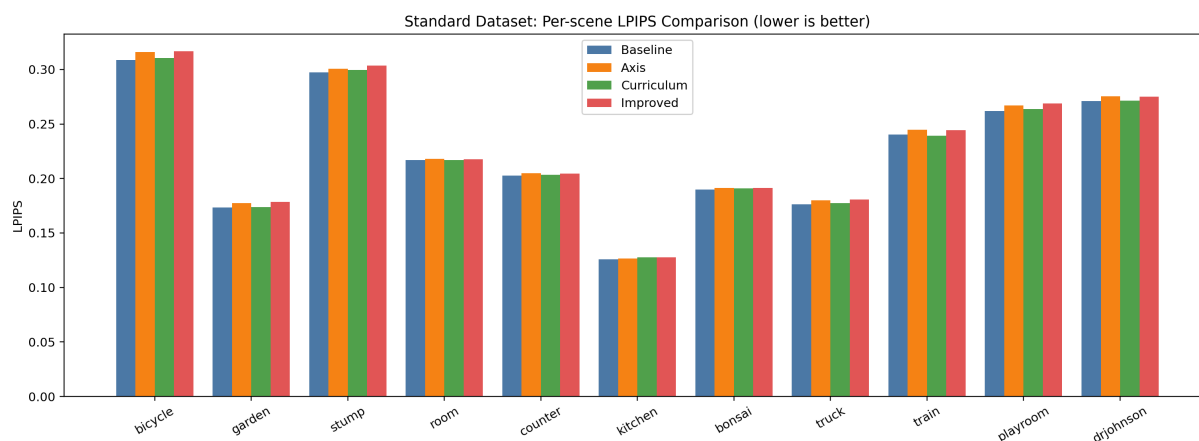


图8 标准集分场景 LPIPS结果

从图中可以看出：

- 某些场景上不同版本差异非常小，说明改动对这些场景影响有限；
- **train** 场景上，**curriculum** 的表现明显突出，是少数在质量上优于 **baseline** 的场景；
- **room** 和 **kitchen** 一类室内场景整体指标较高，说明该类场景更容易获得稳定结果；
- **bicycle**、**stump** 这类室外场景在 LPIPS 上相对更敏感，结构改动更容易带来感知质量波动。

8.3 典型场景可视化对比

为了进一步分析不同方法的重建效果，本文选取若干具有代表性的测试场景进行可视化对比。对于同一测试视角，分别展示真实图像（GT）以及 Baseline、Axis、Curriculum 和 Improved 方法的渲染结果。通过对场景细

节、边缘结构、纹理还原和伪影情况进行观察，可以更直观地评估各方法在视觉质量方面的差异。如图9-15所示。



图9 truck场景可视化结果



图10 train场景可视化结果



图11 playroom场景可视化结果

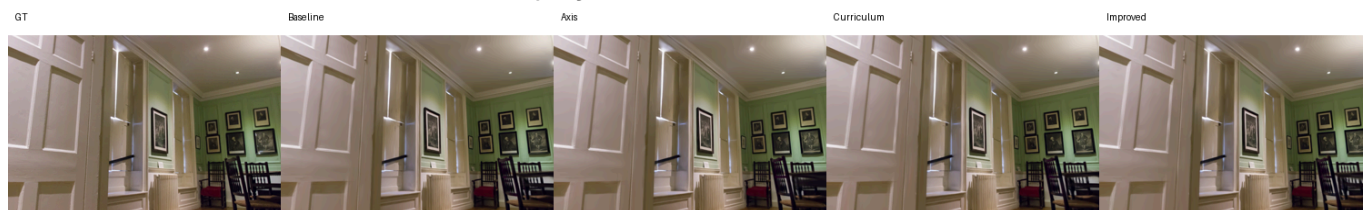


图12 drjohnson场景可视化结果

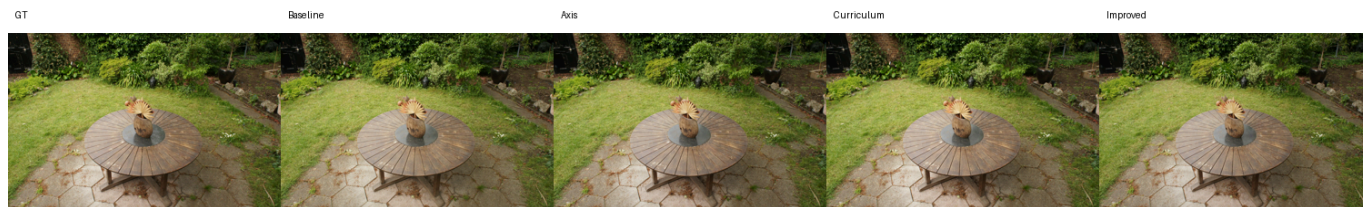


图13 garden场景可视化结果



图14 room场景可视化结果

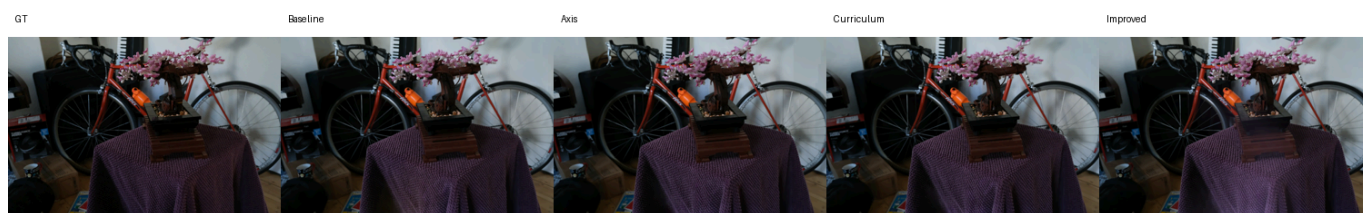


图15 bonsai场景可视化结果

可视化结果表明：

- **baseline** 在大多数场景中依旧保持最稳定的视觉质量；
- **curriculum** 在某些场景中能接近甚至局部优于 baseline；
- **axis** 和 **improved** 往往使模型更紧凑，但在局部纹理细节、边缘平滑度和感知自然度方面可能略有退化。

8.4 标准集实验结论

综合标准公开数据集上的实验结果，可以得到以下结论：

1. **Baseline 方法整体表现最为稳定。** 在大多数测试场景中，Baseline 均能够取得较好的 PSNR、SSIM 和 LPIPS 指标，说明其在重建质量与训练稳定性之间保持了较好的平衡。
2. **Curriculum 策略具有一定的改进效果。** 相较于 Baseline，Curriculum 在保持整体重建质量基本一致的情况下，部分场景的指标出现提升，表明渐进式训练策略有助于优化模型训练过程，并具有进一步研究和改进的潜力。
3. **Axis 方法更注重表示结构的优化。** 从实验结果来看，Axis 在部分场景中的图像质量指标略低于 Baseline，但其结构化的 Gaussian 分裂方式能够减少冗余表示，体现出一定的表示效率优势。
4. **Improved 组合方案未能带来显著的质量提升。** 虽然该方案融合了多种改进策略，但在标准数据集上的性能并未明显优于 Baseline。实验结果表明，不同改进模块之间的协同作用有限，其优势更多体现在表示压缩而非重建质量提升方面。

进一步结合训练时间和 Gaussian 数量进行分析，可以发现：

- 在 **MipNeRF360** 数据集的多个场景中，Axis 方法的训练时间低于 Baseline，说明结构化分裂策略能够在大规模场景下提高训练效率，减少模型优化过程中的计算开销。
- Improved 方法在多个场景中生成的 Gaussian 数量最少，表明其在降低模型复杂度和压缩场景表示方面具有较好的效果。
- 然而，无论是训练效率的提升还是 Gaussian 数量的减少，都未能稳定转化为 LPIPS 等感知质量指标的提升。因此，这些改进更适合作为**提高表示效率和模型紧凑性的方法**，而不是以提升重建画质为主要目标的优化策略。

综上所述，本实验验证了不同改进策略在训练效率、模型规模和重建质量之间存在一定的权衡关系。Baseline 仍然是综合性能最优且最稳定的方案；Curriculum 具有进一步优化的潜力；而 Axis 和 Improved 则更多体现出降低模型复杂度、提高表示效率的优势，为后续研究场景表示压缩与高效训练提供了参考。

9. 自采 badge_demo 实验结果

9.1 自采数据实验定位

badge_demo 不是一个用于严谨 benchmark 的标准数据集，而是一个**真实自采、强调可视化和展示性的补充实验**。

9.2 自采数据四版本对比

其整体定量结果如下图16、17、18所示：

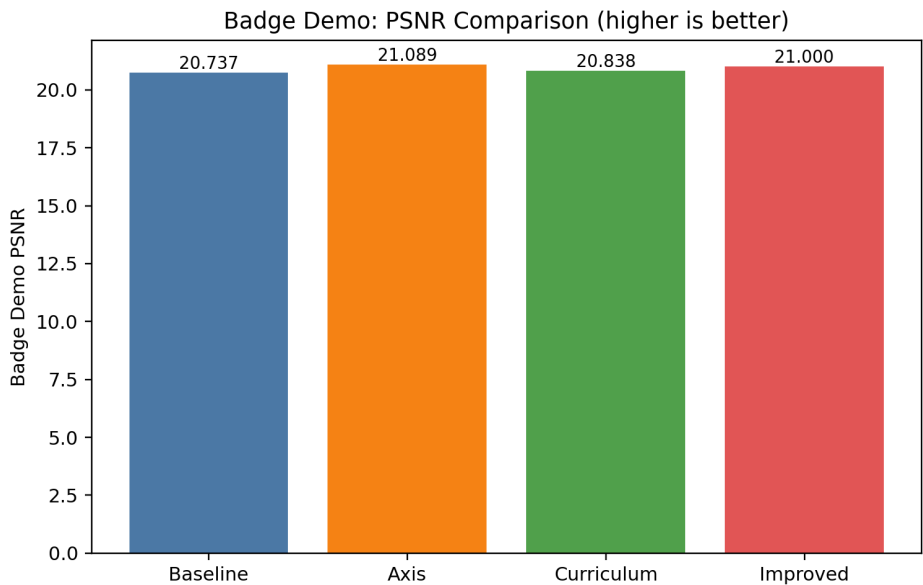


图16 自采数据 PSNR 结果

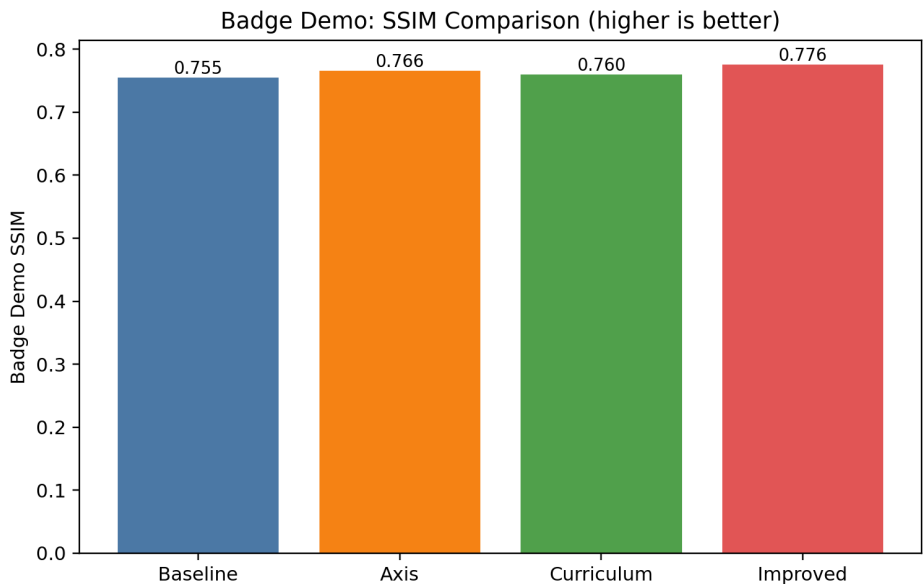
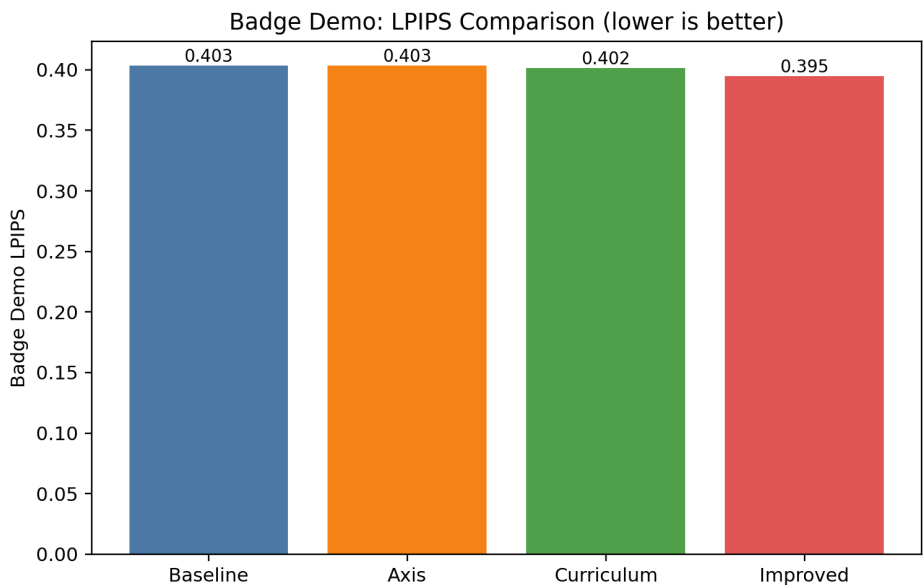


图17 自采数据 SSIM 结果



*图18 自采数据 LPIPS 结果

详细数据如下表5所示。

表5 自采数据上实验指标表格

技术路径	训练时间(s)	Gaussian 数	PSNR	SSIM	LPIPS
baseline	231.48	143,184	20.7372	0.7545	0.4035
axis	231.04	125,690	21.0888	0.7656	0.4033
curriculum	226.19	141,382	20.8375	0.7600	0.4017
improved	223.56	122,433	21.0004	0.7756	0.3947

从表格中可以发现：

- 1. improved 在 badge_demo 上最均衡；
- 2. axis 的 PSNR 最高；
- 3. improved 的 SSIM 最高、LPIPS 最低；
- 4. improved 和 axis 都显著减少了 Gaussian 数量；
- 5. improved 的训练时间也是四个版本中最短。

在自采 demo 上，结论和标准公开数据集有明显不同：

- baseline 并不是最优
- improved 在三项指标上整体最好
- axis 和 curriculum 也都不同程度优于 baseline

这说明：

- 结构化分裂和训练调度改进在小尺度真实物体场景中可能更有利
- 标准数据集上的“最优版本”不一定在自采小物体场景上仍是最优

9.3 自采 demo 可视化对比

下面给出四组代表性测试视角对比图，如图19-22所示。



图19 测试视角 A (00001.png)



图20 测试视角 B (00000.png)



图21 测试视角 C (00003.png)



图22 测试视角 D (00002.png)

这些图中可以观察到：

- 某些视角下，**improved** 的边缘和主体轮廓更稳定；
- **axis** 在部分视角的主体结构还原上优于 **baseline**；
- **curriculum** 对部分视角有一定平滑效果；
- **baseline** 在自采小物体场景下并不总是最优，说明小尺度真实场景确实对技术路线更敏感。

进一步从视角级别分析，如表6所示。

表6 自采数据视角分析表格

技术路径	最好视角	最差视角
baseline	测试视角 B	测试视角 A
axis	测试视角 C	测试视角 A
curriculum	测试视角 B	测试视角 A
improved	测试视角 C	测试视角 A

从最差视角 **测试视角 A** 的结果来看，**improved** 的退化程度最小，这说明它在困难视角上的稳定性可能更强。

9.4 自采 demo 结果结论

从 **badge_demo** 来看：

1. 四个版本都成功完成了从原始照片到建图、训练和渲染的完整流程；
2. 自采数据验证了方法在真实个体场景中的可迁移性；
3. **improved** 在该 demo 上反而优于 **baseline**，说明组合改进在小物体场景中可能更有潜力；
4. 因此，自采 demo 很适合作为“补充展示实验”，而不是与公开 benchmark 同等地位的主结论依据。

同时，**badge_demo** 也说明了一点：

- 在小规模真实静物场景中，结构和调度改进比在公开大场景上更容易体现优势；
- 换言之，改进版不一定能在所有 benchmark 上全面超过 **baseline**，但在“个性化真实小场景”上更有展示潜力。

10. 结果对比分析与总体结论

10.1 标准数据集与自采 demo 的对比

通过本次实验可以观察到：

- 在标准公开数据集上，**baseline** 更稳，**curriculum** 最接近 baseline；
- 在自采小物体场景 **badge_demo** 上，**improved** 更有优势。

这说明：

- 不同数据分布下，方法的行为存在显著差异；
- 标准大场景和真实小尺度静物场景对技术路线的偏好不同；
- 结构化分裂与训练调度改进未必能全面提升标准 benchmark 质量，但在小规模真实物体场景中可能带来更强收益。

10.2 各路线的总体定位

结合所有实验，可以对四条路线做如下总结：

baseline

- 公开数据集上质量最稳
- 是最可靠的高质量参考线

axis

- 更偏结构化分裂与表示紧凑化
- 在标准集上质量略退化
- 在自采 demo 上相较 baseline 有一定提升

curriculum

- 是最有潜力的单项训练改进
- 在公开数据集上最接近 baseline
- 在个别场景上优于 baseline

improved

- 在标准集上更偏“紧凑表示”而非“绝对最佳质量”
- 在自采 demo 上综合表现最好
- 更适合作为“面向个性化小场景的组合改进尝试”

为了更直观对比两类实验，总结如表7所示：

表7 四路线总结对比

路线	标准公开数据集定位	自采 demo 定位
baseline	最稳定的高质量参考线	能稳定完成重建，但不一定是最佳展示版本
axis	更偏结构化分裂与表示紧凑化	对小物体场景有明显帮助

路线	标准公开数据集定位	自采 demo 定位
curriculum	最有潜力的单项训练调度改进	有增益，但不如 axis/improved 显著
improved	更偏向紧凑表示的组合方案	在真实小场景展示中综合效果最好

10.3 最终结论

通过在标准公开数据集以及自采集真实场景上的对比实验，可以得到以下结论：

1. **原始 FastGS 方法具有较好的综合性能和稳定性。** 在标准公开数据集上，其重建质量指标整体表现较优，在不同场景之间具有较好的鲁棒性，是本实验中综合表现最稳定的方案。
2. **分阶段多视图打分（Curriculum）策略具有较好的应用潜力。** 该方法在保持整体重建质量基本不变的前提下，能够在部分场景中取得更好的结果，说明渐进式训练机制有助于提升模型的优化效果，具有进一步研究价值。
3. **主轴定向分裂（Axis）与组合改进（Improved）更侧重于表示效率优化。** 实验结果表明，这两种方法能够有效减少 Gaussian 数量，并在部分场景中降低训练时间，提高场景表示的紧凑性和计算效率，但其重建质量提升并不显著。
4. **改进方法在真实场景中的表现具有一定优势。** 在自采集的小规模真实场景实验中，改进版本在视觉效果和场景细节表达方面展现出一定潜力，说明相关改进策略对特定场景具有较好的适应性，其效果可能受到场景规模、纹理复杂度等因素影响。
5. **重建质量与表示效率之间存在一定权衡。** 标准数据集上的评价指标更关注模型的整体重建精度，而实际场景应用往往同时考虑视觉效果、训练效率和模型规模。因此，不同改进策略在不同应用需求下具有各自优势，需要根据具体任务目标进行选择。

综上所述，本实验验证了不同改进策略对 FastGS 性能的影响。原始 FastGS 在公开基准测试中仍然保持最佳的综合性能，而 Curriculum、Axis 和 Improved 等方法则分别在训练策略优化、表示压缩以及特定场景适应性方面展现出一定优势，为后续研究高效三维场景重建方法提供了参考。

11. 参数级消融与敏感性分析

为分析 FastGS 中关键设计对训练效率、表示紧凑性和重建质量的影响，本文在主实验基础上选择两个代表性场景开展轻量参数分析：**truck** 和 **bicycle**。其中，**truck** 来自 Tanks and Temples，属于大尺度室外几何场景，适合分析 Gaussian 筛选与覆盖范围控制对效率和质量的影响；**bicycle** 来自 MipNeRF360，属于典型室外广视角场景，适合观察多视图采样数量对一致性评分的作用。

需要说明的是，FastGS 论文主文主要通过组件级消融评估 VCD、VCP 和 Compact Box 三个模块的独立贡献；而本文受实验时间和算力限制，未直接复现模块开关式消融，而是在完整 FastGS 框架下围绕关键超参数开展参数级消融与敏感性分析。

所有实验均在 30000 次迭代下完成，并统一记录训练总耗时、最终 Gaussian 数量、测试集渲染帧率以及 PSNR、SSIM、LPIPS 等图像质量指标。围绕这些指标，本文主要分析三组关键参数的影响，分别是 VCD 阈值 **vcd_thresh**、多视图采样数 **K**，以及 Compact Box 的控制参数 **mult**。

11.1 VCD 阈值敏感性分析

FastGS 的多视图一致性增密策略会根据 Gaussian 在多视图高误差区域中的统计结果决定是否继续增殖。该过程中的关键超参数为 VCD 阈值 **vcd_thresh**。阈值越小，表示只要 Gaussian 在较少视图中落入高误差区域，就可

能被视为增密候选；阈值越大，则只有在更多视图中持续表现为高误差相关的 Gaussian 才会被继续增殖。直观上，较小阈值通常会保留更多 Gaussian，提高质量上限，但也会增加训练开销；较大阈值则更强调紧凑表示和训练效率。

本实验在 `truck` 场景上固定其余参数不变，仅改变 `vcd_thresh`，分别测试 `2`、`5`、`10` 和 `20` 四组设置。该实验对应于完整 FastGS 框架下对 VCD 阈值的参数敏感性分析，结果如表 8 所示。

表 8 VCD 阈值敏感性实验结果（truck 场景）

设置	Training Time(s)	Gaussian Num	FPS	PSNR	SSIM	LPIPS
<code>vcd_thresh=2</code>	164.35	294832	309.96	25.7843	0.8746	0.1744
<code>vcd_thresh=5</code>	167.96	253697	267.82	25.7714	0.8726	0.1787
<code>vcd_thresh=10</code>	164.28	212258	310.32	25.6576	0.8694	0.1839
<code>vcd_thresh=20</code>	157.64	171200	281.90	25.4194	0.8639	0.1928

从结果可以看出，随着 `vcd_thresh` 从 `2` 增大到 `20`，最终 Gaussian 数量从 `294832` 持续下降到 `171200`，说明更严格的多视图一致性阈值确实显著抑制了 Gaussian 增长，这一趋势与 FastGS 论文中关于 VCD 的设计动机一致。

质量指标也呈现出较清晰的变化趋势。`vcd_thresh=2` 时三项质量指标最好，PSNR 达到 `25.7843`，SSIM 为 `0.8746`，LPIPS 最低为 `0.1744`。随着阈值增大，PSNR 和 SSIM 逐步下降，而 LPIPS 逐步升高，到 `vcd_thresh=20` 时质量下降最明显。这说明过于严格的增密筛选会抑制对复杂区域的补充表达，使局部细节和整体一致性都受到影响。

训练时间方面，`vcd_thresh=20` 最快，仅为 `157.64s`；而 `vcd_thresh=5` 最慢，为 `167.96s`。虽然时间变化不像 Gaussian 数量那样严格单调，但整体仍然说明较大的阈值有助于减少训练开销。综合质量与效率看，`vcd_thresh=2` 提供了最佳质量，`vcd_thresh=20` 提供了最紧凑表示和最快训练，而 `vcd_thresh=5` 至 `10` 处于相对折中区间。因此，较小的 VCD 阈值更利于保持质量，较大的 VCD 阈值更利于压缩 Gaussian 数量和训练时间，二者之间存在明确的权衡关系。

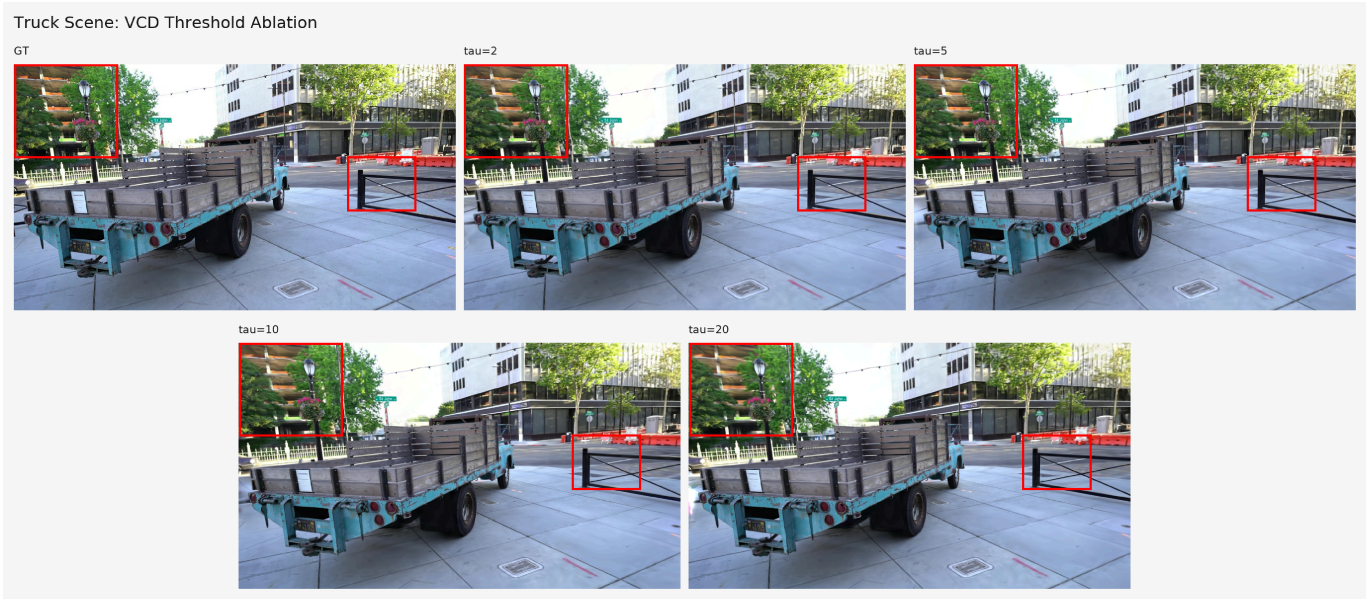


图 23 VCD 阈值敏感性定性对比（truck 场景）

图 23 给出了 **truck** 场景同一测试视角下的定性对比结果。红框标出了左上角的楼房区域以及右侧路面白色标识区域，可以更直观地观察不同 **vcd_thresh** 设置下的局部细节变化。从图中可以看到，**vcd_thresh=2** 时两处区域都最清晰，楼房的结构边缘和路面标识的细节都更接近 **GT**；当阈值增大到 **5**、**10** 和 **20** 后，这些局部区域逐渐变得模糊，尤其在 **vcd_thresh=20** 时退化最明显。这一定性现象与表 7 中的指标趋势一致，进一步说明较大的 VCD 阈值虽然有助于抑制 Gaussian 数量增长，但也会削弱复杂局部结构的重建能力。

11.2 采样视角数 K 敏感性分析

FastGS 的多视图一致性评分依赖于训练阶段随机采样的多视图集合，因此采样视角数 **K** 会直接影响重要性评分以及加密、剪枝过程的稳定性。若采样视角过少，则多视图一致性约束不够充分，容易遗漏跨视角的重要误差区域；若采样视角过多，则虽然判定更稳定，但会带来额外计算开销，而且质量增益可能有限。

本实验在 **bicycle** 场景上固定其余参数不变，仅改变 **mv_num_cams**，分别测试 **5**、**10** 和 **20** 三组设置。该实验关注多视图一致性评分对采样视角数的敏感性，结果如表 9 所示。

表9 采样视角数 K 敏感性实验结果 (**bicycle** 场景)

设置	Training Time(s)	Gaussian Num	FPS	PSNR	SSIM	LPIPS
K=5	220.69	453481	275.45	24.7603	0.7112	0.3147
K=10	231.28	540522	245.83	24.8442	0.7153	0.3093
K=20	243.29	620381	275.90	24.8216	0.7137	0.3106

结果表明，**K=10** 时整体质量最好，**PSNR**、**SSIM** 和 **LPIPS** 分别达到 **24.8442**、**0.7153** 和 **0.3093**。相比之下，**K=5** 的质量略差，说明采样视角过少时，多视图一致性信号不足，导致评分结果不够稳定。继续将 **K** 提升到 **20** 后，质量并未进一步提升，反而较 **K=10** 略有回落，说明更大的视角数并不一定带来等比例收益。

训练时间则随着 **K** 增大而逐步上升，从 **220.69s** 增加到 **243.29s**。这与算法机制一致，因为每次多视图评分都需要渲染更多视角图像并累积统计量，计算成本自然更高。同时，最终 Gaussian 数量也从 **453481** 增长到 **620381**，说明更多视角参与评分后，系统更容易识别跨视角持续存在的高误差区域，从而保留并生成更多 Gaussian。

综合来看，**K=10** 在该实验中体现出最优的质量与效率平衡：相比 **K=5**，它带来了更稳定的质量提升；相比 **K=20**，它避免了过高的时间代价和 Gaussian 膨胀。因此，本组实验支持论文中“默认选择 **K=10** 是合理折中”的结论。图 24 进一步展示了 **bicycle** 场景在 **K=5**、**10**、**20** 下的典型视角对比。



图 24 采样视角数 K 敏感性定性对比 (**bicycle** 场景)

图 24 展示了 **bicycle** 场景下不同 **K** 设置的定性结果。红框标出了上方的树枝区域，可以看到 **K=5** 时该区域最为模糊，细枝末梢的结构恢复不够充分；随着 **K** 从 **10** 增大到 **20**，树枝区域的清晰度有所改善，局部结构更稳定。这说明较少的采样视角会削弱多视图一致性约束，使高频细节更容易丢失；而适当增加 **K** 有助于提升这类区域的重建稳定性。不过结合表 8 的定量结果可以看出，尽管 **K=20** 在局部清晰度上较 **K=5** 有所改善，但其整体质量收益相对有限，同时训练开销进一步增大，因此 **K=10** 仍然是更合理的折中选择。

11.3 Compact Box 参数敏感性分析

除了多视图一致性增删策略外，FastGS 还通过 Compact Box 缩小 2D Gaussian 在屏幕空间中的有效覆盖范围，以减少 Gaussian 与 tile 的配对数量，从而提升光栅化效率。该机制在代码中由参数 `mult` 控制。较小的 `mult` 会让覆盖范围更紧致，理论上可以提升效率，但也可能因为裁剪过度而损害质量；较大的 `mult` 会保留更多覆盖区域，通常更稳健，但计算量也可能更高。这里分析的是 Compact Box 紧致程度的参数效应。

本实验在 `truck` 场景上测试 `mult=0.5` 和 `mult=0.7` 两组设置，其余参数保持一致。结果如表 10 所示。

表10 Compact Box 参数敏感性实验结果（truck 场景）

设置	Training Time(s)	Gaussian Num	FPS	PSNR	SSIM	LPIPS
<code>mult=0.5</code>	156.38	255687	296.83	25.5674	0.8717	0.1786
<code>mult=0.7</code>	176.83	253419	278.04	25.7656	0.8724	0.1786

结果显示，`mult=0.5` 明显缩短了训练时间，并带来了更高的测试渲染 FPS，说明更紧致的覆盖范围确实降低了部分渲染和训练开销。然而在质量指标上，两者 `LPIPS` 几乎相同，而 `mult=0.7` 的 `PSNR` 和 `SSIM` 略高于 `mult=0.5`。这说明当 Compact Box 过于紧凑时，虽然速度更快，但可能会在一定程度上牺牲重建完整性，尤其是对边缘或低贡献但仍有价值的区域覆盖不足。

因此，Compact Box 的作用更偏向效率优化，而不是直接提升质量。就本实验而言，`mult=0.5` 更适合强调速度的配置，`mult=0.7` 则在效率与质量之间更稳健。这一结果也与论文中对 CB 的分析一致，即其主要贡献是减少无效 Gaussian-tile 交互，而非从根本上改变 Gaussian 数量控制逻辑。

11.4 参数分析总结

总体来看，参数实验的规律与论文设计逻辑一致，VCD 通过多视图约束控制 Gaussian 增删，在质量与速度间取得平衡；而 Compact Box 主要承担进一步优化光栅化效率的作用。

首先，FastGS 中真正主导 Gaussian 数量变化和质量-效率平衡的核心仍然是多视图一致性机制，即 VCD 及其相关评分过程。在 `truck` 场景的 `vcd_thresh` 消融中，阈值变化带来了最明显的 Gaussian 数量差异，同时也直接影响了训练时间和三项质量指标，说明 VCD 是控制表示紧凑性和重建质量的重要因素。

其次，采样视角数 `K` 对多视图一致性评分的稳定性具有显著影响。视角数过少会削弱一致性约束，导致质量下降；视角数过多则会增加计算成本，并带来更多 Gaussian，而质量提升有限。实验结果表明，`K=10` 是较为合理的折中选择，这与论文中的默认设置结论一致。

最后，Compact Box 参数 `mult` 的作用更偏向效率调节。较小的 `mult` 能提升速度，但质量会略有下降；较大的 `mult` 更保守，质量略优但开销更高。相比 VCD 阈值和视角数，`mult` 对最终 Gaussian 数量影响较小，也更适合作为渲染效率调节项理解。

作为正文实验的补充，上述消融分析进一步明确了 VCD 阈值、采样视角数和 Compact Box 参数各自的影响方式与作用边界，为理解 FastGS 的设计逻辑提供了更细粒度的实验支撑。

12. 总结与不足

本文在原始论文的基础上，完成了 FastGS 的基本复现流程，包括训练、渲染、指标评测以及结果整理等主要环节。结合主实验与参数分析结果可以看出，当前复现结果在整体趋势上与论文结论基本一致，即 FastGS 能够在尽量保持重建质量的同时显著改善训练效率，并在质量、速度和表示紧凑性之间形成较清晰的权衡关系。参数分

析部分也进一步表明，`vcd_thresh`、`K` 和 `mult` 会分别从增密强度、多视图统计稳定性和覆盖范围控制等方面影响最终结果，这让我们对 FastGS 的工作方式有了更具体的认识。

本次复现工作的不足主要有两点。首先，实验场景数量有限，主实验和参数分析仅覆盖了少量代表性场景，因此结论更适理解对整体趋势的验证。其次，训练时间、FPS、最终 Gaussian 数量以及图像质量指标会受到 GPU 硬件、CUDA 环境、依赖版本和随机采样视角等因素影响，因此本文结果与论文原始数值之间可能存在一定差异。

从方法本身来看，FastGS 的主要贡献在于通过多视图一致性约束更严格地控制 Gaussian 的增密与剪枝过程，从而避免传统 3DGS 在训练中产生过多冗余表示。同时，Compact Box 进一步从光栅化阶段减少无效计算，使该方法不仅在表示规模上更紧凑，也在渲染效率上具有明显优势。结合本文的实验现象可以看出，FastGS 的设计思路是清晰且有效的，并且尤其适合在质量与训练效率之间寻求平衡的场景。

小组分工

霍智童：引言与3DGS技术概述部分、报告整合完善

侯昱帆：FastGS代码分析部分

姜依然：参数级消融实验、结果分析与总结部分

何依航：FastGS方法原理部分

宋泰霖：原始实验复现和技术路线改进对比实验