3D Gaussian Splatting for Real-Time Radiance Field Rendering

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, George Drettakis

2023

摘要

在 3D Gaussian Splatting 领域已经出现了 NeRFs 这类渲染效果很好的神经网络高斯拼接方法，但是这类方法存在训练时间较长的缺点。本篇论文提供了一个新的方法解决了这个问题，实现了能够高质量的实时得到渲染结果的同时，极大的减少了模型的训练时间以及渲染时间。该方法从相机校准期间产生的稀疏点开始，用 3D 高斯表示场景，保留连续体积辐射场的所需属性以进行场景优化，同时避免在空白空间中进行不必要的计算；然后，对 3D 高斯进行交错优化或密度控制，特别是优化各向异性协方差以实现场景的准确表示；最后，使用一种快速可见性感知渲染算法，该算法支持各向异性泼溅，既加速训练又允许实时渲染。

关键词：3D Gaussian Splatting；三维高斯重建；实时渲染

1 引言

首先介绍一下传统的场景重建和渲染的方法。第一个是基于光场进行密度采样，该方法允许非结构化捕捉；然后就是 SFM 出现后的一段时间，使用一组图片来合成新颖的视图，最初仅用于估计稀疏点云，实现 3D 空间的简单可视化，直到 MSV 产生了新的 3D 重建算法。MSV 在许多情况下都有出色的结果，但当 MVS 生成不存在的几何体时，通常无法从未重建区域或“过度重建”中完全恢复，神经渲染算法大大减少了此类问题。在该研究方向，目前存在以下技术难点亟待解决：1、数据的表示和存储：3D Gaussian splatting 需要存储大量的高维信息，每个点对应一个三维位置、协方差矩阵（描述高斯分布的形状和方向）、颜色、透明度等属性，处理和存储这些数据内存开销很大，此外，高斯分布的参数需要足够精确，以确保重建的三维场景细节准确。如果存储的精度不足，可能会导致场景渲染时的模糊或不真实；2、计算的复杂性：由于需要渲染大量的高斯 splats（每个 splat 都是一个高斯函数），计算和渲染的开销非常大，尤其是在处理高分辨率的场景时。为了在实时应用中使用，必须设计高效的算法和数据结构，以减少计算和渲染时间；3、精度问题：从稀疏的三维点云生成高斯 splats，尤其是处理复杂和不规则的场景时，可能会导致重建误差。如何根据点云的密度和分布合理地选择高斯分布的参数（位置、方差等），以保持准确性并减少伪影，是一个技术难点；4、噪声问题：输入的点云数据通常是带有噪声的，需要在生成高斯 splats 的同时去噪并保证渲染结果的质量；5、实时性和交互性：对于需要实时渲染的应用（如虚拟现实、增强现

实等），3D Gaussian Splatting 的渲染速度可能不够快，需要进一步优化算法；6、多模态数据的问题：在很多应用中，3D Gaussian Splatting 需要结合不同类型的数据（如深度图、RGB 图像、激光雷达数据等）。需要有效地将不同模态的信息融合成一致的 3D 高斯分布。

2 相关工作

此部分主要介绍目前主流的方法。

2.1 传统 3D Gaussian Splatting

该方法利用高斯核对 3D 点云进行建模和渲染，每个点被表示为一个具有空间扩展的高斯分布，而不是单个点。每个点的高斯分布包含位置、颜色、透明度等信息。通过对每条射线进行积分，生成光线传播过程中与物体表面相交的颜色和透明度，从而实现对 3D 点云的渲染 [2]。有如下特点：（1）通过高斯核的扩展，避免了传统的点云渲染带来的 “颗粒感”，产生更平滑、自然的渲染效果；（2）相较于传统的网格模型或体素表示，Gaussian Splatting 通常能在较小的内存空间内表示复杂的场景或物体；（3）对于密集的点云数据，高斯核的存储仍然可能占用较多内存，尤其是在需要大量高斯核时；（4）高斯核的渲染和合成过程计算量大，尤其是在处理大型、复杂场景时，可能会导致渲染速度较慢；（5）经典的 3D Gaussian Splatting 主要适用于静态场景，对于动态场景或运动物体的建模，往往需要额外的优化和扩展。

2.2 基于深度学习的 3D Gaussian Splatting 方法（DeepVoxels）

该方法结合了深度学习与 3D Gaussian Splatting，通过神经网络来自动化地生成高斯核的参数，并且优化这些参数以适应特定的输入数据 [3]。网络可以根据训练数据自动学习点云中的结构特征，从而生成更为精确的高斯分布，增强渲染的真实性和细节表现。有如下特点：（1）能够自适应优化，神经网络可以根据输入数据的特征自动调整高斯核的形状、大小和透明度等参数，提高渲染质量；（2）能够实现高质量的渲染效果，通过深度学习，网络能够更精确地捕捉场景中的细节；（3）训练时间较长，深度学习模型通常需要大量的训练数据，且训练过程非常耗时，尤其是在大规模的 3D 数据集上；（4）推理速度慢，虽然神经网络可以提高渲染质量，但推理速度可能会较慢，尤其是在需要实时渲染的场景中；

2.3 基于体积渲染的 3D Gaussian Splatting（NeRF）

该方法将高斯点云与体积渲染技术相结合。体积渲染通过沿着光线的路径对 3D 空间中的每个体积元素进行采样，进而模拟光的传播、吸收和散射。每个点（体素）通过高斯核进行表示，并在渲染时与其他点进行光线合成，模拟真实的光照效果 [1]。有如下特点：（1）通过体积渲染，可以处理反射、折射、光散射等复杂光学现象，使得渲染效果更加真实，尤其适用于半透明物体和复杂光照环境；（2）通过对高斯分布的体积渲染，能够产生非常自然的视觉效果，尤其是在模拟烟雾、云层、雾霾等光学效应时表现出色；（3）计算开销大，通常涉及大量的光线追踪和体积采样操作，因此计算量很大，需要较高的计算资源；（4）由于渲染过程复杂，实时渲染的效率较低，尤其是在大规模场景下，渲染速度受限，实时性差。

2.4 动态高斯拼接 Dynamic Gaussian Splatting

该方法结合了传统的高斯点云和动态物体的建模技术，能够实时调整点云中每个高斯核的参数，适应动态场景或物体的变化，通过引入运动补偿和时间序列的优化，动态高斯点云渲染可以对场景中的动态物体进行高效渲染。有如下特点：（1）通过优化渲染过程中的高斯核和点云更新机制，使得渲染过程更加高效，适用于实时渲染需求；（2）动态场景中物体的光照变化、纹理和透明度的调整困难；（3）计算复杂。

3 本文方法

3.1 本文方法概述

此部分对本文将要复现的工作进行概述，首先，输入一组静态场景的图像，然后由 SFM 校准相应摄像机，得到一组稀疏点云。根据这些点，可以创建了一组 3D 高斯，由位置（平均值）、协方差矩阵和不透明度 定义，能够实现非常灵活的优化机制。由于高度各向异性的体积片可用于紧凑地表示精细结构，这个过程会产生相当紧凑的 3D 场景的表示。辐射场的方向外观分量（颜色）通过球谐函数 (SH) 表示。该算法通过 3D 高斯参数的一系列优化步骤（即位置、协方差、 和 SH 系数）与高斯密度自适应控制的操作交织来创建辐射场表示。该方法效率的关键是基于图块的光栅化器，它允许各向异性图块的 混合，通过快速排序可见性顺序。快速光栅化器还包括通过跟踪累积的 值进行快速向后传递，而对可以接收梯度的高斯数量没有限制 [4]。图的插入如图 1所示：

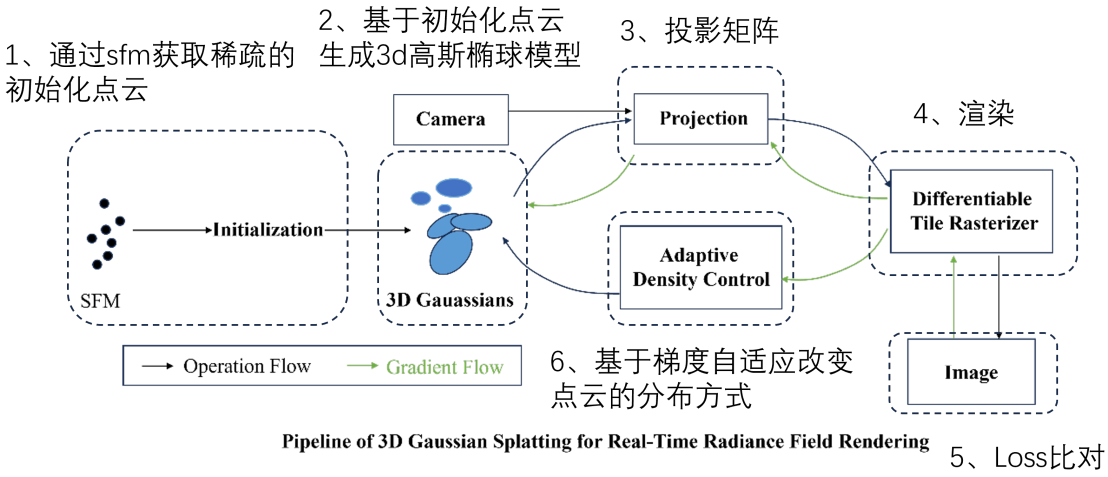


图 1. 方法示意图

将几何体建模为一组不需要法线的 3D 高斯函数。高斯函数由世界空间中定义的完整 3D 协方差矩阵 Σ 定义：

$$G(x) = e^{-\frac{1}{2}(x)^T \Sigma(x)^{-1}} \tag{1}$$

将 3D 高斯投影到 2D 进行渲染，给定观察变换 W，相机坐标中的协方差矩阵 Σ 如下给出：

$$\Sigma' = JW\Sigma w^T J^T \tag{2}$$

3D 高斯的协方差矩阵 Σ 类似于描述椭球体的配置。给定缩放矩阵 S 和旋转矩阵 R，我们可以找到相应的 Σ：

$$\Sigma = RSS^T R^T \tag{3}$$

点云的颜色使用球谐函数来表达，使点云在不同角度下呈现不同的颜色，并且能够提高迭代的速度。

3.2 点云基础

基于点的方法可以有效地渲染断开连接和非结构化的几何样本（即点云），在最简单的形式中，点样本渲染非结构化光栅化具有固定大小的点集，为此可以利用图形 API 的本机支持的点类型或 GPU 上的并行软件光栅化。

3.3 优化过程

3DGS 采用类型 point-based 的渲染公式，通过点云中一定半径范围能影响的像素的 N 个有序点来计算一个点的颜色 C，公式如下：

$$C = \sum_{i \in N} c_i \left(\alpha\right)_i \prod_{j=1}^{i-1} \left(1 - \left(\alpha\right)_j\right) \tag{4}$$

$_i$ 表示当前点 i 的不透明值，$_j$ 表示 i 之前前面的点的不透明值，$1 - _j$ 进行累乘作为权重，表示前面的的点越透明，这个权重越大，当前的不透明值权重越小。损失函数如下：

$$L = (1 - \lambda) L_1 + \lambda L_{D-SSIM} \tag{5}$$

优化基于渲染的连续迭代并将生成的图像与捕获的数据集中的训练视图进行比较。3D 高斯协方差参数的质量对于表示的紧凑性至关重要，因此可以用少量的大各向异性高斯捕获大的均匀区域。快速光栅化是整个算法的瓶颈所在，使用随机梯度下降技术进行优化，充分利用标准 GPU 加速框架，以及为某些操作添加自定义 CUDA 内核的能力来实现快速光栅化。

3.4 自适应高斯控制

从 SFM 的初始稀疏点集开始，然后使用本文中提供的方法自适应地控制单位体积上高斯的数量及其密度，从而使我们能够从初始的稀疏高斯集变为更好地表示场景的更密集的集，并且具有正确的参数。每 100 次迭代进行一次致密化，并删除任何本质上透明的高斯分布，即 小于阈值 。具有高方差的区域中的大高斯需要被分割成更小的高斯，用两个新的高斯函数替换这些高斯函数，并将它们的尺度除以通过实验确定的因子 = 1.6。低方差的区域则复制高斯进行迭代。示意图如 2所示：

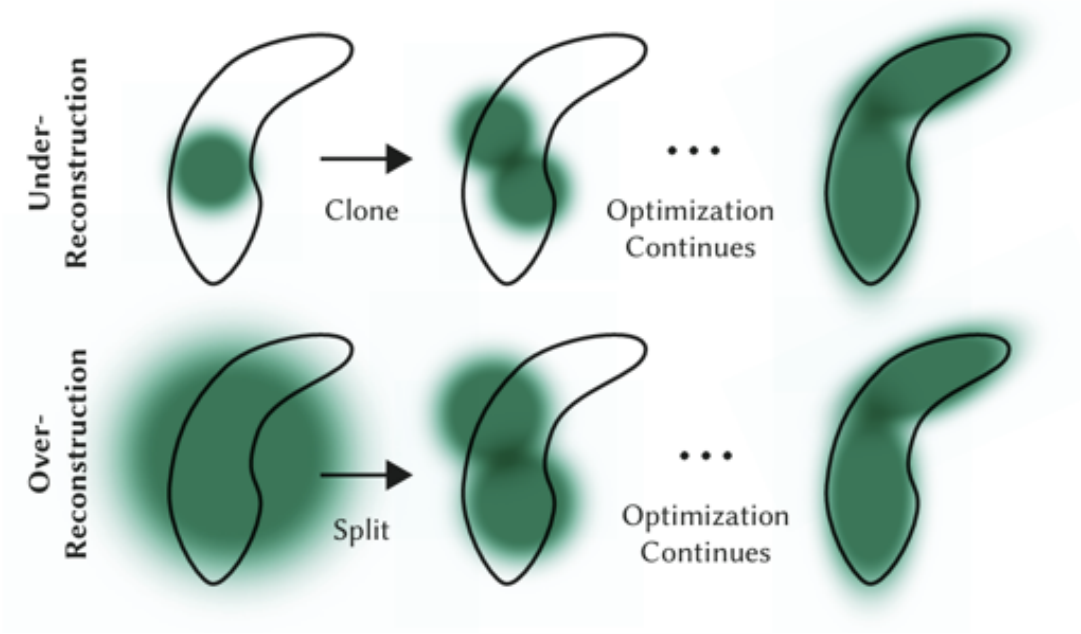


图 2. 自适应控制示意图

3.5 快速光栅化

一次性对整个图像的图元进行预排序，避免了对每个像素进行排序产生额外开销。然后，允许在任意数量的混合高斯上进行有效的反向传播，这样附加内存消耗会降低，每个像素只需要恒定的开销。最后可以对各向异性 splats 进行光栅化。首先将屏幕分割成 16×16 块，然后根据视锥体和每个块剔除 3D 高斯，然后，根据每个高斯重叠的图块数量来实例化它们，并为每个实例分配一个结合了视图空间深度和图块 ID 的键。

4 复现细节

4.1 与已有开源代码对比

参考了原论文的代码部分，主要引用代码的 train 和 render 部分，即对输入的稀疏点云数据进行训练之后进行渲染，这里使用了自己的制作的数据集进行尝试，在对本文论文方法的基础上对球谐函数（SH）部分进行了调参，目的是为了使得高斯模型对于颜色的表达更好，重点在于使用自己的数据集，这部分数据首先拍摄了一组图片后，使用 colmap 进行转换，然后对该组图片进行 sfm。

4.2 实验环境搭建

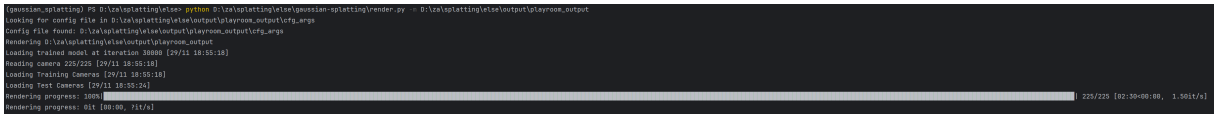
该实验环境的搭建需要特定的版本，其中有两个特定的包需要下载指定版本的编译器才能成功搭建环境，首先，先下载 Visual Studio 2019 版本，随后将 VC 中的工具包 MSVC 添加到系统环境变量中，这样就成功添加了该编译器，之后再下载 Cmake-3.24 以上版本，同样的将 Cmake 添加到系统环境变量中，然后在终端输入如下：conda env create –file environment.yml conda activate gaussian-splatting 等待成功下载即可搭建环境，在搭建环境中容易出现各类问

题，需要对这些问题注意解决。后面安装 SIBR 查看器的时候，需要将 CUDA 的 Viual Studio 工具集成勾选，否则会在安装查看器的时候出现缺少 CUDA 工具箱的报错。

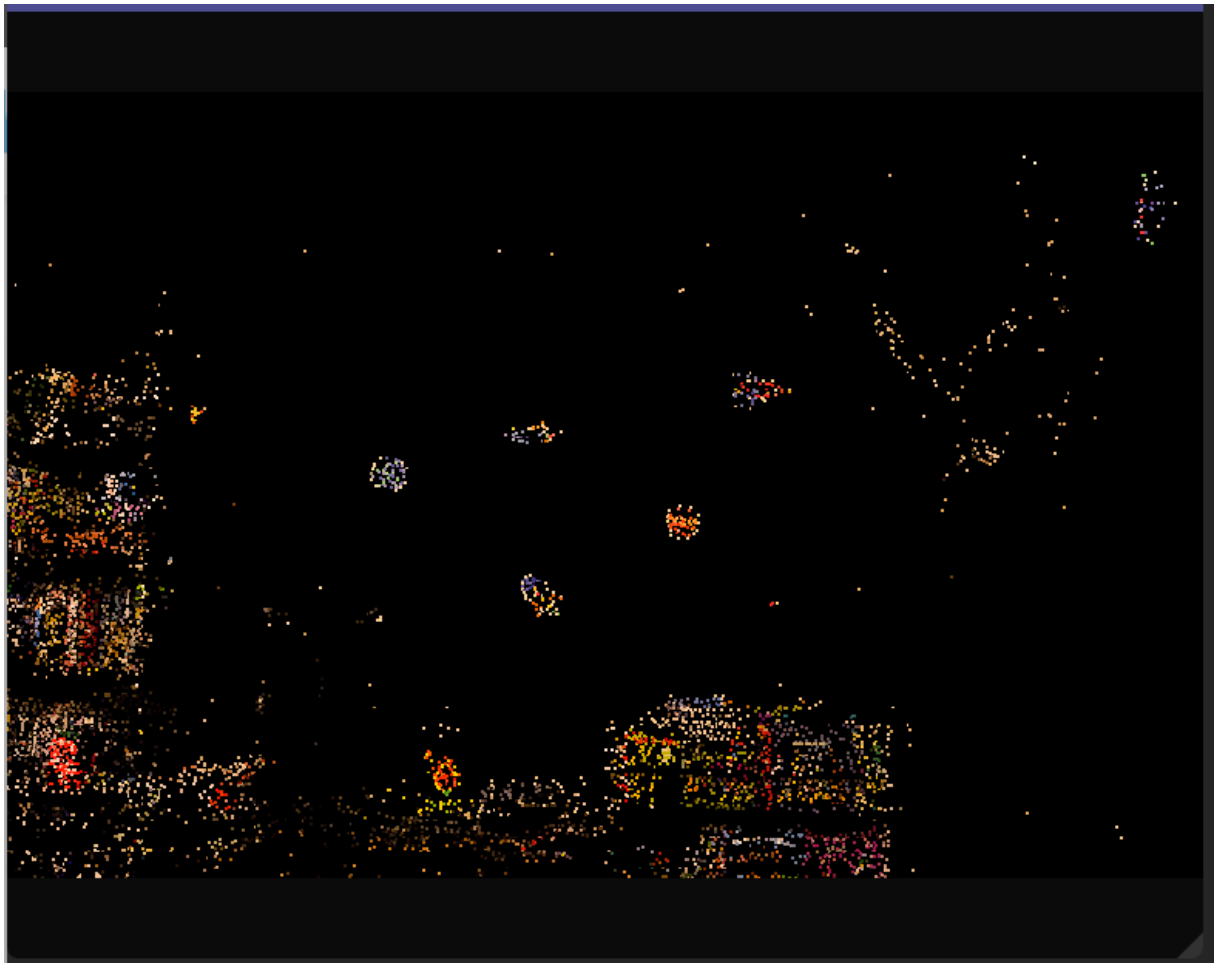
4.3 实验结果

训练过程：

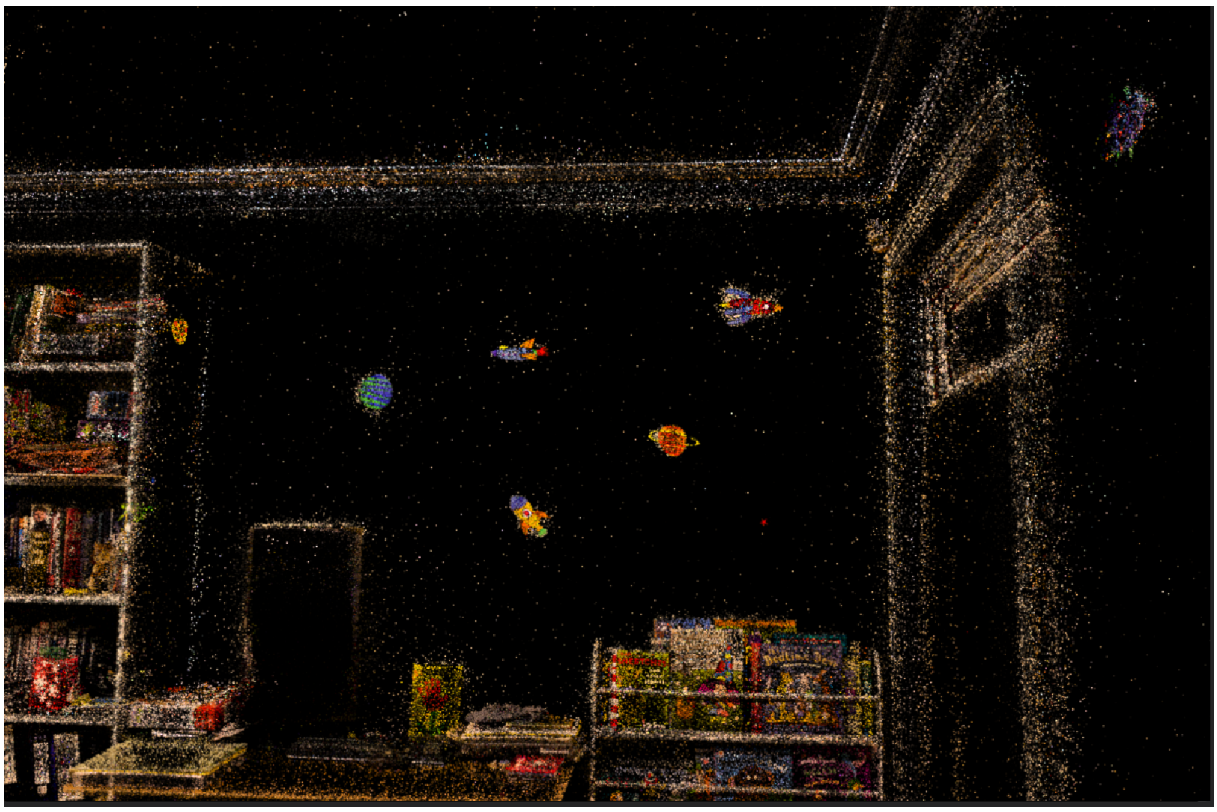
渲染过程：



playroom 数据集的初始输入点云：



椭球模型的中心点：



3D 椭球模型：

不透明度为 0.4 的结果：

实时渲染的最后结果：

4.4 创新点

使用高斯椭球体（Gaussian ellipsoids）可以更好地表达点的局部形状和不确定性，这种表示方式能够更精确地反映物体表面的几何形状，尤其是在具有一定模糊性和噪声的点云数

据中。高斯椭球模型通常只需要存储每个点的少量参数（例如位置、协方差矩阵），而不需要大量存储复杂的网格结构或高分辨率数据，因此能够通过简单的卷积或加权渲染方法快速渲染。另一个就是该方法增强鲁棒性，具有更强的抗噪声能力，在高斯椭球模型中，由于每个点被表示为一个高斯分布，可以有效地平滑噪声和不规则性，提供更稳定的重建效果。

5 实验结果分析

由于并没有像论文中进行的对比实验那样，通过制定指标来比较该算法和其他算法的优劣，这里只根据我本次在两个不同环境下运行的结果给出结论，Playroom 是在 4080s 下运行得出的结果，训练时间 39 分钟，渲染时间 7 分钟；drjohnson 是在 4060 下运行得出的结果，训练时间 51 分钟，渲染时间 14 分钟，从上可以分析得到该论文的算法优化，确实能够改进 3DGS 的训练时间的问题，其他更详细的结论这里不给出了，可以参考论文中的各类对比结果、消融实验的结果。结果如下图所示：

Dataset	Mip-NeRF360						Tanks&Temples						Deep Blending					
Method|Metric	$SSIM^{\uparrow}$	$PSNR^{\uparrow}$	$LPIPS^{\downarrow}$	Train	FPS	Mem	$SSIM^{\uparrow}$	$PSNR^{\uparrow}$	$LPIPS^{\downarrow}$	Train	FPS	Mem	$SSIM^{\uparrow}$	$PSNR^{\uparrow}$	$LPIPS^{\downarrow}$	Train	FPS	Mem
Plenoxels	0.626	23.08	0.463	25m49s	6.79	2.1GB	0.719	21.08	0.379	25m5s	13.0	2.3GB	0.795	23.06	0.510	27m49s	11.2	2.7GB
INGP-Base	0.671	25.30	0.371	5m37s	11.7	13MB	0.723	21.72	0.330	5m26s	17.1	13MB	0.797	23.62	0.423	6m31s	3.26	13MB
INGP-Big	0.699	25.59	0.331	7m30s	9.43	48MB	0.745	21.92	0.305	6m59s	14.4	48MB	0.817	24.96	0.390	8m	2.79	48MB
M-NeRF360	0.792	27.69	0.237	48h	0.06	8.6MB	0.759	22.22	0.257	48h	0.14	8.6MB	0.901	29.40	0.245	48h	0.09	8.6MB
Ours-7K	0.770	25.60	0.279	6m25s	160	523MB	0.767	21.20	0.280	6m55s	197	270MB	0.875	27.78	0.317	4m35s	172	386MB
Ours-30K	0.815	27.21	0.214	41m33s	134	734MB	0.841	23.14	0.183	26m54s	154	411MB	0.903	29.41	0.243	36m2s	137	676MB

图 3. 实验结果示意

6 总结与展望

从论文描述的内容来看，依然存在的局限性在于在场景观察不佳的区域，会出现伪影以及内存消耗明显高于基于 NeRF。在场景观察不佳的地方，可以尝试使用自适应分辨率技术来增加细节，对场景的渲染进行动态调整，采用更精细的细节等级来表示观察不到或重要性较低的区域，减少内存消耗。在出现伪影的地方，可以通过训练专门的伪影检测网络来识别和去除伪影，或者结合光线传播路径的优化，减小伪影产生的可能性，或者通过增加相对较强的先验知识（如物理渲染模型）或后处理的去噪技术，可以进一步优化生成质量。坦白的说，我并不是研究这个方向，我只能从我的研究角度给出我的改进思路，我们可以通过生成对抗网络对效果不佳的地方进行训练来增加该区域的鲁棒性，从而能够降低伪影的生成概率。

参考文献

[1] Yanqi Bao, Tianyu Ding, Jing Huo, Yaoli Liu, Yuxin Li, Wenbin Li, Yang Gao, and Jiebo Luo. 3d gaussian splatting: Survey, technologies, challenges, and opportunities. *arXiv preprint arXiv:2407.17418*, 2024.

[2] Guikun Chen and Wenguan Wang. A survey on 3d gaussian splatting. *arXiv preprint arXiv:2401.03890*, 2024.

[3] Ben Fei, Jingyi Xu, Rui Zhang, Qingyuan Zhou, Weidong Yang, and Ying He. 3d gaussian splatting as new era: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 2024.

[4] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023.