浙江大學

硕 士 研 究 生 读 书 报 告

题目 基于 3D 高斯飞溅的神经渲染方法

作者姓名 闵致远

作者学号 22351317

指导教师 李启雷

学科专业 人工智能

所在学院 软件学院

提交日期 二〇二三年一月五日

摘要

本报告基于对《3D Gaussian Splatting for Real-Time Radiance Field Rendering》的深入分析，探讨了一种创新的基于 3D 高斯飞溅（Gaussian Splatting）的神经渲染方法。该方法在实时辐射场渲染领域提出了突破性的技术,通过结合 3D 高斯模型和高效的渲染算法,实现了在保持高视觉质量的同时，大幅提升渲染速度。

原论文的核心贡献在于其对 3D 高斯表示法的创新应用，该表示法通过模拟场景中的光线分布，提供了一种更为灵活和富有表现力的场景描述方式。这种方法不仅提高了场景的真实感，还为后续的优化和渲染过程提供了便利。此外，原论文中提出的交错优化和密度控制策略，有效地提高了渲染效率，这对于实时应用场景尤为重要。

在实验评估方面，原论文通过与现有技术的对比分析，展示了该方法在视觉质量和渲染速度上的显著优势。这一结果不仅证明了 3D 高斯飞溅方法在理论上的有效性，也展示了其在实际应用中的潜力，特别是在虚拟现实、增强现实和游戏开发等领域。

综上所述，原论文提出的基于 3D 高斯飞溅的神经渲染方法，在实时辐射场渲染技术领域中代表了一种重要的进步。该方法不仅在提高渲染质量和速度方面取得了显著成果，也为未来相关技术的发展提供了新的研究方向和应用可能。

关键词：实时渲染；神经渲染；3D 高斯飞溅；

1 引言

计算机图形学领域随着实时渲染技术的出现而经历了一次范式转变，这在从虚拟现实（VR）和增强现实（AR）到先进游戏和交互媒体等应用中至关重要。传统的渲染技术常常在视觉保真度和计算效率之间进行权衡。《3D Gaussian Splatting for Real-Time Radiance Field Rendering》引入了一种突破性的方法，解决了上述问题，为实时场景中实现高质量渲染提供了新的途径。

原论文的创新之处在于其采用并增强了 3D 高斯飞溅技术，这是神经渲染领域的一次重大飞跃。该方法通过使用 3D 高斯模型来更准确、更灵活地描述场景几何和光照，超越了传统限制。这种方法的本质是更真实地模拟场景中光线的复杂交互，从而在不影响渲染速度的情况下提高整体视觉质量。

此外，原论文的方法不仅是对渲染流程的微小改进，而是对其进行了重新思考。通过整合交错优化和密度控制等先进技术，论文展示了在渲染效率方面的显著提升。这对于需要实时交互的应用尤为重要，其中延迟和响应速度与视觉质量同样重要。

这项工作的重要性不仅在于其技术优点。它代表了理论创新和实际应用性的结合，为实时渲染领域树立了新的标杆。这项研究的影响深远，可能会彻底改变我们在 VR 和 AR 中与数字环境互动的方式，并在游戏开发和交互媒体中开辟新的前沿。

图 1 基于 3D 高斯飞溅的实时高质量渲染

2 高斯飞溅方法

2.1 基于点的辐射场渲染

与神经辐射场中体渲染方程类似，基于点的辐射场渲染具有类似的公式。典型的基于神经点的方法通过混合与像素重叠的 N 个有序点来计算像素的颜色:

$$C=\sum_{i\in \mathrm{N}} c_i \alpha_i \prod_{j=1}^{i-1}(1-\alpha_j)$$

上式中 c_i 指每个点的颜色，α_i 是通过评估具有协方差 Σ 的 2D 高斯并乘以学习的不透明度得到的。

2.2 高斯飞溅定义与 2D 投影

该方法的核心在于使用 3D 高斯函数来表示场景中的辐射场。每个高斯函数 $G(x)=e^{-\frac{1}{2}(x)^T\Sigma^{-1}(x)}$ 被用来模拟场景中一个小区域的光照分布，其参数包括位置、大小、形状和颜色等。这种表示方法的优势在于其灵活性和表现力，能够更精确地捕捉场景中的细节和光照变化。

为了将 3D 高斯投影至 2D，该论文参考 Zwicker 等人将这种投影投影到图像空间的方法。给定一个相机投影矩阵 W，投影后的 2D 高斯的协方差矩阵 Σ' 计算方式如下：

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

其中 J 为投影变换仿射近似的雅可比矩阵。

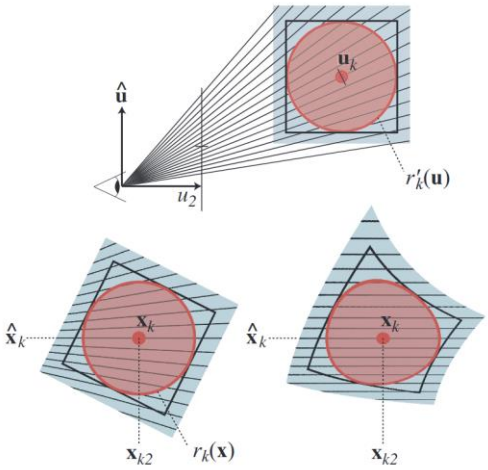

图 2.2 3D 高斯（上）仿射近似 2D 高斯（下左）真实投影 2D 高斯（下右）

2.3 自适应高斯控制

在训练优化过程中，3D 高斯实行的自适应高斯致密化方案。当重建不足时，即小尺度几何(黑色轮廓)未被充分覆盖时，我们克隆各自的高斯；当重建过度时，即如果小规模几何由一个大型 3D 飞溅表示，我们将其分成两部分。

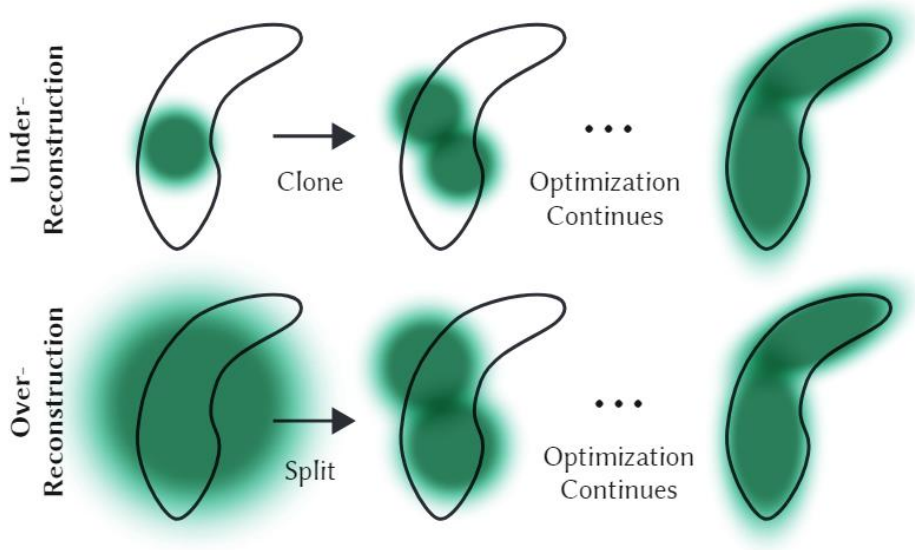

图 2.3 过小的高斯进行复制操作（上）过大的高斯进行分裂操作（下）

3 基于 3D 高斯飞溅的快速可微光栅化

该论文提出了一种快速可微分光栅化器，用于处理高斯表示的场景。该方法的目标是实现快速的整体渲染和快速的光栅化过程。为了实现这一目标，作者设计了一个基于条带的光栅化器，受到最近的软件光栅化方法的启发，通过对原始点进行预排序，避免了每个像素都进行排序的开销。这使得光栅化过程具有可微分性，并且可以在低额外内存消耗下实现高效的反向传播。此外，该方法在训练和渲染性能方面具有显著优势，可以处理具有任意深度复杂性的场景，而无需进行场景特定的超参数调整。

4 实验与评估

该论文使用 PyTorch 框架在 Python 中实现该方法，并为光栅化编写自定义 CUDA 内核，这些内核是先前方法的扩展版，并使用 NVIDIA CUB 排序例程进

行快速基数排序。

该论文在多个真实和合成场景数据集中进行了测试，并且选择的场景有非常不同的拍摄风格，涵盖了有限的室内场景和无界室外环境。

Dataset	Mip-NeRF360					
Method|Metric	$SSIM^{\uparrow}$	$PSNR^{\uparrow}$	$LPIPS^{\downarrow}$	Train	FPS	Mem
Plenoxels	0.626	23.08	0.463	25m49s	6.79	2.1GB
INGP-Base	0.671	25.30	0.371	5m37s	11.7	13MB
INGP-Big	0.699	25.59	0.331	7m30s	9.43	48MB
M-NeRF360	$0.792^{\dagger}$	$27.69^{\dagger}$	$0.237^{\dagger}$	48h	0.06	8.6MB
Ours-7K	0.770	25.60	0.279	6m25s	160	523MB
Ours-30K	0.815	27.21	0.214	41m33s	134	734MB

表 4 在渲染精度指标接近甚至超越目前 SOTA 方法，在渲染速度方面远超目前快速方法

该表显示，该方法的完全收敛的模型达到了与 SOTA Mip-NeRF360 方法相当的质量，有时甚至略好于 SOTA Mip-NeRF360 方法；请注意，在相同的硬件上，他们的平均训练时间为 48 小时，而 3D 高斯飞溅只需要 35-45min，并且 Mip-NeRF360 的渲染时间为 10s/frame，远低于实时渲染的标准。3D 高斯飞溅方法在 5-10m 的训练后实现了与 InstantNGP 和 Plenoxels 相当的质量，但额外的训练时间使其能够实现 SOTA 质量，这是其他快速方法无法做到的。

5 小结

总体而言，该方法通过结合 3D 高斯场景表示、交错优化和密度控制策略，以及快速可见性感知渲染算法，为实时辐射场渲染提供了一种高效且高质量的解决方案。这一方法不仅在理论上具有创新性，而且在实际应用中展现了巨大的潜力，特别是在虚拟现实、增强现实和游戏开发等领域。

然而，目前的基于 3D 高斯飞溅的神经渲染方法仍然存在不足之处：1. 没有反走样机制，依赖于训练数据的分布；2. 无法应用于动态场景或后续场景的编辑、动画等操作。

参考文献

[1]. Kerbl B, Kopanas G, Leimkühler T, et al. 3D Gaussian Splatting for Real-Time Radiance Field Rendering[J]. ACM Transactions on Graphics, 2023, 42(4).

[2]. Zwicker M, Pfister H, Van Baar J, et al. EWA volume splatting[C]//Proceedings Visualization, 2001. VIS'01. IEEE, 2001: 29-538.

[3]. Kopanas G, Leimkühler T, Rainer G, et al. Neural point catacaustics for novel-view synthesis of reflections[J]. ACM Transactions on Graphics (TOG), 2022, 41(6): 1-15.

[4]. Mildenhall B, Srinivasan P P, Tancik M, et al. Nerf: Representing scenes as neural radiance fields for view synthesis[J]. Communications of the ACM, 2021, 65(1): 99-106.

[5]. Merrill D G, Grimshaw A S. Revisiting sorting for GPGPU stream architectures[C]//Proceedings of the 19th international conference on Parallel architectures and compilation techniques. 2010: 545-546.