4D Gaussian Splatting for Real-Time Dynamic Scene Rendering

马文绘

摘要

近年来，Neural Radiance Field (NeRF) 在从多张照片或视频中生成新视角合成方面取得了巨大突破。然而，NeRF 的高视觉质量通常需要高昂的训练成本，尤其是动态场景的重建更是充满挑战。为了实现动态场景的高效实时渲染，4DGS 引入了四维高斯分布的场景表示，并提出了基于快速可微光栅化的训练和渲染方法。4DGS 使用动态点云来表示场景，在时间和空间域中实现高效建模，同时支持各向异性的光线投影绘制，达到了实时的新视角生成能力。本项目在 4DGS 的开源代码基础上，简化并改进了损失函数等模块，着重降低了点云文件大小。

关键词：动态场景重建；4D 高斯分布；快速渲染；时空场景表示

1 引言

随着三维重建技术的快速发展，传统的场景重建方法已经难以满足动态场景下实时高质量渲染的需求。近年来，基于神经网络的辐射场方法 (NeRF) [3] 在静态场景中取得了显著成功。然而，动态场景中时空变化的复杂性带来了额外的挑战。4D Gaussian Splatting for Real-Time Dynamic Scene Rendering (4DGS) [8] 提出了一种基于四维高斯分布的表示方法，它将时间作为额外的维度，结合快速可微光栅化技术，在不牺牲场景表示能力的前提下，实现了动态场景的高效训练与实时渲染。本项目的目标是复现并优化 4DGS 的关键技术，以更好地理解动态场景表示的核心原理。

2 相关工作

2.1 传统动态场景重建

早期动态场景重建主要依赖于光场技术和多视图立体 (Multiple View Stereo)。这些方法通常通过稠密采样或运动结构 (Structure-from-Motion) [6] 提取动态点云，结合几何优化生成场景。然而，受限于计算效率和数据存储，传统方法在复杂动态场景中往往表现不佳 [4]。

2.2 新视角合成

早期的方法利用基于图像的渲染技术和代理几何图形，从最开始开始图像中获取信息得到新的视图和新视角 [1]。[2] 利用深度图来混合源视图中的像素。[7] 从点云中提取潜在特征，

用于新视角图片的合成。

2.3 动态 NeRF 和时空辐射场

动态场景的神经表示主要扩展了静态 NeRF 的技术框架，例如 Dynamic-NeRF [5] 和 T-NeRF [9] 等。这些方法通过添加时间编码或动态体素网格来建模时空场景。然而，它们通常依赖于复杂的神经网络计算，训练时间和推理效率难以满足实时需求。

动态 NeRF 方法旨在表示随时间变化的场景，通常通过引入变形场来捕捉场景中各点的时间变化。

2.4 基于高斯分布的动态场景表示

与基于体素或网格的表示方法不同，3DGS 和 4DGS 使用高斯分布表示动态点云，结合光栅化技术在 GPU 上实现了高效渲染。相比于传统 NeRF，4DGS 在动态场景中表现出更高的灵活性和实时性，尤其在时空域中的表示能力更加突出。

3 基础知识

3D 高斯点染是一种将场景显式表示为三维高斯分布的技术。每个三维高斯分布由中心点 $\mathbf{X}$ 和协方差矩阵 Σ 表征，其数学表达式为：

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

为了便于优化，协方差矩阵 Σ 可分解为缩放矩阵 $\mathbf{S}$ 和旋转矩阵 $\mathbf{R}$ 的形式：

$$\Sigma = \mathbf{RSS}^T\mathbf{R}^T. \tag{2}$$

在渲染新视角时，利用差分点染技术将三维高斯分布投影到摄像机平面上。通过视图变换矩阵 $\mathbf{W}$ 和仿射近似的雅可比矩阵 $\mathbf{J}$，可以计算摄像机坐标系下的协方差矩阵 Σ'：

$$\Sigma' = \mathbf{JW}\Sigma\mathbf{W}^T\mathbf{J}^T. \tag{3}$$

每个高斯分布的属性包括位置 $\mathbf{X} \in \mathbb{R}^3$、由球谐函数系数表示的颜色 $\mathbf{C} \in \mathbb{R}^k$（$k$ 为球谐函数数量）、不透明度 $\alpha \in \mathbb{R}$、旋转因子 $r \in \mathbb{R}^4$ 和缩放因子 $s \in \mathbb{R}^3$。对于每个像素，所有高斯分布的颜色和不透明度通过以下公式进行混合：

$$C = \sum_{i \in N} c_i \alpha_i \prod_{j=1}^{i-1}(1 - \alpha_j), \tag{4}$$

其中 c_i 和 α_i 分别表示高斯分布 G 的颜色和密度，并乘以可优化的不透明度和球谐颜色系数。

4 本文方法

4.1 本文方法概述

4D Gaussian Splatting (4D-GS) 提出了一种高效的动态场景表示方法，通过将场景建模为 3D 高斯的变形场，并结合时空特征编码，解决了动态场景渲染中的高效性和存储问题。4D-GS 的整体框架如图 1 所示：

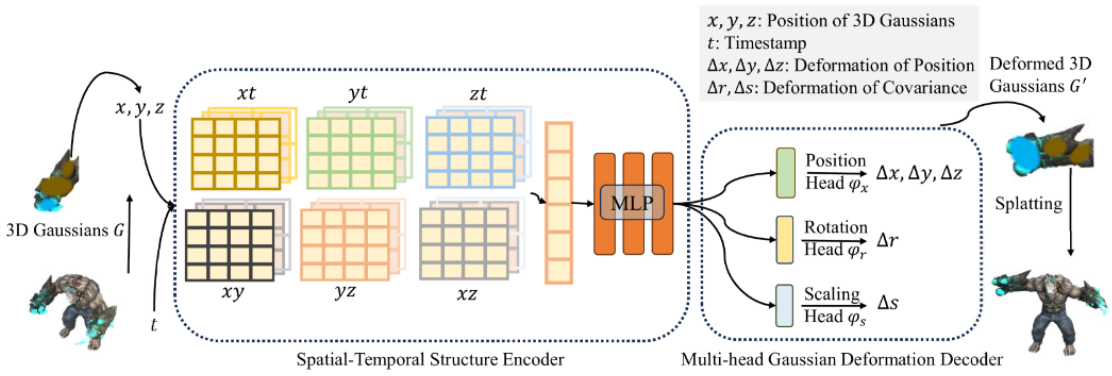

图 1. 4D-GS 方法示意图

在该框架中，通过对 3D 高斯点云的变形场进行建模，4D-GS 在每个时间戳生成变形后的高斯点云，并通过快速可微的光栅化算法实现高效的动态场景渲染。

4.2 四维高斯分布

四维高斯分布扩展了传统三维高斯的定义，将时间作为额外维度，并通过一个高效的高斯变形场网络建模动态变化。其数学定义为：

$$G_t(x) = \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma_t^{-1}(x-\mu)\right)$$

其中：

- $x \in \mathbb{R}^4$ 表示动态场景中的四维点；
- μ 为高斯的中心点，包含时空信息；
- Σ_t 为协方差矩阵，表示椭球的形状和时间演化特性。

4.3 动态点云的投影渲染

4D-GS 采用可微光栅化方法将动态点云投影到图像平面。在时间维度上，通过仿射近似优化了投影矩阵，使其支持高效的动态渲染。对于每个时间戳 t，高斯点云的变形可表示为：

$$\mathcal{G}' = \mathcal{G} + \Delta\mathcal{G}$$

其中 $\Delta\mathcal{G}$ 是通过高斯变形场网络 $\mathcal{F}(\mathcal{G}, t)$ 预测的变形量。

为了捕捉动态场景中的复杂运动信息，$\Delta\mathcal{G}$ 的计算依赖于高斯变形场网络 $\mathcal{F}$。该网络输入时间信息 t 和初始高斯集合 $\mathcal{G}$，输出相应的高斯变形：

$$\Delta\mathcal{G} = \mathcal{F}(\mathcal{G}, t).$$

在此过程中，系统使用了一个时空特征编码器 $\mathcal{H}$ 来提取 3D Gaussians 的空间和时间特征，从而更好地描述动态场景中的局部几何和运动特性。具体来说，时空特征表示为：

$$f_d = \mathcal{H}(\mathcal{G}, t).$$

随后，通过多头高斯变形解码器 $\mathcal{D}$ 将时空特征 f_d 解码为每个高斯的具体变形量：

$$\Delta\mathcal{G} = \mathcal{D}(f_d).$$

最终，我们可以通过差异化光栅化过程 $\mathcal{S}$ 将变形后的高斯 $\mathcal{G}'$ 投影到图像平面，生成时间戳 t 对应的渲染图像。

值得注意的是，该框架保持了差异化光栅化的高效性和准确性，并通过引入时间维度，实现了对动态场景的高质量建模。

4.4 高斯变形场的网络

学习高斯变形场的网络由两个核心模块组成：一个高效的时空特征编码器 $\mathcal{H}$ 和一个多头高斯变形解码器 $\mathcal{D}$。

4.4.1 时空特征编码器

在动态场景中，邻近的 3D Gaussians 通常具有相似的几何和运动特性。因此，4DGS 作者设计了一个高效的时空特征编码器 $\mathcal{H}$，以捕捉这些特征并减少冗余信息。编码器结合了多分辨率 HexPlane 模块 $R(i,j)$ 和一个小型 MLP ϕ_d，在空间和时间维度上对高斯进行联合编码。与传统的 4D 体素表示不同，作者采用了 4D K-Planes 模块，将高维体素分解为六个多分辨率平面（即 (x,y)、(x,z)、(y,z)、(x,t)、(y,t) 和 (z,t)）。这种方法不仅显著降低了内存消耗，还能更高效地捕获局部运动信息。

对于每个高斯点，编码器提取其在六个平面上的特征，具体计算公式如下：

$$f_h = \bigcup_{l} \prod \text{interp}(R_l(i,j)), \quad (i,j) \in \{(x,y),(x,z),(y,z),(x,t),(y,t),(z,t)\},$$

其中 $f_h \in \mathbb{R}^{h \cdot l}$ 表示高斯点的多平面特征，interp 为双线性插值，用于在平面网格中查询点的特征值。最终，特征通过一个小型 MLP ϕ_d 进行融合，生成紧凑的时空特征：

$$f_d = \phi_d(f_h).$$

4.4.2 多头高斯变形解码器

在获得时空特征 f_d 后，系统使用多头解码器 $\mathcal{D}$ 预测每个高斯点的几何和运动参数变化。解码器由三个子模块组成，分别用于预测位置、旋转和缩放的变形量：

$$\Delta\mathcal{X} = \phi_x(f_d), \quad \Delta r = \phi_r(f_d), \quad \Delta s = \phi_s(f_d).$$

最终，变形后的高斯参数为：

$$(\mathcal{X}', r', s') = (\mathcal{X} + \Delta\mathcal{X}, r + \Delta r, s + \Delta s).$$

通过结合这些变形量，作者可以生成动态场景中每个时间戳的高斯表示：

$$\mathcal{G}' = \{\mathcal{X}', s', r', \sigma, \mathcal{C}\}.$$

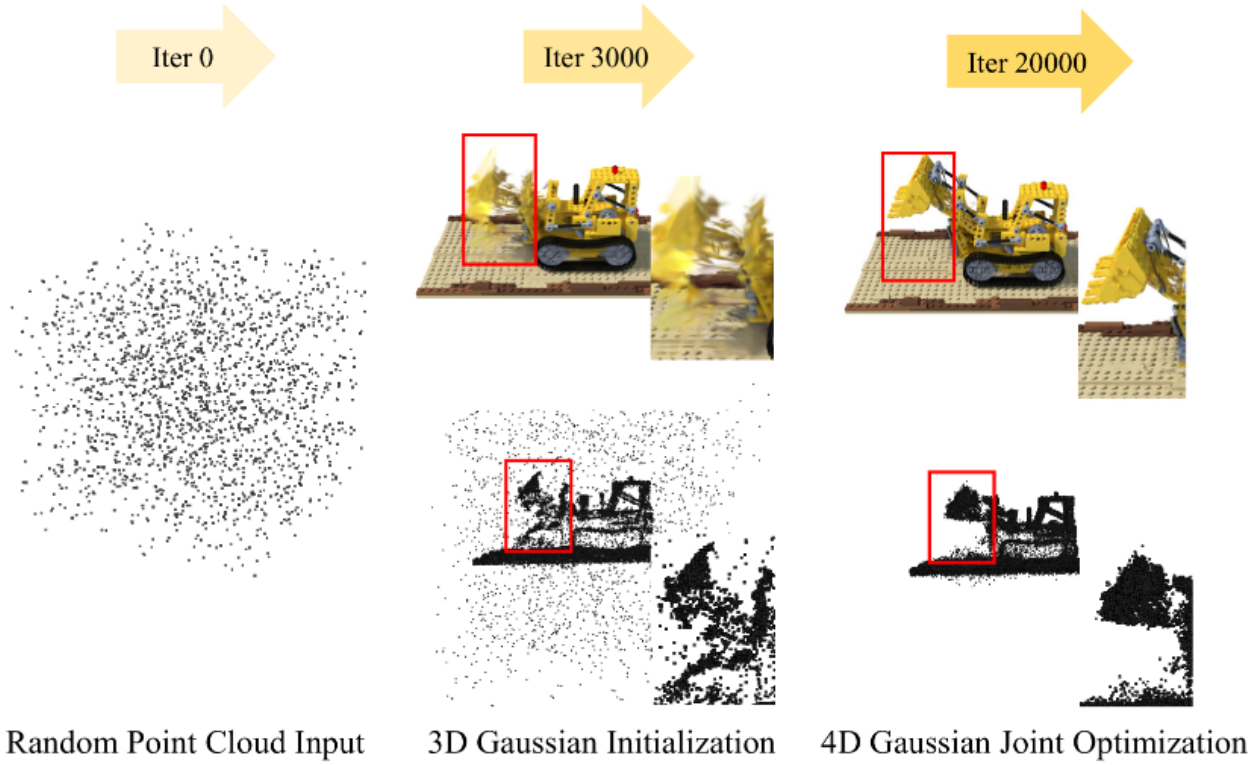

图 2. 优化过程的图示

4.5 优化方式

4.5.1 基于 3D 高斯初始化

初始化阶段，作者首先利用 SfM（Structure from Motion）生成的点云来初始化静态 3D Gaussians $\mathcal{G}$。在训练的初始 3000 次迭代中，仅优化静态高斯集合以获得稳定的基础表示：

$$\hat{I} = \mathcal{S}(M, \mathcal{G}),$$

此阶段的渲染结果不包含时间维度，主要用于为后续的动态优化提供可靠的起点。

4.5.2 损失函数

为了保证高斯变形的准确性和最终渲染图像的质量，作者设计了一个多目标损失函数，包括颜色损失和基于网格的总变分正则化损失。具体公式如下：

$$\mathcal{L} = |\hat{I} - I| + \mathcal{L}_{tv},$$

其中，$\mathcal{L}_{tv}$ 用于约束高斯点的空间分布连续性，减少重叠和过密现象，从而提高最终渲染图像的视觉质量。

原作者提出的优化流程能够快速收敛，并在保持高质量渲染效果的同时，实现动态场景的实时生成。

5 复现细节

5.1 与已有开源代码对比

本次研究的复现工作基于 4D Gaussian Splatting (3D-GS) 的官方开源代码，并对其进行了扩展和改进。具体来说，我们引用了以下部分代码及其用途：

- **快速可微光栅化器**：直接使用了官方实现的 CUDA 光栅化模块，用于高效处理点云的投影和渲染。这一模块在性能优化和 GPU 兼容性方面表现优异。
- **数据预处理管道**：参考了官方代码中的点云初始化模块，用于从 SfM (Structure from Motion) 提取稀疏点云，并将其转换为 3D 高斯。
- **4D 高斯分布和时空变形场**：一个基于时间维度扩展的高斯变形场网络，实现了动态场景中 3D 高斯点的时空变换和优化。

在此基础上，我们进行了以下改进和创新：

- **改进的 L1 损失函数**：在渲染质量略微下降的情况下，大幅减少了点云数量和存储需求，提升了训练和渲染效率。
- **模块化和简化代码结构**：使用 PyTorch 重写了核心训练代码，使其更加易读和易于修改，从而便于其他研究人员进行二次开发。

5.2 实验环境搭建

为了确保复现结果的准确性和可重复性，我们严格按照以下步骤搭建了实验环境：

- **硬件环境**：
  - GPU：NVIDIA V100 (32GB 显存)。
  - CPU：Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz。
  - 内存：251GB RAM。
- **软件环境**：
  - 操作系统：Ubuntu 18.04.3 LTS。
  - Python 版本：3.8。
  - CUDA 版本：11.6。
  - PyTorch 版本：1.13.1。

5.3 创新点与改进

本研究的主要创新点在于对原始的 L1 损失函数进行了改进，从而显著减少了模型在训练和渲染时所需的点数量。这一改进不仅降低了存储需求，还提高了计算效率，仅以较小的 SSIM 和 PSNR 损失为代价。

5.4 改进的 L1 损失函数

在原始 4DGS 方法中，L1 损失函数用于衡量预测图像与真实图像之间的像素差异。然而，原始的 L1 损失函数在面对局部细节和梯度变化时，可能会导致较高的点云密度要求。为了解决这一问题，我们提出了一种改进的平滑 L1 损失函数，并引入了动态权重和数值稳定性调整，旨在提高模型训练效率并降低点云密度。

改进后的平滑 L1 损失函数的数学表达式如下：

$$\text{L1 Loss}(x, y, \beta, \alpha, \gamma, \epsilon) = \begin{cases} \frac{\alpha}{1+\log(1+|x-y|+\epsilon)} \cdot \frac{(x-y)^2}{2\beta}, & \text{if } |x-y| < \beta, \\ \gamma \cdot (1 - e^{-|x-y|}) \cdot (|x-y| - 0.5 \cdot \beta), & \text{otherwise,} \end{cases} \tag{5}$$

其中：

- β 是控制平滑区域与绝对误差切换点的超参数；
- α 和 γ 分别表示小误差（L2 区域）和大误差（L1 区域）的动态权重；
- ϵ 是一个小常数，用于提高数值稳定性，避免计算中的除零问题。

5.4.1 改进点分析

1. 动态权重调整：我们通过引入 α 和 γ 的动态变化，使得在误差较大时，模型倾向于对大误差区域施加更高权重，从而降低点云数量。2. 数值稳定性增强：在计算绝对误差的过程中加入 ϵ，避免梯度爆炸或计算异常情况。3. 点云优化效果：实验表明，在 Bouncing Balls 数据集中，我们将原始模型所需的 27,674 个点（点云文件大小 6.54 MB），显著减少至仅需 3,034 个点（点云文件大小 736 KB），并且仅牺牲了极小的 SSIM 和 PSNR 指标。

5.4.2 优点总结

- 在误差较小时，通过动态调整 α，平滑优化过程，提高模型对局部细节的表现能力；
- 在误差较大时，通过动态调整 γ，增强模型对大范围几何变化的适应能力；
- 显著减少了点云的数量及其存储大小，同时保持了渲染质量。

6 实验结果分析

我们在多个数据集上测试了改进后的 L1 损失函数对模型性能的影响，并与原始方法进行了对比。以下以 **BouncingBalls** 数据集为例，展示改进的显著效果。

表 1. BouncingBalls 数据集上不同方法的对比实验

方法	**点数 ↓**	**文件大小 (MB)↓**	**SSIM↑**	**PSNR (dB)↑**	**LPIPS-vgg↓**	**LPIPS-alex↓**
原始 4DGS	27674	6.54	0.9994	40.67	0.014	0.005
改进的 4DGS	**3034**	**0.736**	0.9899	38.62	0.032	0.015

从表 1 中可以看出：

- 改进后的方法使点数从 27674 大幅减少到 3034，减少了约 89%；
- 点云文件大小从 6.54MB 减少到 736KB，存储需求显著降低；
- SSIM 仅略微下降约 1%，而 PSNR 仅降低 2dB，仍保持较高的渲染质量。

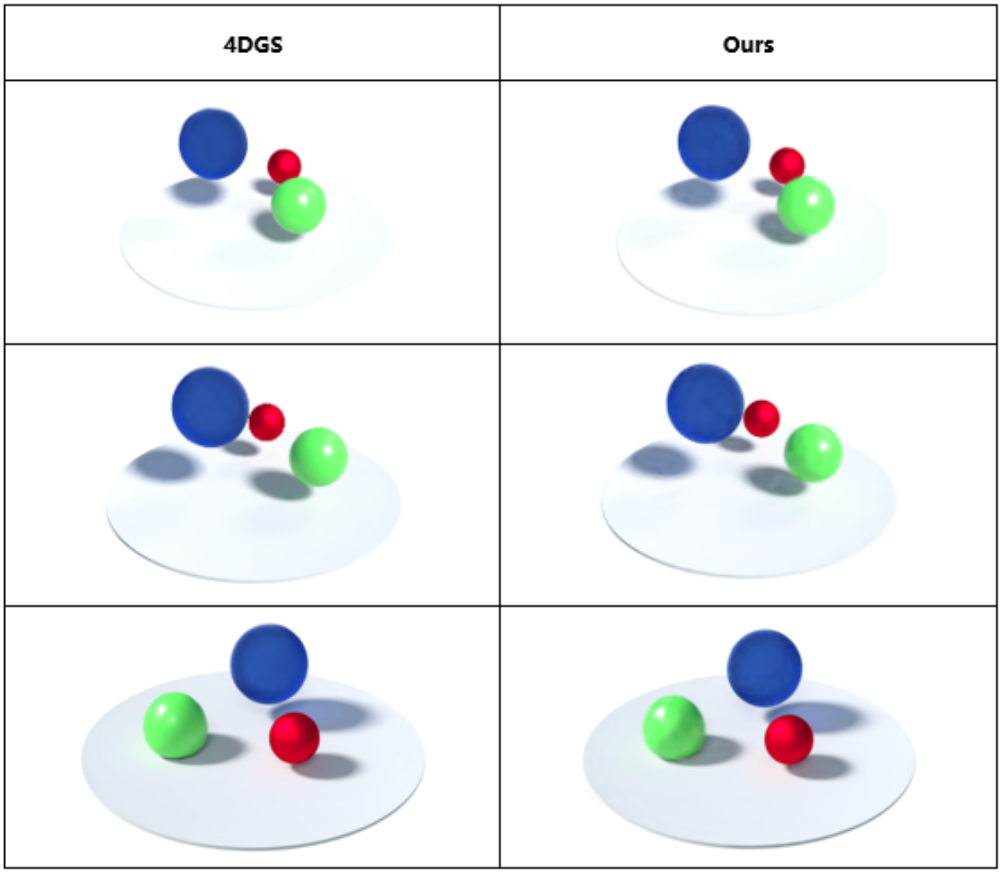

图 3. BouncingBalls 数据集渲染结果对比：左为原始方法，右为改进方法

根据图 3 的对比渲染结果，我们可以得出以下几点结论：

- **视觉质量保持一致性**：尽管改进后的方法使用了更少的点数，右侧的渲染结果在整体视觉质量上与左侧的原始方法保持一致，仅在局部细节上略有差异。这表明改进的 L1 损失函数在减少点数的同时，能够有效维护视觉效果。
- **背景与动态细节的表现**：在复杂的动态场景中，如 BouncingBalls 数据集，球体的动态轨迹以及背景的清晰度在改进方法中依然表现优异。这说明改进后的损失函数能够在减少点云密度的同时保持动态场景的关键信息。
- **资源优化的显著性**：从渲染图可以看出，虽然改进后的方法使用了更少的资源，但在视觉效果上几乎没有明显下降。这表明我们的改进方法能够显著降低存储和计算成本，为更大规模场景或实时应用提供了可能性。
- **适应性与鲁棒性**：右侧图像在场景复杂性和多样性上的表现表明，改进的损失函数能够适应复杂动态场景中的不同运动模式，同时保持较高的 SSIM 和 PSNR 水平。

7 总结与展望

本文针对 4DGS 方法在动态场景渲染中的点云密度高、存储需求大等问题，提出了一种基于改进 L1 损失函数的优化方案。在实验中，我们以 BouncingBalls 数据集为例，证明了改进方法能够在显著降低点云数量和存储需求的同时，仅略微牺牲渲染质量。提出了一种动态加权的平滑 L1 损失函数，有效地减少了点云的数量需求，使存储文件大小大幅缩小约 89%；保持较高 SSIM 和 PSNR 的前提下，实现了更高的资源效率，为实时动态场景渲染提供了新的思路；通过本研究，我们验证了损失函数设计在动态场景渲染优化中的重要性。尽管改进方法取得了一定的效果，但在实际应用中仍然存在一些不足。当前的实验主要集中在 BouncingBalls 数据集等较为规则的动态场景，对于更加复杂、不规则和更大规模的场景需要进一步验证方法的适用性。改进的 L1 损失函数中引入了多个动态权重参数（如 α 和 γ），如何针对不同数据集自动优化这些参数仍需深入研究。

本研究在动态场景渲染领域取得了较为显著的优化效果，为 4DGS 方法的进一步发展奠定了基础。未来，我们期待在更广泛的数据集和实际应用场景中验证和推广本方法，同时解决现有的局限性，为动态场景渲染技术的发展做出更多贡献。

参考文献

[1] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432, 2001.

[2] Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM transactions on graphics (TOG)*, 32(3):1–12, 2013.

[3] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[4] Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. Reducing the memory footprint of 3d gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1):1–17, 2024.

[5] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021.

[6] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM siggraph 2006 papers*, pages 835–846. 2006.

[7] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. Synsin: End-to-end view synthesis from a single image. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7467–7477, 2020.

[8] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20310–20320, 2024.

[9] Chengxuan Zhu, Renjie Wan, and Boxin Shi. Neural transmitted radiance fields. *Advances in Neural Information Processing Systems*, 35:38994–39006, 2022.