

4D Gaussian Splatting for Real-Time Dynamic Scene Rendering

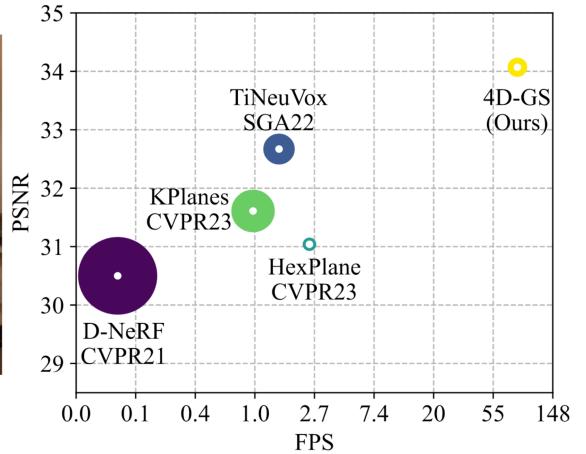


Figure 1. Our method achieves real-time rendering[‡] for dynamic scenes at high image resolutions while maintaining high rendering quality. The right figure is tested on synthetic datasets, where the radius of the dot corresponds to the training time. “Res”: resolution.

[‡]The rendering speed not only depends on the image resolution but also the number of 3D Gaussians and the scale of deformation fields which are determined by the complexity of the scene.

Introduction

新视点合成 (Novel View Synthesis, NVS) 是 3D 视觉领域的重要任务，广泛应用于 虚拟现实 (VR)、增强现实 (AR) 和 电影制作 等场景。NVS 的目标是从多个 2D 图像中准确建模场景，从任意视角或时间点渲染图像。

动态场景的渲染尤为重要且具有挑战性，因为需要处理 时空稀疏输入 下的复杂运动建模。

近年来，**NeRF** 提出通过隐函数表示场景，在新视点图像合成方面取得了巨大成功。其主要基于体积渲染技术：

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t) \cdot \sigma(t) \cdot c(t) dt,$$

其中：

- $C(\mathbf{r})$ 是沿视线 \mathbf{r} 的颜色值；
- $T(t)$ 表示光穿透场景的传输函数；
- $\sigma(t)$ 为场景的密度；
- $c(t)$ 为颜色值。

然而，原始的 **NeRF** 存在训练和渲染成本过高的问题。尽管一些变体（例如 Instant-NGP）显著减少了训练时间，但渲染延迟问题仍未完全解决。

为了加速动态场景渲染，**3D Gaussian Splatting (3D-GS)** 提出用 3D 高斯点表示场景，采用高效的可微分点云投影方法直接在 2D 平面渲染：

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

其中：

- \mathbf{x} 表示点的坐标；
- μ 为高斯中心；
- Σ 为协方差矩阵。

尽管 3D-GS 能实时渲染静态场景，其在动态场景上的扩展仍面临存储和计算效率的挑战。为此，本文提出 **4D Gaussian Splatting (4D-GS)**，将动态场景压缩为更紧凑的表征，包含以下创新：

1. 构建高效的 **高斯变形场网络 (Gaussian Deformation Field Network)**，捕获高斯点的运动和形状变化；
2. 使用多分辨率编码方法连接临近 3D 高斯点，生成丰富的 3D 特征；
3. 实现高分辨率动态场景的实时渲染，性能超越现有的 SOTA 方法。

本文的贡献总结如下：

- 提出一种高效的 4D-GS 框架，包含高斯运动和形状变化的建模；
- 开发了一种空间-时间结构编码器，高效提取时空特征；
- 实现每秒 82 帧（800×800 分辨率）的实时渲染，并支持 4D 场景的编辑与跟踪。

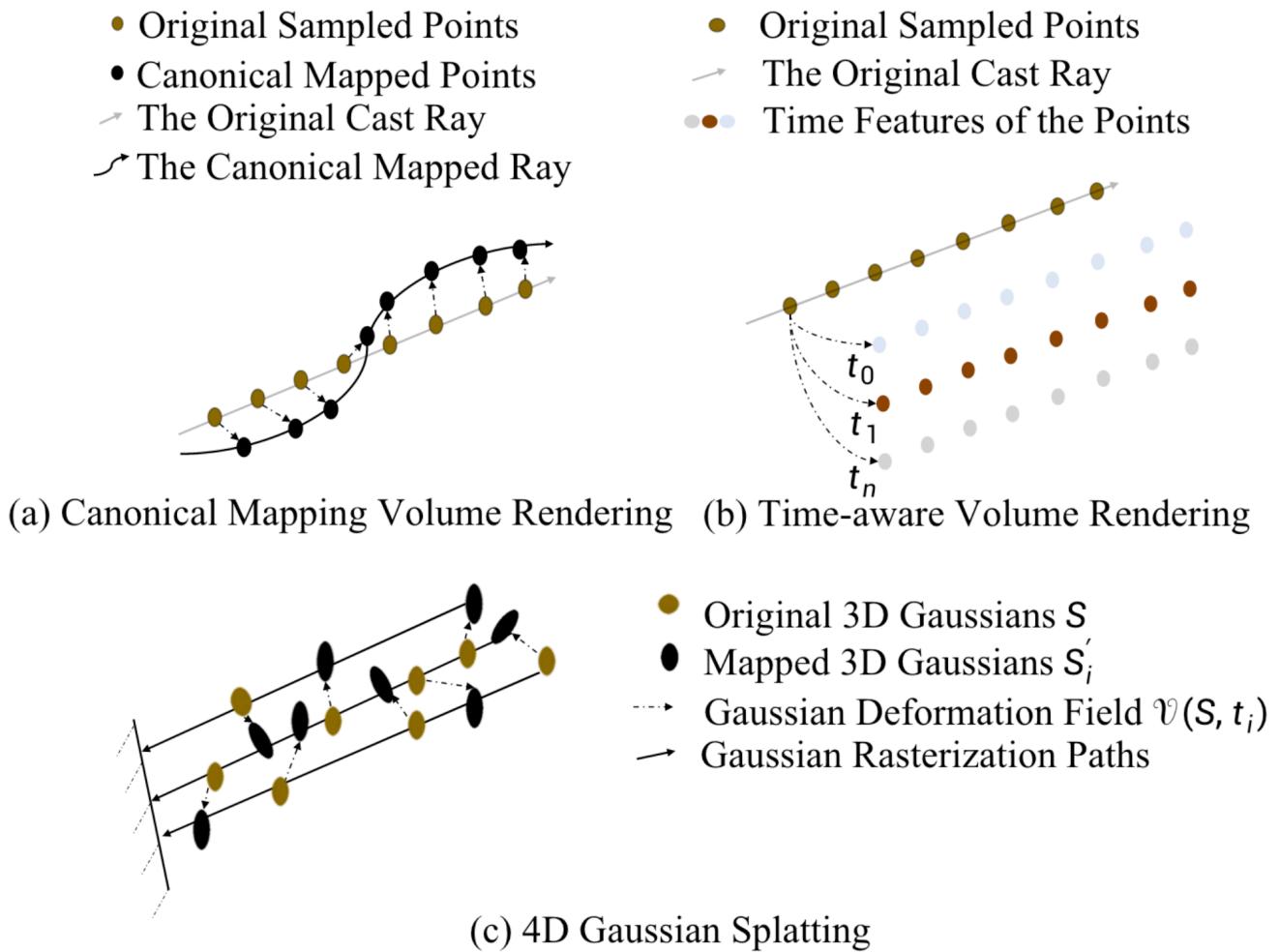


Figure 2. Illustration of different dynamic scene rendering methods. (a) Points are sampled in the casted ray during volume rendering. The point deformation fields proposed in [6, 28] map the points into a canonical space. (b) Time-aware volume rendering computes the features of each point directly and does not change the rendering path. (c) The Gaussian deformation field converts original 3D Gaussians into another group of 3D Gaussians with a certain timestamp.

Related Works

本部分对动态 **NeRFs** 的发展进行综述，并介绍基于点云的神经渲染算法的研究进展。

1. Dynamic Neural Rendering

动态场景的新视点合成要求对场景中的复杂运动进行建模。经典的 **NeRF** 方法通过隐函数表征场景，但主要针对静态场景进行优化。随着动态场景需求的增加，以下几种关键方法被提出：

1. Canonical Mapping Volume Rendering

- 通过变形网络 ϕ_t 将采样点 (\mathbf{x}, t) 映射到规范空间：
$$\Delta \mathbf{x} = \phi_t(\mathbf{x}, t),$$
- 在规范空间中回归体积密度 σ 和视点相关的 RGB 值 \mathbf{c} ：
$$\mathbf{c}, \sigma = \text{NeRF}(\mathbf{x} + \Delta \mathbf{x}).$$
- 这种方法的渲染公式：
$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t) \cdot \sigma(t) \cdot c(t) dt.$$

2. Time-Aware Volume Rendering

- 不改变采样路径，直接在时间维度上对每个点的特征进行建模。

上述方法尽管能够加速动态场景的训练，但在单目输入场景中实现实时渲染仍然具有挑战性。

2. Neural Rendering with Point Clouds

点云被广泛用于 3D 场景表示。基于点云的神经渲染方法主要包括：

1. Differentiable Point Rendering

- 基于可微分点投影技术，将 3D 点投影到 2D 平面。例如，3D 高斯点的表示为：

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

其中 μ 为点的中心， Σ 为协方差矩阵。

2. 3D Gaussian Splatting (3D-GS)

- 将场景表示为 3D 高斯点，通过高效的可微分点云投影替代原始 NeRF 的体积渲染。

- 使用 Jacobian 矩阵 \mathbf{J} 将高斯点投影到相机平面：

$$\Sigma' = \mathbf{J} \mathbf{W} \Sigma \mathbf{W}^T \mathbf{J}^T,$$

其中 \mathbf{W} 为视图变换矩阵。

尽管 3D-GS 在静态场景上取得显著效果，但其扩展至动态场景时存储和内存消耗随时间线性增长：

$$O(N \cdot t),$$

其中 N 为高斯点数， t 为时间步数。

3. Dynamic3DGS

- 引入时间高斯分布，将原始 3D 高斯提升到 4D，但该方法局限于局部时空范围。
-

3. Our Contributions

相比于上述方法，本文提出的 **4D Gaussian Splatting (4D-GS)** 在存储和计算效率上有显著优势：

- 内存复杂度仅依赖于高斯点数和高斯变形场网络参数：
 $O(N + F)$ ，
其中 F 表示网络参数的规模。
- 提出紧凑的空间-时间结构编码器，捕获动态场景中高斯点的运动和形状变化。

通过与现有方法的对比，本文方法在动态场景的训练效率和实时渲染方面表现优越。

Preliminary

本部分回顾 **3D Gaussian Splatting (3D-GS)** 的表示与渲染过程，以及动态 NeRFs 的公式化建模。

1. 3D Gaussian Splatting (3D-GS)

3D-GS 是一种基于点云的显式 3D 场景表示方法，每个 3D 高斯点通过以下公式定义：

$$G(\mathbf{x}) = e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)}, \quad (1)$$

其中：

- $\mathbf{x} \in \mathbb{R}^3$ 是点的坐标；
- μ 是高斯中心；
- Σ 是协方差矩阵，用于定义点的形状和尺度。

为了便于优化，协方差矩阵 Σ 可以分解为以下形式：

$$\Sigma = \mathbf{R} \mathbf{S} \mathbf{S}^T \mathbf{R}^T, \quad (2)$$

其中：

- \mathbf{R} 是旋转矩阵；
- \mathbf{S} 是缩放矩阵。

在渲染过程中，使用可微分投影方法将 3D 高斯点映射到相机平面。通过视图变换矩阵 \mathbf{W} 和仿射近似的 Jacobian 矩阵 \mathbf{J} ，可以将协方差矩阵转换为相机坐标系下的协方差：

$$\Sigma' = \mathbf{J} \mathbf{W} \Sigma \mathbf{W}^T \mathbf{J}^T. \quad (3)$$

每个像素颜色由覆盖该像素的所有高斯点的颜色和透明度进行加权融合：

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

其中：

- c_i 是高斯点的颜色；
 - α_i 是高斯点的不透明度；
 - N 是像素内的高斯点数量。
-

2. Dynamic NeRFs with Deformation Fields

动态 NeRF 模型通过对时空采样点 (\mathbf{x}, t) 的建模，生成视点相关的 RGB 和体积密度值：

$$\mathbf{c}, \sigma = M(\mathbf{x}, t), \quad (5)$$

其中 M 是从空间 (\mathbf{x}, t) 映射到 (\mathbf{c}, σ) 的隐函数。

(1) Canonical Mapping

采样点首先被变形网络 ϕ_t 映射到规范空间：

$$\Delta \mathbf{x} = \phi_t(\mathbf{x}, t), \quad (6)$$

然后在规范空间内使用 NeRF 模型回归 RGB 和密度：

$$\mathbf{c}, \sigma = \text{NeRF}(\mathbf{x} + \Delta\mathbf{x}). \quad (7)$$

(2) Rendering Formula

渲染公式表示为：

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t) \cdot \sigma(t) \cdot \mathbf{c}(t) dt, \quad (8)$$

其中：

- $T(t)$ 是光传输函数，表示光从起点到 t 的衰减。

上述方法对复杂动态场景建模具有一定效果，但存在训练时间长和渲染延迟的问题。

3. Summary

3D-GS 的高效性在于其显式的点云表示，但存储成本随时间增长线性扩展。而动态 NeRFs 尽管能够建模复杂动态场景，但在单目输入和稀疏时间采样下，实时性仍存在不足。

本文提出的 **4D Gaussian Splatting** 结合两者优点，通过 **高斯变形场网络** 建模动态场景，显著提高存储和渲染效率。

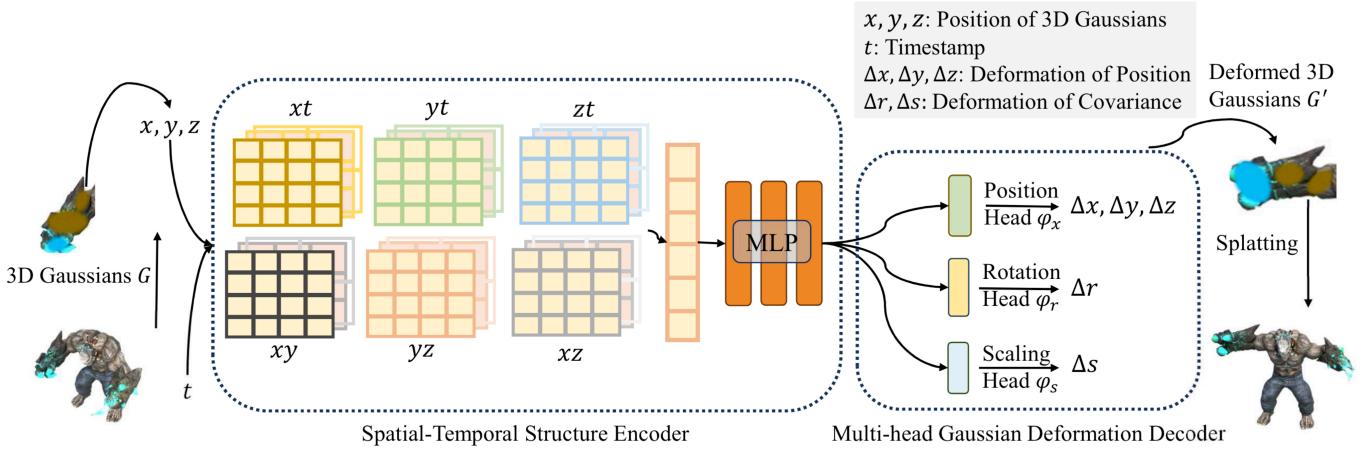


Figure 3. The overall pipeline of our model. Given a group of 3D Gaussians \mathcal{G} , we extract the center coordinate of each 3D Gaussian \mathcal{X} and timestamp t to compute the voxel feature by querying multi-resolution voxel planes. Then a tiny multi-head Gaussian deformation decoder is used to decode the feature and get the deformed 3D Gaussians \mathcal{G}' at timestamp t . The deformed Gaussians are then splatted to the rendered image.

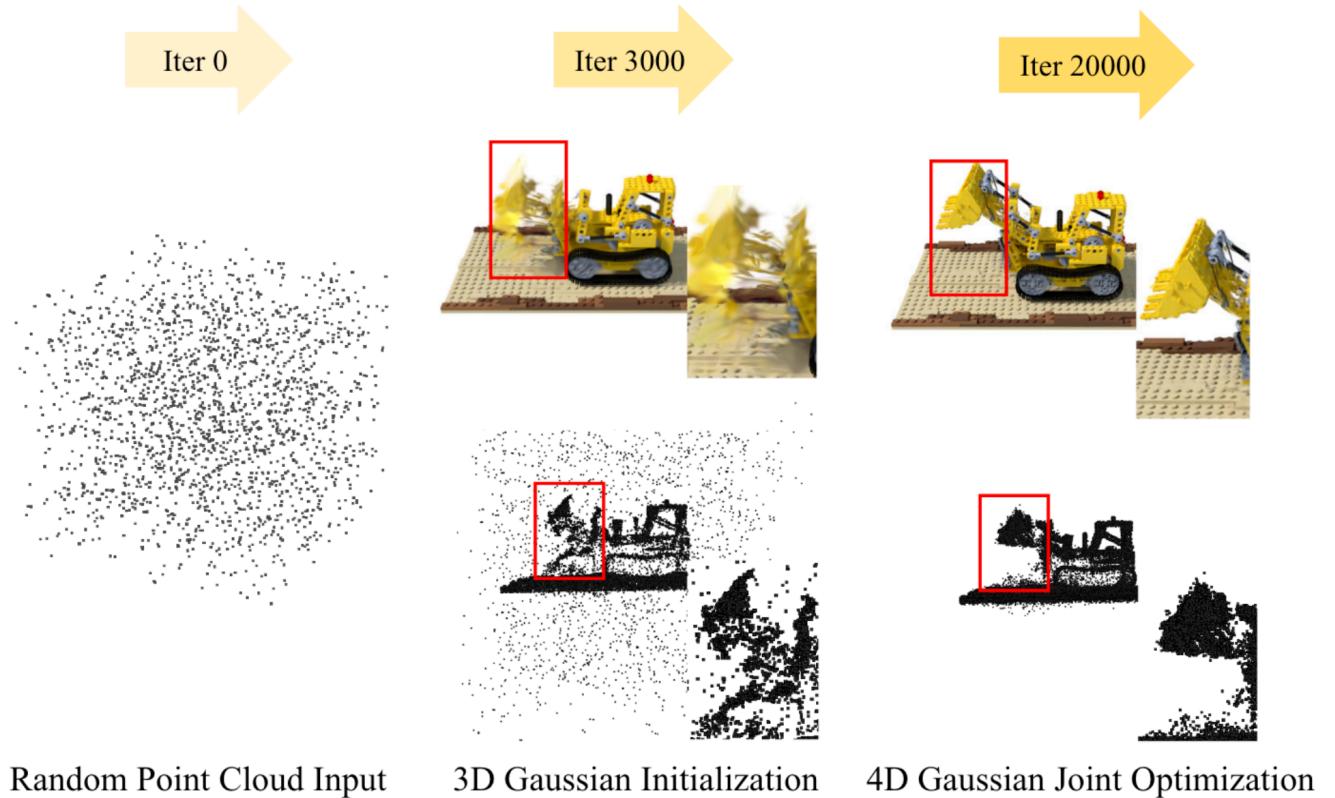


Figure 4. Illustration of the optimization process. With static 3D Gaussian initialization, our model can learn high-quality 3D Gaussians of the motion part.

Method

本文提出了一种用于动态场景实时渲染的 **4D Gaussian Splatting (4D-GS)** 框架，包括高效的高斯变形场网络及其优化过程。以下对各小节进行详细分解。

4.1 4D Gaussian Splatting Framework

框架描述：

4D-GS 的核心思想是使用单一的规范 3D 高斯集合，通过高斯变形场网络（Gaussian Deformation Field Network）将其变换为任意时间截下的形状和位置。

主要步骤：

1. 输入：

- 相机视图矩阵 $M = [\mathbf{R}, \mathbf{T}]$;
- 时间截 t ;
- 3D 高斯集合 G 。

2. 高斯变形：

- 通过高斯变形场网络预测 3D 高斯的变形 ΔG :

$$\Delta G = F(G, t),$$

其中 F 是高斯变形场网络。

3. 渲染:

- 对变形后的高斯集合 $G' = G + \Delta G$ 进行可微分投影 (Differentiable Splatting) :

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

其中 S 表示投影操作, \hat{I} 是渲染图像。

框架特点:

- 通过时间和空间的特征编码, 构建紧凑的 4D 表征。
- 渲染过程完全可微, 有助于优化。

4.2 Gaussian Deformation Field Network

网络结构:

高斯变形场网络由两个主要模块组成:

(1) Spatial-Temporal Structure Encoder

目标:

提取 3D 高斯点在时空上的特征, 生成高效的时空编码。

方法：

- 使用多分辨率体素平面编码（Multi-resolution HexPlane Encoding）。
- 将 4D 体素分解为 6 个 2D 平面：
 $R_l(i, j), \quad (i, j) \in \{(x, y), (x, z), (y, z), (x, t), (y, t), (z, t)\}$,
其中 l 表示不同分辨率。

公式：

- 体素特征通过双线性插值计算：
 $f_h = \bigcup_l \prod \text{interp}(R_l(i, j)),$
其中 interp 表示插值操作。
- 然后通过一个小型 MLP 进一步融合特征：
 $f_d = \phi_d(f_h),$
其中 ϕ_d 是用于融合的 MLP。

结果：

- 编码后的特征 f_d 捕捉了 3D 高斯点的空间和时间特性。

(2) Multi-head Gaussian Deformation Decoder

目标：

根据编码特征 f_d , 预测 3D 高斯的变形量。

方法：

- 使用独立的 MLP 预测位置、旋转和缩放的变形：

- 位置变形：

$$\Delta \mathbf{x} = \phi_x(f_d),$$

- 旋转变形：

$$\Delta r = \phi_r(f_d),$$

- 缩放变形：

$$\Delta s = \phi_s(f_d).$$

结果：

- 变形后的高斯点属性：

$$\mathbf{x}', r', s' = \mathbf{x} + \Delta \mathbf{x}, r + \Delta r, s + \Delta s.$$

- 最终的高斯集合：

$$G' = \{\mathbf{x}', r', s', \sigma, C\}.$$

4.3 Optimization

优化目标：

通过对 4D 高斯的训练，使其准确表示动态场景，同时保持高效渲染。

(1) 3D Gaussian Initialization

- 使用结构化运动 (SfM) 初始化 3D 高斯：
 - 将初始 3D 高斯点分布调整为较好的结构。
- 预训练阶段：
 - 初始 3000 次迭代只优化 3D 高斯，而不引入时间变形： $\hat{I} = S(M, G)$ ，
其中 G 是静态 3D 高斯集合。

效果：

- 提高收敛稳定性，避免数值误差。
-

(2) Loss Function

损失项：

1. 颜色重建损失：

- 监督渲染图像与目标图像之间的像素误差： $L_{\text{color}} = \|\hat{I} - I\|_1$ ，
其中 \hat{I} 是渲染图像， I 是目标图像。

2. 总变分损失 (Total Variation Loss)：

- 限制特征变化，保持平滑：

$$L_{\text{tv}} = \sum \|\nabla f\|_1.$$

总损失：

$$L = L_{\text{color}} + \lambda_{\text{tv}} L_{\text{tv}}, \quad (9)$$

其中 λ_{tv} 是权重参数。

总结

4D Gaussian Splatting 框架通过以下方法实现动态场景的高效建模：

1. 结合多分辨率时空编码提取高斯点特征；
2. 使用高斯变形场网络预测动态变形；
3. 通过可微分渲染进行优化，实现实时渲染效果。

Table 1. Quantitative results on the synthesis dataset. The **best** and the **second best** results are denoted by pink and yellow. The rendering resolution is set to 800×800 . “Time” in the table stands for training times.

Model	PSNR(dB)↑	SSIM↑	LPIPS↓	Time↓	FPS ↑	Storage (MB)↓
TiNeuVox-B [6]	32.67	0.97	0.04	28 mins	1.5	48
KPlanes [8]	31.61	0.97	-	52 mins	0.97	418
HexPlane-Slim [4]	31.04	0.97	0.04	11m 30s	2.5	38
3D-GS [14]	23.19	0.93	0.08	10 mins	170	10
FFDNeRF [12]	32.68	0.97	0.04	-	< 1	440
MSTH [37]	31.34	0.98	0.02	6 mins	-	-
Ours	34.05	0.98	0.02	20 mins	82	18

Table 2. Quantitative results on HyperNeRF’s [25] vrig dataset. Rendering resolution is set to 960×540 .

Model	PSNR(dB)↑	MS-SSIM↑	Times↓	FPS↑	Storage(MB)↓
Nerfies [24]	22.2	0.803	~ hours	< 1	-
HyperNeRF [25]	22.4	0.814	32 hours	< 1	-
TiNeuVox-B [6]	24.3	0.836	30 mins	1	48
3D-GS [14]	19.7	0.680	40 mins	55	52
FFDNeRF [12]	24.2	0.842	-	0.05	440
Ours	25.2	0.845	1 hour	34	61

Table 3. Quantitative results on the Neu3D’s [17] dataset, rendering resolution is set to 1352×1014 .

Model	PSNR(dB)↑	D-SSIM↓	LPIPS↓	Time ↓	FPS↑	Storage (MB)↓
NeRFPlayer [35]	30.69	0.034	0.111	6 hours	0.045	-
HyperReel [2]	31.10	0.036	0.096	9 hours	2.0	360
HexPlane-all* [4]	31.70	0.014	0.075	12 hours	0.2	250
KPlanes [8]	31.63	-	-	1.8 hours	0.3	309
Im4D [18]	32.58	-	0.208	28 mins	~5	93
MSTH [37]	32.37	0.015	0.056	20 mins	2(15 [‡])	135
Ours	31.15	0.016	0.049	40 mins	30	90

*: The metrics of the model are tested without “coffee martini” and resolution is set to 1024×768 .

‡: The FPS is tested with fixed-view rendering.

Table 4. Ablation studies on synthetic datasets using our proposed methods.

Model	PSNR(dB)↑	SSIM↑	LPIPS↓	Time↓	FPS↑	Storage (MB)↓
Ours w/o HexPlane $R_l(i, j)$	27.05	0.95	0.05	10 mins	140	12
Ours w/o initialization	31.91	0.97	0.03	19 mins	79	18
Ours w/o ϕ_x	26.67	0.95	0.07	20 mins	82	17
Ours w/o ϕ_r	33.08	0.98	0.03	20 mins	83	17
Ours w/o ϕ_s	33.02	0.98	0.03	20 mins	82	17
Ours	34.05	0.98	0.02	20 mins	82	18

Experiment

本部分介绍实验的设置、结果对比、消融实验和讨论，展示 **4D Gaussian Splatting (4D-GS)** 的高效性和优越性能。

1. Experimental Settings

实验使用 **PyTorch** 框架，测试平台为单张 RTX 3090 GPU。超参数基于 **3D-GS** 配置进行了细化调整。

Synthetic Dataset

- **D-NeRF 数据集：**
 - 包含单目设置下的动态场景，每个时间点仅有一张图像；
 - 动态帧数在 50-200 之间；
 - 数据集无背景，场景结构相对简单。
- **训练设置：**
 - **训练迭代数：** 20000 次；
 - **Pruning Interval：** 8000；
 - 多分辨率 HexPlane 模块上采样率设置为 2；
 - 3D 高斯点的生长截止于第 15000 次迭代。

Real-World Dataset

- 使用 **HyperNeRF** 和 **Neu3D** 提供的真实世界数据集作为基准：
 - **HyperNeRF 数据集：**
 - 单目采集，包含 1-2 台相机，摄像机运动简单。
 - 上采样率设置为 [2, 4]，解码器隐藏维度为 256。
 - **Neu3D 数据集：**
 - 包含 15-20 台静态相机，拍摄时间较长，运动复杂；
 - 使用 SfM 计算的初始点云作为高斯点初始化。

2. Results

通过以下指标评估实验结果：

- **PSNR (峰值信噪比)**: 衡量图像重建质量；
- **SSIM (结构相似性指数)**: 评价重建图像的结构一致性；
- **LPIPS (感知质量指标)**: 衡量视觉质量的差异性；
- **FPS (帧率)**: 评估实时性；
- **Storage (存储成本)**: 比较模型的存储效率。

Synthetic Dataset Results

表中展示了在 D-NeRF 数据集上的结果：

Model	PSNR (dB) ↑	SSIM ↑	LPIPS ↓	Time ↓	FPS ↑	Storage (MB) ↓
TiNeuVox-B	32.67	0.97	0.04	28 mins	1.5	48
KPlanes	31.61	0.97	-	52 mins	0.97	418
3D-GS	23.19	0.93	0.08	10 mins	170	10
Ours	34.05	0.98	0.02	20 mins	82	18

Real-World Dataset Results

在 HyperNeRF 的 `vrig` 数据集中的结果如下：

Model	PSNR (dB) \uparrow	MS-SSIM \uparrow	Time \downarrow	FPS \uparrow	Storage (MB) \downarrow
NeRFies	22.2	0.803	~hours	<1	-
HyperNeRF	22.4	0.814	32 hours	<1	-
TiNeuVox-B	24.3	0.836	30 mins	1	48
3D-GS	19.7	0.680	40 mins	55	52
Ours	25.2	0.845	1 hour	34	61

3. Ablation Study

通过消融实验验证关键模块对模型性能的影响：

Spatial-Temporal Structure Encoder

去除 **HexPlane** 模块（仅保留浅层 MLP 进行特征建模）后的对比结果如下：

Model	PSNR (dB) \uparrow	SSIM \uparrow	LPIPS \downarrow	Time \downarrow	FPS \uparrow	Storage (MB) \downarrow
Ours w/o HexPlane	27.05	0.95	0.05	10 mins	140	12
Ours	34.05	0.98	0.02	20 mins	82	18

Gaussian Deformation Decoder

分别去除位置、旋转和缩放头模块后的实验结果：

Model	PSNR (dB) \uparrow	SSIM \uparrow	LPIPS \downarrow	Time \downarrow	FPS \uparrow	Storage (MB) \downarrow
Ours w/o ϕ_x	26.67	0.95	0.07	20 mins	82	17
Ours w/o ϕ_r	33.08	0.98	0.03	20 mins	83	17
Ours w/o ϕ_s	33.02	0.98	0.03	20 mins	82	17
Ours	34.05	0.98	0.02	20 mins	82	18

4. Discussions

Rendering Speed

4D-GS 的渲染速度与屏幕中 3D 高斯点的数量呈负相关。在分辨率为 800×800 的设置下，点数低于 30000 时，渲染速度可达 90 FPS。

Tracking with 3D Gaussians

实验表明，4D-GS 不仅支持动态场景的实时渲染，还能用于 3D 目标的跟踪任务，每个时间点的存储成本仅为 18MB。

5. Limitations

尽管 4D-GS 实现了快速收敛和高效渲染，仍存在以下局限：

1. 大规模运动的建模：

- 在复杂运动场景中，网络可能无法有效捕获大幅运动。

2. 单目场景的稀疏输入问题：

- 输入数据在时间和空间维度的稀疏性可能导致模型过拟合，影响新视点的渲染质量。

3. 城市规模的重建：

- 由于高斯变形场网络的查询开销较大，处理大规模场景仍面临挑战。

Conclusion

本论文提出了一种名为 **4D Gaussian Splatting (4D-GS)** 的新方法，用于高效地表示和渲染动态场景。以下是本研究的主要贡献与结论：

1. 核心贡献

1. 高效的动态场景建模

- 本文引入了 **高斯变形场网络 (Gaussian Deformation Field Network)**，能够同时建模高斯点的运动和形状变化。
- 使用空间-时间结构编码器将相邻高斯点进行连接，预测更精确的运动和形状变形。

2. 实时渲染性能

- 通过优化 4D 表示法，4D-GS 达到了在高分辨率 (800×800) 下以 82 FPS 的速度进行实时渲染的能力，同时保持与 SOTA 方法相当或更优的图像质量。

3. 扩展性与多功能性

- 除了动态场景的建模，4D-GS 还展示了在 4D 目标跟踪和编辑任务中的潜力。
-

2. 主要实验结论

- **渲染速度：**

- 在动态场景下，4D-GS 能够显著提升渲染速度，同时极大地降低存储和内存开销。

- **质量与效率的平衡：**

- 相较于现有方法，4D-GS 实现了更高的渲染质量、更快的训练速度和更低的存储需求。
-

3. 局限性与未来方向

尽管 4D-GS 在多个方面表现出色，但仍存在以下局限：

1. 大规模运动建模：

- 当场景中存在大幅度运动时，模型的收敛和渲染效果可能受到限制。

2. 稀疏输入问题：

- 在单目输入和稀疏时间采样下，模型可能过拟合训练集数据，导致新视图渲染效果下降。

3. 城市规模场景的适配：

- 当前方法对大规模城市场景的重建效率较低，需要进一步优化查询开销。

未来工作

为解决上述问题，未来的研究方向包括：

1. 设计更紧凑的变形网络，以适配大规模场景；
 2. 引入更多先验信息（如深度或光流），提高稀疏输入场景的表现；
 3. 扩展方法的适用性，使其能够更好地处理复杂的动态场景和大规模运动。
-

通过本研究，4D-GS 为动态场景渲染和表示提供了一种高效且扩展性强的解决方案，在 3D 视觉领域的多项应用中展现了潜力。