arXiv:2611.32202v1 [cs.CV] 24 Nov 2026

SplatTok：利用场景级令牌进行数据驱动的 3D高斯泼溅压缩

Sai Karthikey Pentapati[1],Shashank Nelamangala Sridhara[2],Birendra Kathariya[2],
Peng Yin[2],Guandao Yang[1],Alan C. Bovik[3]

[1]德克萨斯大学奥斯汀分校 [2]杜比实验室 [3]科罗拉多大学博尔德分校

联系邮箱： karthikey97@gmail.com | 项目主页： https://karthikey97.github.io/splattok/

摘要

3D高斯泼溅支持实时渲染，但需要大量存储空间。现有压缩方法要么对给定泼溅的基元进行编码，要么针对每个场景独立优化紧凑表示。然而，高斯泼溅只是场景的一种呈现：不同的基元排列也能生成几乎相同的渲染视图。我们利用解码端的这一自由度，提出 SplatTok，这是一种学习式场景重合成编解码器。Transformer 在数千个真实场景上进行训练，将已有泼溅编码为紧凑且具有空间锚定的潜在令牌；共享解码器则合成一组新的标准高斯基元，而无需保留与输入的对应关系。该表示通过渲染监督而非基元重建进行学习。在压缩时，量化感知潜变量搜索仅优化场景编码，保持网络冻结，并弥补编码器剩余误差的大部分。在标准基准测试中，仅使用输入泼溅的渲染图像进行潜变量搜索，而不使用原始训练图像时，SplatTok 达到与前馈编解码器 FCGS 相同的峰值信噪比，文件大小却缩小 4 至 6.5 倍，输入泼溅压缩率约为 75 至 110 倍。使用训练图像时，其峰值信噪比超过逐场景方法 HAC，文件大小缩小 13% 至 41%。在 DL3DV 上训练的单个模型能够泛化到包含数百万个高斯的未见场景，并支持多种目标码率，无需重新训练。

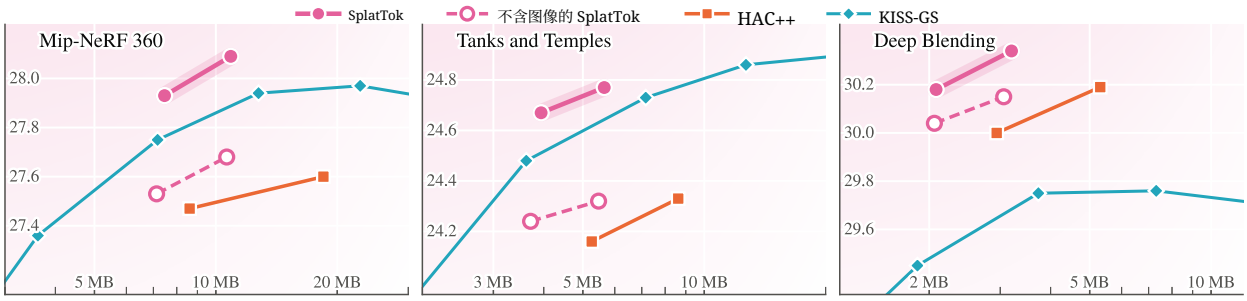

图 1. 压缩现有的高斯泼溅。 峰值信噪比（分贝）与每个场景的大小（MB，对数刻度）的关系。以训练图像为潜变量搜索的目标时，SplatTok 在所有三个数据集上的峰值信噪比均优于 HAC++ [10] 和 KISS-GS [31]。即使仅以输入 splat 的渲染图像为目标，它的表现仍优于 HAC++；HAC 则是以训练图像进行训练的。SplatTok 的所有数据点均压缩了表 2中的 3DGS-MCMC splat。

1 引言

三维高斯泼溅 [21] 已迅速成为捕获和分享逼真三维场景的标准方式。一个场景
它从多个相机位姿拍摄的多视图图像中重建一次，随后便可在普通硬件上实时渲染。渲染速度很快，因为场景直接以数百万个高斯基元存储，每个基元都带有数十种属性；但这也使文件体积庞大：一个典型场景占用数百兆字节，甚至超过一吉字节。这对于流式传输到头戴式设备或手机、在资产库中存储成千上万个场景，或像发送图像或视频那样轻松发送来说，都太大了。在这项工作中，我们压缩现有的泼溅表示。我们希望文件体积缩小一到两个数量级

同时渲染效果几乎与原始文本相当，生成速度快，无需提供构建该泼溅表示时所用的图像，并且解码后能还原为任何 3DGS 渲染器都可显示的标准 splat。最后两个条件在实践中很重要：泼溅表示越来越多地以独立文件的形式分发，此时训练图像早已被丢弃；而它们是在为标准格式构建的渲染器中查看的。

现有压缩方法各自实现了其中一些目标，但无法兼顾所有目标。最紧凑的结果来自逐场景方法：这类方法为每个场景构建新的结构化表示，并针对该场景的训练图像对其进行优化，通常还会采用学习式上下文模型进行熵编码 [8, 10, 28, 42]；综述请参见 [2]。其中表现最强的方法可实现近 100 倍的压缩比，但需要训练图像，每个场景需要耗费数十分钟，而且通常必须针对每种目标尺寸重新训练。它们的输出也不是标准 splat：基于锚点的方法会存储学习式特征，并在渲染时通过小型网络为每个视图生成高斯，因此需要使用专用渲染器。训练后方法会对 splat 进行剪枝和量化，而不构建新的表示 [15, 24, 33, 38, 43]，并保留标准格式，但压缩比相对有限，或需要数分钟的微调才能恢复质量。前馈编解码器 FCGS [9] 彻底省去了逐场景优化：在大量场景上训练的网络可在数秒内压缩 splat，无需图像，并解码为标准 splat。它逐个基元进行处理，利用有限的邻居上下文对每个高斯的属性进行编码，同时保留原始高斯集合，压缩比约为 10 至 30 倍。现有方法都无法在无需训练图像、无需逐场景表征训练的情况下实现逐场景方法的压缩比，同时又能解码为标准 splat。

我们采取了不同的视角：我们不为每个场景学习新的表示，也不为单个高斯学习编解码器，而是学习整个场景的模型，并在其中进行压缩。我们的方法 SplatTok 将整个泼溅编码为一组紧凑的潜在令牌，这些令牌锚定在场景各处的三维位置上，再将它们解码为一组放置在这些锚点周围的新高斯。编码器会关注整个场景，因此每个令牌都能结合其他所有内容来概括其所在区域；解码器从未接触输入高斯，可以自由合成任意排列，以实现最佳场景渲染效果。我们仅以渲染结果作为监督，在大型真实场景数据集上训练模型，并对令牌进行量化和熵编码，以形成比特流。这与 FCGS 在学习内容和存储内容上有所不同：FCGS 学习对给定基元的属性进行编码，而 SplatTok 学习场景的结构，并存储一份用于重新生成基元的描述。

由于解码器能够将其潜空间中的任意编码映射为可渲染的泼溅，压缩新场景便归结为寻找其编码。编码器的输出已经是优质的潜在编码；在网络冻结的情况下，仅对令牌进行短暂搜索，并在搜索过程中应用量化，即可使其接近解码器所能表达的最佳编码。搜索只需将输入泼溅从其相机视角渲染出的图像作为目标，因此无需训练图像也能进行；若有训练图像，也可将其用作目标。这与学习式图像压缩中的编码时细化相似，[7, 46], 此处将其应用于整个三维场景。我们的分析揭示了其有效原因：搜索仅改变了 4% 至 10% 的编码，却使输入泼溅的保真度提升约 8 分贝，这表明在固定解码器和允许的编码空间内，细化弥补了至少 84% 的初始最优性差距。码率由令牌的精度和锚点的密度决定，因此一个训练模型即可适用于多种目标尺寸，而且在我们的实验中，所有数据集都使用相同的权重。

最终得到的编解码器在更小的文件体积下，PSNR 优于已发表的最佳逐场景方法。在 Mip-NeRF 360 [5], 上，SplatTok 每个场景以 7.46MB 的体积达到 27.93 分贝，比 HAC++ [10] 在 8.61MB 时高 0.46 分贝；以 10.89 MB 的体积达到 28.09 分贝，比 HAC++ 在 18.48 MB 时高 0.49 分贝。在 Tanks and Temples [23] 和 Deep Blending [18], 上也取得了类似提升，而且所有体积都使用同一个模型，而非每种体积各自训练一次。采用更稀疏的锚点时，同一模型的 PSNR 仍高于 HAC++，且文件体积缩小了 53% 至 60%。在不使用训练图像的情况下，对于相同的输入泼溅，SplatTok 在 DL3DV、Mip-NeRF 360、Tanks and Temples 和 Deep Blending 上取得与 FCGS 相当的 PSNR，而文件体积小 4 至 6.5 倍。同一模型仅使用 DL3DV 泼溅进行训练，便能压缩来自其他数据集、包含数百万个高斯的场景，无需重新训练。

总而言之，我们的贡献如下：

- 一种用于高斯泼溅的场景重合成编解码器，可将整个场景编码为一组紧凑的三维锚定令牌，并根据这些令牌合成一组新的标准3DGS 基元；该编解码器仅通过渲染监督进行训练，训练数据涵盖数千个真实场景。

- 将压缩视为在解码器潜空间中的短程搜索，无需训练图像、无需针对每个场景进行表征训练，也无需为不同目标尺寸分别训练模型，即可达到逐场景压缩质量。我们证明，这种方法之所以有效，是因为编码器剩余误差大多来自摊销差距，而非解码器的局限。

解码器。

- 在 DL3DV、Mip-NeRF 360、Tanks and Temples 和 Deep Blending 上的结果：PSNR 高于 HAC++；在默认锚点密度下，文件大小减少 13% 至 41%；使用更稀疏的锚点时，文件大小减少 53% 至 60%；FCGS 在不使用训练图像的情况下，文件大小缩小 4 至 6.5 倍，且 PSNR 更高；每个场景的高斯体数量可从 350,000 泛化到数百万。

本报告其余部分安排如下。第 2节回顾相关工作，第 3 节明确符号表示，第 4 节介绍 SplatTok。第 5 节将其与前馈方法、训练后方法和逐场景方法进行评估比较，研究其码率控制、潜变量搜索及设计选择，并报告运行时间。第6 节讨论局限性和未来方向。

2 相关工作

逐场景训练的紧凑高斯泼溅。第一类方法在拟合训练图像时缩小泼溅本身。这些方法学习掩码以剪枝冗余高斯体 [24, 34], 将属性替换为码本索引或量化潜变量 [17, 24, 32], 将属性排列成平滑的二维网格，以便使用标准图像编解码器高效压缩 [30], 或在训练目标中加入显式码率项 [40]。SpeedyGS [52]通过内容感知两阶段优化压缩普通的 3DGS 基元。这些方法使输出接近标准 splat，在 Mip-NeRF 360 上实现约 15 至 45 倍压缩，但每个场景仍需使用训练图像进行完整优化，而且改变目标大小就意味着必须重新优化。

采用学习式熵模型的基于锚点的方法。最小的文件来自将非结构化泼溅替换为结构化泼溅的方法。Scaffold-GS [28] 存储带有学习式特征的稀疏锚点，再由小型网络为每个视图生成附近的高斯体。HAC [8] 和 HAC++ [10] 使用哈希网格上下文对这些锚点特征的分布建模；而在 HAC++中，还会对每个锚点内部的上下文建模。ContextGS [42]以由粗到细的方式解码锚点，并让每个层级受前序层级的条件约束；CompGS [26] 则将基元预测为相对于锚点的残差。近期方法进一步改进了上下文模型，采用上下文自适应三平面、 [50],分层稀疏变换编码、 [44], 局部几何感知分层上下文 [12], 或支持仅整数推理的逐锚点因果上下文 [48]。这些方法合计实现了接近 100 倍的压缩率，创下已报告的最佳水平。代价是：每个场景都要针对训练图像进行数十分钟的逐场景优化；除渐进式编码外，每种目标码率都要重复优化 [11], ；此外，输出并非标准 splat，渲染时还需要使用该方法专属的网络。相比之下，SplatTok 在数千个场景上一次性学习场景结构，因此新场景无需单独训练表示，其潜变量可解码为普通高斯体。

后训练压缩。 另一类方法压缩已经训练完成的 splat。LightGaussian [15]根据高斯对训练视图的贡献进行剪枝，将球谐函数蒸馏至更低阶，并对其余部分进行向量量化；Compressed3D [33]根据属性对渲染误差的敏感性对其进行聚类；MesonGS [43] 将基于重要性的剪枝与位置的八叉树编码以及属性的变换编码器相结合。随后，每种方法都会针对训练图像对压缩后的 splat 进行数分钟的微调。FlexGaussian [38] 取消了微调，并在无需训练的情况下选择剪枝和混合精度量化。KISS-GS [31] 将强力压实与基于图像的属性编解码器及编码感知微调相结合，在保留标准 splat 的同时达到了该类方法中最高的压缩比。这些方法保留了标准格式，但压缩比相对有限，尤其是在不进行微调时。相比之下，SplatTok 的潜变量搜索直接优化潜在编码，并将量化纳入其中；冻结解码器在数千个场景上训练而成，可使编码保持在合理场景的空间内。与对基元进行剪枝和量化相比，该方法实现了高出数倍的压缩比；由于 SplatTok 接受任意标准 splat，因此它与这些方法互为补充，还能进一步压缩已经被它们压实过的 splat（第 5.5节）。

前馈压缩。FCGS [9]据我们所知，它是首个能够通过在多个场景上训练的网络进行一次前向传播来压缩泼溅的编解码器。它不需要图像，也不需要逐场景优化；解码后得到标准 splat，且运行只需数秒。它对给定的高斯元本身进行编码：每个基元的属性都利用已解码邻居提供的上下文进行熵编码，解码后的泼溅与输入包含相同的一组高斯元，因此压缩率限制在约 10 至 30 倍。LocoMoco [27] 通过对按Morton序排列的高斯元窗口进行长上下文熵建模扩展了 FCGS，并且仍对给定的基元进行编码；由于没有公开代码或模型，我们未将其纳入比较。前馈重建方法也与其预测高斯元的压缩相结合过 [19, 49]；这类方法压缩的是从图像重建得到的输出，而非现有的

泼溅。SplatTok 同样是在多个场景上训练的，但它学习的是整个场景的模型，而非单个高斯元的模型，并存储一种紧凑描述，借此生成一组新的高斯元。

学习式压缩与编码时细化。 学习式图像编解码器将分析与合成变换同熵模型结合 [3, 4, 29]。由于摊销编码器无法为每个输入找到最佳编码，因此在编码时、解码器固定的情况下细化每个潜变量，可以弥补其中一部分差距 [7, 46]。隐式神经表示则采取相反的极端做法，使网络对每个实例过拟合 [14], , 逐场景泼溅压缩方法也是如此。SplatTok 介于两者之间：它是一个在数千个场景上进行摊销的编解码器，并在编码时冻结网络、逐场景进行细化；我们表明，这种细化弥补了大部分与解码器能够表达的最佳编码之间的差距。

集合潜变量与高斯泼溅的潜变量模型。 我们的架构建立在用于三维形状的集合潜变量之上，通过交叉注意力将点云编码为固定大小的潜向量集合 [51] ，如今已成为许多三维生成模型的基础 [53]。高斯泼溅的潜变量模型 [36, 45] 旨在支持生成先验，主要面向物体级应用，而不是忠实地重建给定场景，也不会根据大小和渲染质量进行评估。在场景级应用中，Can3Tok [16] 是一种用于 3D 高斯的变分自编码器，旨在为基于扩散的场景生成提供潜空间。一组固定的学习式查询对输入高斯执行交叉注意力，因此无论场景的范围或复杂程度如何，每个场景都会被映射到相同大小的潜变量，而解码器始终输出相同数量的高斯。为适应该预算，每个输入泼溅首先会经过筛选，并下采样至固定数量的基元（其实验中为 40K），这会丢弃真实采集中的大部分内容。该模型使用输入高斯属性上的 L2损失进行训练，而不是基于渲染结果进行训练。由于许多不同的高斯集合都能渲染出相同的场景，这种损失要求解码器复现某一种特定的基元排列，而非场景的外观；同时，重建结果通过属性误差而非渲染保真度进行衡量。其检查点未公开，因此无法进行直接比较。GaussianGPT [39]通过稀疏三维卷积自编码器的离散令牌，以自回归方式生成场景，并采用无查找量化。其编码器接收一个体素网格，每个体素中最多包含一个高斯，该高斯以相对于体素中心的偏移量存储；当一个体素包含多个高斯时，除其中一个外的其他高斯都会被随机丢弃。因此，其训练场景会按照这一结构构建：使用每个锚点对应一个高斯、且锚点与体素中心对齐的 Scaffold-GS 进行拟合；对于场景，不会对视角相关颜色建模。使用标准3DGS训练的泼溅在每个体素中包含许多大小不同的重叠基元，因此超出了该输入域。相比之下，SplatTok 会随着输入泼溅增大锚点数量，从而同步增大编码和输出；它能够接受任意标准泼溅，并且仅通过渲染结果进行监督。

来自图像的场景标记。TokenGS [35] 和 SceneTok [1] 同样使用一组学习到的令牌来描述场景，但它们旨在基于带姿态图像进行前馈重建，而不是压缩泼溅。在 TokenGS 中，可学习的 3DGS 令牌与少量输入视图的特征进行交叉注意力，并分别回归固定数量的高斯，通过渲染损失进行训练。SceneTok 将许多上下文视图编码为一小组无结构令牌，随后由整流流解码器从中生成新视图图像，而不是生成显式的三维表示。二者都不以现有泼溅作为输入，也不生成以大小和渲染保真度为评估指标的比特流；它们的架构、损失和处理流程旨在从图像快速构建可渲染场景。相比之下，SplatTok 从已完成的泼溅开始，在整个场景的三维空间中设置其令牌锚点，并作为编解码器进行训练和评估。

3 背景与记号

高斯泼溅 [21]将场景表示为由 N 个各向异性三维高斯组成的集合 $\mathcal{G} = \{g_i\}_{i=1}^{N}$。每个高斯 $g_i = (\boldsymbol{\mu}_i, s_i, q_i, o_i, f_i)$包含一个位置 $\boldsymbol{\mu}_i \in \mathbb{R}^3$、一个对数尺度 $s_i \in \mathbb{R}^3$、由四元数 $q_i \in \mathbb{R}^4$表示的旋转、一个不透明度逻辑值 $o_i \in \mathbb{R}$，以及最高至 ℓ阶的球谐（SH）颜色系数 $f_i \in \mathbb{R}^{(\ell+1)^2\times 3}$。其协方差为 $\boldsymbol{\Sigma}_i = R(q_i)\,\mathrm{diag}(e^{s_i})^2 R(q_i)^\top$，不透明度为 $\mathrm{sigmoid}(o_i)$。我们用 $\mathcal{R}(\mathcal{G},P)$ 表示从相机 P 渲染得到的泼溅图像，并将渲染器视为固定且可微的算子。泼溅通过从相机 $\{P_v\}_{v\in\mathcal{V}}$拍摄的多视图图像 $\{I_v\}_{v\in\mathcal{V}}$（即训练图像）进行拟合，其中 $\mathcal{V} = \{1,\ldots,V\}$为视图索引。

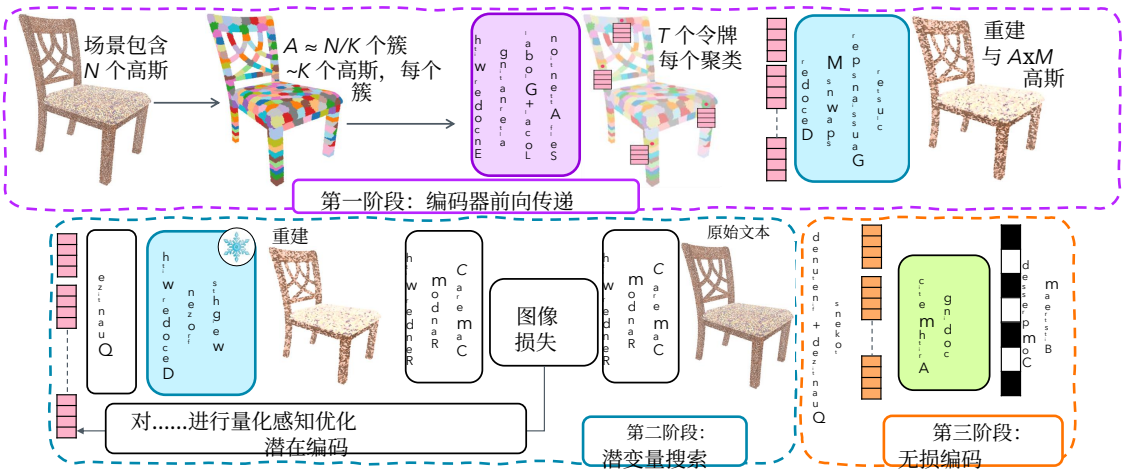

图 2. SplatTok 概览。 输入 splat 被划分为锚点。编码器将每个锚点的高斯汇总为固定数量的潜变量令牌。解码器根据每个锚点的令牌，在其周围重新生成固定数量的新高斯。在编码时，在两个网络均被冻结的情况下，潜变量搜索会对量化令牌进行优化。

4 SplatTok

4.1 概述

ˆ SplatTok 将泼溅 $\mathcal{G}$ 压缩为一组锚定在三维位置上的潜在令牌，并据此重建新的泼溅 $\mathcal{G}$（图 2）。我们首先将场景划分为 A 个锚点，每个锚点都具有一个位置和一个尺寸（第 4.2 节）。编码器 E 将每个锚点周围的高斯转换为 T 个令牌，每个令牌包含 C 个通道；编码时会关注整个场景，由此得到编码器的编码 $Z^0 = E(\mathcal{G}) \in \mathbb{R}^{A\times T\times C}$（第 4.3 节）。解码器 D 将编码 $Z \in \mathbb{R}^{A\times T\times C}$ 中每个锚点对应的令牌映射为围绕该锚点放置的 M 个新高斯，因此 $\hat{\mathcal{G}} = D(Z)$ 是由 AM 个高斯构成的标准 splat，任何 3DGS 渲染器都能将其显示出来（第 4.4 节）。此处用帽号表示重建的量。两个网络仅通过渲染进行联合训练，训练数据涵盖许多场景（第 4.5 节）。存储场景时，令牌会被量化为 b 位并进行熵编码（第 4.6 节）。最后，在编码时冻结两个网络，通过潜变量搜索优化量化后的编码；搜索目标是输入泼溅的渲染图像，或者在可用时使用训练图像（第 4.7 节）。

输入与归一化。SplatTok 接收存储形式完全不变的泼溅：即第3节所述参数化下的一组无序高斯，数量为任意N，且不需要训练图像。编码前，根据相机计算相似变换，将泼溅及其相机变换到归一化坐标系中，与 *gsplat* 库的归一化方式一致 [47], ，使不同场景的泼溅具有统一的位置和尺度。解码后的泼溅位于同一坐标系中，并使用变换后的相机进行渲染。

4.2 将场景划分为锚点

我们将场景划分为A 个锚点，其数量与N 成正比，以保证每个锚点对应固定数量的高斯，因此锚点数量会随场景规模变化。锚点位置 $\{\mathbf{a}_k\}_{k=1}^{A}$ 是对所有高斯位置 $\{\boldsymbol{\mu}_i\}$进行k-均值聚类后得到的中心。成员数超过K 的簇会被拆分。每个高斯都会归入最近锚点；只要该锚点的成员数少于K，它也会归入次近锚点。由此得到的成员集合$\mathcal{N}_k$，其中 $|\mathcal{N}_k| \le K$，会在聚类边界附近重叠，使相邻锚点都能覆盖它们之间的高斯。每个锚点还会存储一个大小 σ_k，即其成员到$\mathbf{a}_k$的距离中位数，并将其限制在场景锚点大小中位数的某个倍数以内，避免稀疏离群点上的孤立锚点覆盖过大区域。锚点内的位置以归一化偏移量 $(\boldsymbol{\mu}_i - \mathbf{a}_k)/\sigma_k$表示，因此无论锚点覆盖的是稠密表面还是稀疏背景区域，每个锚点都以相同尺度进行处理。锚点位置和大小是编码中仅有的几何辅助信息。

4.3 编码器

尽管锚点包含的高斯数量可能不同，编码器仍会将每个锚点概括为相同数量T的令牌。首先，根据每个高斯的非位置属性$\mathbf{e}_i = \phi(\mathbf{s}_i, \bar{\mathbf{q}}_i, o_i, \mathbf{f}_i)$生成嵌入，其中 ϕ是一个多层感知机，$\bar{\mathbf{q}}_i$ 是单位四元数。在锚点k内，将该嵌入与成员的归一化偏移量结合，形成一个键值令牌

$$\mathbf{h}_{k,i} = \psi\big(\mathbf{e}_i,\ \gamma\big((\boldsymbol{\mu}_i - \mathbf{a}_k)/\sigma_k\big)\big), \qquad i \in \mathcal{N}_k, \tag{1}$$

其中 γ是傅里叶嵌入，ψ 是多层感知机。一组由所有锚点共享、并通过锚点大小的嵌入进行偏移的 T 学习式查询 $\{\mathbf{w}^{\mathrm{e}}_t\}_{t=1}^{T}$，$\mathbf{w}^{\mathrm{e}}_t + \mathbf{W}^{\mathrm{e}} \log\,\sigma_k$，对 $\{\mathbf{h}_{k,i}\}_{i\in\mathcal{N}_k}$执行交叉注意力，随后经过一个前馈层。由于注意力会对锚点所包含的成员数量进行归一化，因此输出始终是每个锚点对应的 T 个令牌。

这是 SplatTok 的核心结构选择。锚点的编码长度是固定的，不受其中包含的高斯数量、排列方式及其各自属性的影响。因此，比特流会随锚点数量和令牌精度增长，而且解码器可以用一组不同于输入的高斯来表示锚点。与之相比，FCGS [9], 会对给定的每个高斯分别编码，因此其编码长度取决于输入基元。它还将锚点密度作为码率控制：无需重新训练，只需减少每个场景中的锚点数量，就能缩短编码（第 5.5 节）。

随后，Transformer 会联合处理所有锚点的令牌，使每个令牌都能结合场景其余部分的上下文来概括其所在区域。在 Transformer 之前，会将令牌索引t 的可学习嵌入添加到每个令牌中。前几层在每个锚点的T 个令牌之间应用自注意力，其余层则交替使用这种局部注意力和针对场景全部AT 个令牌的全局自注意力，类似于 VGGT [41]。在全局层中，查询和键携带其锚点位置$\mathbf{a}_k$的三维旋转位置编码 [37]，该编码由锚点的T 个令牌共享，因此令牌之间的注意力取决于其锚点的相对位置。全局注意力使每个令牌都能获取自身锚点以外的上下文，这是逐锚点或逐基元编码器无法利用的；将编码器限制为仅在锚点内部进行注意力计算，会降低重建场景的质量（第5.7节）。最后，层归一化和线性投影将每个令牌降至C 个通道，得到编码器编码$\mathbf{Z}^0$的令牌$\mathbf{z}^0_{k,t} \in \mathbb{R}^C$。

4.4 解码器

解码器仅根据每个锚点自身的令牌生成该锚点的高斯。一组M 学习式查询$\{\mathbf{w}^{\mathrm{d}}_m\}_{m=1}^{M}$由所有锚点共享，并通过尺寸嵌入$\mathbf{W}^{\mathrm{d}} \log \sigma_k$进行偏移；这些查询与某个编码$\mathbf{Z}$的锚点令牌$\{\mathbf{z}_{k,t}\}_{t=1}^{T}$进行交叉注意力计算，残差多层感知机则将每个经过注意力处理的查询映射为一个高斯的参数（$\boldsymbol{\delta}_{k,m}, \hat{\mathbf{s}}_{k,m}, \hat{\mathbf{q}}_{k,m}, \hat{o}_{k,m}, \hat{\mathbf{f}}_{k,m}$）。该高斯相对于其锚点放置，

$$\hat{\boldsymbol{\mu}}_{k,m} = \mathbf{a}_k + \sigma_k\, \boldsymbol{\delta}_{k,m}, \tag{2}$$

其对数尺度、旋转和不透明度逻辑值直接用于第3节所述的参数化。颜色采用亮度–色度（BT.709 YCbCr）球谐基进行编码和预测，其中只有亮度包含视角相关项，而两个色度通道仅保留直流项。这样可以去除大部分颜色系数，而输入泼溅的损失不到 0.1 分贝；预测出的颜色随后会转换回 RGB球谐。解码后的泼溅$\hat{\mathscr{G}} = D(\mathbf{Z}) = \{\hat{g}_{k,m}\}$ 包含AM个符合标准格式的高斯。解码器从未接触输入高斯，因此不受复现它们的限制：只要能更好地渲染场景，它就可以采用与输入不同的放置方式、尺寸和权重来设置基元。每个锚点都独立解码，从而使解码过程能够并行进行，且内存占用与锚点数量成正比。

4.5 训练

编码器和解码器在许多场景上进行联合训练，监督信号仅来自渲染。对于一个训练场景，我们对其泼溅进行编码和解码，并从每一步重新抽取的相机随机子集中渲染结果 $\mathscr{V}' \subset \mathscr{V}$；每个渲染结果都与目标图像 $\mathbf{Y}_v$进行比较；对于渲染图像$\mathbf{I}$ 和目标图像$\mathbf{Y}$，损失为

$$\mathscr{L}(\hat{\mathbf{I}}, \mathbf{Y}) = (1-\lambda)\,\mathrm{MSE}(\hat{\mathbf{I}}, \mathbf{Y}) + \lambda\,\big(1 - \mathrm{SSIM}(\hat{\mathbf{I}}, \mathbf{Y})\big), \tag{3}$$

其中，MSE 表示像素上的均方误差。训练目标为

$$\mathbb{E}_{\mathscr{G},\mathscr{V}'}\Big[\frac{1}{|\mathscr{V}'|}\sum_{v\in\mathscr{V}'} \mathscr{L}\big(\mathscr{R}(D(E(\mathscr{G})), \mathbf{P}_v),\ \mathbf{Y}_v\big)\Big], \tag{4}$$

对训练场景和视图子集取期望。训练期间，目标是训练图像，$\mathbf{Y}_v = \mathbf{I}_v$。单个高斯的属性不参与损失计算，输入与输出基元之间也无需建立对应关系，因为解码器会生成一组不同的基元。网络使用连续令牌进行训练；只有在存储场景时才进行量化（第4.6节）。

4.6 量化与熵编码

为存储场景，我们使用b 位对令牌逐通道进行量化，采用对称均匀量化器。With$u_{\max} = 2^{b-1} - 1$ 以及逐通道步长 $\Delta_c = \max_{k,t} |\overset{0}{z}_{k,t,c}| / u_{\max}$，该步长根据编码器的编码$\mathbf{Z}^0$, 设定；编码$\mathbf{Z}$ 中的每个值都映射为一个整数符号，并可反向映射，

$$u_{k,t,c} = \mathrm{round}\big(\mathrm{clip}(z_{k,t,c}/\Delta_c, -u_{\max}, u_{\max})\big), \qquad \hat{z}_{k,t,c} = \Delta_c\, u_{k,t,c}, \tag{5}$$

我们用 $\hat{}\mathbf{Z} = Q_b(\mathbf{Z})$ 表示反量化后的编码。

这些符号通过 rANS [13]进行无损编码。锚点按其位置的 Morton 顺序进行编码。每个符号 $u_{k,t,c}$ 所用的概率表，根据其在锚点内的位置 (t, c)，以及 $\pi(k)$ 中同一 (t,c) 位置符号的粗略分箱来选择；$\pi(k)$ 是在 k 之前编码的最近锚点。这些表是整数频数，针对每个位深度，从训练场景的编码器码中收集一次，并由编码器和解码器固定共享：与网络权重一样，它们属于编解码器，而不是任何比特流的一部分，并且没有学习得到的熵模型。除编码后的符号外，比特流还包含锚点位置 a_k、锚点大小 σ_k 以及逐通道步长Δ_c。若不进行潜变量搜索，存储的编码为 $\hat{}Z^0 = Q_b(Z^0)$；第 4.7 节中的搜索会以相同位深的精化编码取代它。位深 b 决定码率，因此同一个训练模型可用于不同的目标大小。

4.7 潜变量搜索

编码器在许多场景之间进行摊销，因此无法为每个场景找到最佳编码。因此，在编码时，我们像学习式图像压缩中的编码时细化 [7, 46]一样，在冻结编码器和解码器的情况下细化编码。从$\mathbf{Z}^0$, 开始，我们求解

$$\mathbf{Z}^\star = \arg\min_{\mathbf{Z}} \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \mathscr{L}\big(\mathscr{R}(D(\hat{\mathbf{Z}}), \mathbf{P}_v), \mathbf{Y}_v\big), \qquad \hat{\mathbf{Z}} = Q_b(\mathbf{Z}), \tag{6}$$

使用随机梯度下降，并在每一步随机选取一部分 $\mathcal{V}' \subset \mathcal{V}$相机，然后存储 $\hat{}\mathbf{Z}^\star = Q_b(\mathbf{Z}^\star)$。网络权重、锚点$\mathbf{a}_k$、尺寸 σ_k 和步数 Δ_c 均保持不变，因此只有编码符号会改变。量化在循环内部进行，并通过直通估计器 [6]让梯度穿过舍入操作。对于连续令牌，仅在搜索后进行量化，会使 Mip-NeRF 360 上的峰值信噪比在 8 位时降低 0.33 分贝，在 6 位时降低 0.67 分贝（第5.7节）。搜索以目标位深运行，因此可以针对任何所需的量化级别细化编码。目标图像是输入泼溅的渲染图像，$\mathbf{Y}_v = \mathscr{R}(\mathscr{G}, \mathbf{P}_v)$，只需泼溅及其相机即可生成；如果有训练图像，也可以用它们作为目标，$\mathbf{Y}_v = \mathbf{I}_v$。搜索只会略微改变编码，却能显著提高保真度（第5.6节），这表明编码器剩余的大部分误差来自摊销差距，而非解码器表达能力的限制。

5 实验

5.1 实验设置

评估数据集。 在 DL3DV 上， [25], 我们在 173 个留出测试场景上进行评估，这些场景与我们的训练场景没有重叠。包括 FCGS 在内的所有方法都压缩相同的输入泼溅。部分测试场景可能出现在 FCGS 的训练集中，这只会对 FCGS 有利，尽管我们的输入泼溅经过了重新训练，与 FCGS 见过的输入不同。为了衡量对其他采集设置和场景规模的泛化能力，我们进一步在 Mip-NeRF 360 [5], 360、Tanks and Temples [23] 和 Deep Blending [18], 上进行评估，这些数据集中的泼溅最多包含比 DL3DV 训练泼溅多 $10\times$的基元。对于这些数据集，我们遵循 3DGS [21] 中规定的评估协议 [2, 31]。留出视图既不用于拟合输入泼溅，也不作为潜变量搜索的目标，因此所有报告的指标都基于 SplatTok 的搜索从未见过的视图。

输入泼溅。 两项主要比较压缩的是不同的输入泼溅，因此它们的数值不能直接比较。表 1 使用 采用 *gsplat* 训练的原始 *3DGS* 泼溅 [47], 其质量与 FCGS 报告的基础结果相当 [9]，且该表中的每种方法都恰好压缩这些泼溅。表 2使用采用 *3DGS-MCMC* [22], 训练的更强泼溅；对于 Mip-NeRF 360，泼溅数量上限为 350 万个高斯，对于 Tanks and Temples 为 175 万个，对于 Deep Blending 为 100 万个；只有 SplatTok 压缩这些泼溅，而该表中的逐场景方法则将各自的表示拟合到训练图像上。表2 因此是端到端比较，其中每种方法的重建都是其流程的一部分；KISS-GS 同样从 3DGS-MCMC 泼溅开始。

基线方法。不使用训练图像（表1）时，我们将前馈编解码器 FCGS [9] 在低码率和高码率下的表现、无需训练的 FlexGaussian [38], 和 LightGaussian [15] 以及 HAC++ [10]进行比较。我们在输入 splat 的渲染图像上对这些方法进行微调或重新训练，而非使用训练图像。除 FCGS 外，所有方法都使用输入 splat 的相机，这与 SplatTok 的潜变量搜索一致：FlexGaussian 用于评估高斯的重要性，LightGaussian 和 HAC++ 则用于渲染目标图像。使用训练图像时（表2），我们将结果与最强的逐场景方法 ContextGS [42] 和 HAC++ [10]进行比较。由于 HAC 的运行间方差，我们对每个场景重新运行三次；此外，还与最强的训练后方法 KISS-GS [31], 进行比较，该方法也能解码为标准 splat；并与具有代表性的紧凑型方法和训练后方法 [34, 43]进行比较。SplatTok、表1 中的方法以及我们对 HAC++ 的重复运行采用相同的指标实现和分辨率；表2 中的其余数值取自相应论文。

指标。 我们报告留出测试视图的真实图像对应的峰值信噪比、结构相似性指数和学习感知图像块相似度（VGG），压缩文件大小（MB，2^{20} 字节），以及每个场景的压缩时间。除非另有说明，时间不包括拟合输入泼溅所需的时间，且均在单个 GH200 GPU 上测量。SplatTok 的低码率和高码率分别为 $b{=}6$ 和 $b{=}8$ 位，每 128 个输入高斯设置一个锚点。本报告中的所有 SplatTok 结果均来自同一个训练模型，且对每个数据集、位深、锚点密度和搜索目标使用相同的权重；网络不会针对不同数据集、码率或场景重新训练或微调。

训练数据。 我们使用 9,101 个 DL3DV 场景进行训练，并留出 173 个场景进行测试，即排序列表中每第 50 个场景；同时排除泼溅无法良好拟合其训练图像或图像分辨率不标准的场景。我们使用 gsplat 在分辨率为 $480{\times}270$ 的图像上对每个场景的输入泼溅拟合 10k 步。每个训练步骤都会渲染该场景的 32 个随机视角。

架构与训练。每个锚点包含 $T{=}16$ 个令牌，每个令牌有 $C{=}32$ 个通道；每 128 个输入高斯放置一个锚点，每个锚点最多容纳$K{=}640$ 个成员，解码器则为每个锚点生成 $M{=}160$ 个高斯。编码器宽度为 512，包含 12 层，每层有 8 个注意力头，维度为 64；前四层在锚点内进行注意力计算，其余八层交替采用场景级注意力和锚点内注意力。解码器使用一个位于令牌宽度上的交叉注意力层，后接六个宽度为 128 的残差前馈层，以及一个线性输出头。颜色的亮度使用最高至 $\ell{=}3$ 阶的球谐函数，两个色度通道则仅使用直流项。模型包含 57.3M 个参数：编码器占 56.0M（其中 50.4M 位于 12 个 Transformer 层中，5.0M 位于从学习得到的锚点查询到高斯的交叉注意力中，0.6M 位于输入投影、查询与位置嵌入，以及向令牌输出的输出投影中），解码器占 1.25M。我们使用 Muon [20] 优化 86 个权重矩阵（共 56.6M 个参数，学习率放大 3 倍，并使更新幅度与 AdamW 匹配），其余参数则使用 AdamW；权重衰减为 10^{-4}，梯度裁剪阈值为 1.0，$\lambda{=}0.2$ 在公式（3）中采用 bf16 混合精度，批次大小为 16 个场景（包括 4 步梯度累积）。训练共进行 300 轮，学习率为 $5{\times}10^{-5}$，并在此之前进行 10 轮线性预热；训练使用梯度检查点和 FlashAttention-2，在 4 块 NVIDIA A100 40GB GPU 上耗时约 6 天。

潜变量搜索。我们首先搜索 8位编码，然后从中推导出 6位编码。8 位搜索从编码器的编码开始，每一步在 32 个随机训练视图上执行 500 步 Adam 优化，峰值学习率为 0.2，并通过余弦调度衰减至 0.02（在 DL3DV 上则执行 300 步，峰值学习率为 0.1）。存储的编码是迭代结果的指数移动平均，时间常数为步数的 5%。对于 6位编码，我们将 8 位结果舍入到 6 位，并以 0.03 的峰值学习率进一步优化 100 步，同样采用移动平均。这比单独进行 6 位搜索更快，也能得到更好的编码（第 5.6节）。

量化与熵编码。 逐通道步长 Δ_c 根据每个位深度下编码器的编码设置，并在搜索过程中保持不变。rANS编码器使用16位概率和逐字节重新归一化。锚点位置沿每个轴量化为10位，以用于Morton 顺序；上下文邻居 $\pi(k)$是该顺序中位于k之前的32个最近锚点中距离最近的一个。邻居的符号被分入最多64个箱中，另加一个用于没有更早邻居的锚点的箱，

表1. 在没有训练图像的情况下，对原始 3DGS 椭球进行压缩 (而非表 2中的 3DGS-MCMC splat)。所有方法均压缩相同的输入 splat，这些 splat 使用 gsplat 训练 [47]；微调方法以输入 splat 的渲染图像为目标，除 FCGS 外的所有方法均使用其相机。指标基于留出视图的真实图像计算。大小以 MB 为单位（2^{20} 字节），时间以秒/场景为单位，不包括 splat 拟合。由于 DL3DV 包含 173 个场景，逐场景优化方法未在其上运行，因为成本过高。

数据集	DL3DV [25]					Mip-NeRF 360 [5]					Tanks and Temples [23] + Deep Blending [18]				
方法	PSNR↑	SSIM↑	学习感知图像块相似度↓	大小↓	时间↓	PSNR↑	SSIM↑	学习感知图像块相似度↓	大小↓	时间↓	PSNR↑	SSIM↑	学习感知图像块相似度↓	大小↓	时间↓
未压缩															
输入泼溅（原版 3DGS）	30.86	0.905	0.094	82.96	–	27.49	0.813	0.223	749.1	–	26.55	0.872	0.212	551.23	–
免训练压缩															
FlexGaussian [38]	29.31	0.857	0.146	4.14	6	26.78	0.787	0.268	38.42	19	25.66	0.833	0.241	20.21	15
前馈压缩															
FCGS（低码率） [9]	29.21	0.854	0.148	3.28	7	27.05	0.798	0.237	36.24	33	26.27	0.844	0.230	23.20	18
FCGS（高码率） [9]	29.94	0.875	0.132	4.69	9	27.39	0.805	0.226	64.05	44	26.44	0.859	0.219	41.99	22
SplatTok，仅编码器（低码率）	26.59	0.837	0.181	0.73	2.7	22.97	0.707	0.377	6.94	82	24.01	0.804	0.355	5.01	28
SplatTok，仅编码器（高码率）	26.81	0.842	0.194	1.08	2.7	23.05	0.714	0.371	9.85	82	24.18	0.810	0.350	7.35	28
在输入 *splat* 的渲染图像上进行微调或训练															
LightGaussian [15]	–	–	–	–	–	27.33	0.802	0.239	45.30	855	26.15	0.844	0.240	36.82	578
HAC++ （低码率） [10]	–	–	–	–	–	26.76	0.783	0.274	7.01	1980	25.89	0.830	0.256	4.11	1214
HAC++ （高码率） [10]	–	–	–	–	–	26.96	0.794	0.266	15.02	1989	26.01	0.839	0.248	5.76	1221
SplatTok（低码率）	29.87	0.877	0.136	0.74	45	27.21	0.801	0.237	6.99	757	26.34	0.849	0.228	5.04	391
SplatTok（高码率）	30.08	0.889	0.128	1.09	39	27.35	0.805	0.229	9.93	631	26.42	0.858	0.220	7.42	326

并且对于每个位深度，频率表都根据DL3DV训练场景的编码器码统计一次。对于每个（令牌、通道、邻居箱）上下文，它们为每个符号保存一个16位条目：8位时为16.2 MB（16×32×65 个上下文，每个包含255个符号），6位时为3.9 MB（16×32×64 个上下文，每个包含63个符号）。辅助信息以32位浮点数存储锚点位置和大小以及逐通道步长；每个锚点占16字节，每个通道占4字节。

5.2 不使用训练图像进行压缩

表。1 比较了不使用场景训练图像来压缩泼溅的方法；所有方法均根据留出视图的真实图像进行评估，而非根据输入泼溅的渲染图像进行评估。未进行潜变量搜索时，SplatTok 编码器的质量远低于 FCGS：高码率下低 2.3 至 4.3 分贝，尽管其文件已经小了 4 至 6.5 倍。潜变量搜索弥补了这一差距：在高码率下，SplatTok 的峰值信噪比与 FCGS 持平（在 Mip-NeRF 360 和采用 Deep Blending 的 Tanks and Temples 上相差不超过 0.05 分贝，在 DL3DV 上则高出 0.14 分贝），而文件小了 4 至 6.5 倍；在低码率下，SplatTok 的表现超过 FCGS 0.07 至 0.66 分贝，文件小了 4.4 至 5.2 倍。相较于输入泼溅，SplatTok 在两种码率下对 Mip-NeRF 360 和采用 Deep Blending 的 Tanks and Temples 的压缩率分别约为 75 倍和 110 倍，同时仅损失 0.1 至 0.3 分贝。FlexGaussian 和 LightGaussian 保留了标准格式，但其文件比 SplatTok 高码率下的文件大约 3 至 5 倍，峰值信噪比却相当或更低；此外，LightGaussian 每个场景需要 10 至 14 分钟。HAC++在输入泼溅的渲染图像上重新训练后，也能像 SplatTok 一样将每个场景的文件压缩到 10 MB 以下，但质量明显更低。

5.3 与逐场景方法的比较

在训练图像可用的情况下，表 2 比较了面向训练图像进行搜索的 SplatTok 与逐场景方法；后者都需要训练图像。以训练图像为目标时，SplatTok 在所有三种数据集和两种码率下均优于 HAC++这一表现最强的方法：文件更小，峰值信噪比却更高。在 Tanks and Temples 上，文件大小分别缩小 25% 和 34%，峰值信噪比分别高出 0.51 和 0.44 分贝；在 Deep Blending 上，文件大小分别缩小 29% 和 40%，峰值信噪比分别高出 0.18 和 0.15 分贝；在 Mip-NeRF 360 上，文件大小分别缩小 13% 和 41%，峰值信噪比分别高出 0.46 和 0.49 分贝，叶片和细结构也明显更清晰（第 5.4节）。结构相似性指数和学习感知图像块相似度与 HAC++相当；而在 Mip-NeRF 360 和 Tanks and Temples 上，KISS-GS 的高码率结果具有更高的结构相似性指数和更低的学习感知图像块相似度，但文件分别大 17% 和 27%。即使以输入泼溅的渲染图像为目标（因此不使用训练图像），SplatTok 的结果与 HAC++ 相差不超过 0.04 分贝，或在相同码率下最多高出 0.08 分贝，且文件小 17% 至 42%。SplatTok 甚至能达到高于其输入泼溅的质量，例如在 Tanks and Temples 上，原因有二：以训练图像为目标时，潜变量搜索可以修正输入泼溅中的错误；解码器生成的高斯数量约为输入的 1.25 倍。SplatTok 会压缩给定的任何泼溅，此处的泼溅是用 3DGS-MCMC 训练得到的 [22], , 因此会直接继承泼溅重建中的任何改进

表 2。 **在 3DGS-MCMC splat 上与逐场景方法的比较** （不是表 1 中的原始 3DGS 椭球）。所有方法均从相同的训练图像开始，并在相同的留出视图上进行评估；每种方法如何重建场景属于其流程的一部分。HAC++ 和ContextGS拟合各自的表示；KISS-GS 和 SplatTok 压缩 3DGS-MCMC splat。SplatTok 的潜变量搜索以训练图像为搜索目标；对于 SplatTok ◇，则以输入 splat 的渲染图像为搜索目标，且不使用训练图像。HAC++ 为三次重新运行的平均值，与 KISS-GS 的重新评估保持一致。†由作者报告。大小以 MB（2^{20} 字节）为单位。

数据集	Mip-NeRF 360 [5]				Tanks and Temples [23]				Deep Blending [18]			
方法	峰值信噪比↑	结构相似性指数↑	学习感知图像块相似度↓	大小↓	峰值信噪比↑	结构相似性指数↑	学习感知图像块相似度↓	大小↓	峰值信噪比↑	结构相似性指数↑	学习感知图像块相似度↓	大小↓
3DGS† [21]	27.46	0.812	0.222	750.9	23.69	0.844	0.178	431.0	29.42	0.899	0.247	663.9
输入泼溅（3DGS-MCMC）	28.17	0.834	0.189	787.7	24.58	0.872	0.139	393.9	30.12	0.913	0.246	225.1
Reduced3DGS† [34]	27.19	0.807	0.230	29.54	23.57	0.840	0.188	14.00	29.63	0.902	0.249	18.00
MesonGS-FT† [43]	26.99	0.796	0.247	27.16	23.32	0.837	0.193	16.99	29.51	0.901	0.251	24.76
ContextGS（低码率）† [42]	27.62	0.808	0.237	12.68	24.20	0.852	0.184	7.05	30.11	0.907	0.265	3.45
ContextGS（高码率）† [42]	27.75	0.811	0.231	18.41	24.29	0.855	0.176	11.80	30.39	0.909	0.258	6.60
HAC++ （低码率） [10]	27.47	0.802	0.255	8.61	24.16	0.848	0.190	5.26	30.00	0.900	0.269	2.94
HAC++ （高码率） [10]	27.60	0.806	0.233	18.48	24.33	0.853	0.178	8.62	30.19	0.905	0.259	5.31
KISS-GS† （低码率） [31]	27.75	0.809	0.239	7.17	24.48	0.853	0.183	3.62	29.45	0.896	0.288	1.87
KISS-GS† （高码率） [31]	27.94	0.823	0.208	12.76	24.73	0.868	0.151	7.16	29.75	0.901	0.268	3.73
SplatTok ◇ （低码率）	27.53	0.801	0.223	7.13	24.24	0.835	0.196	3.71	30.04	0.903	0.273	2.06
SplatTok ◇ （高码率）	27.68	0.808	0.216	10.64	24.32	0.840	0.189	5.47	30.15	0.906	0.269	3.06
SplatTok（低码率）	27.93	0.813	0.222	7.46	24.67	0.847	0.183	3.94	30.18	0.903	0.271	2.08
SplatTok（高码率）	28.09	0.820	0.215	10.89	24.77	0.854	0.177	5.65	30.34	0.911	0.258	3.20

；而逐场景方法会构建自己的表示，必须重新设计才能从中获益。在其他方法中，ContextGS 要达到相当的峰值信噪比，文件需大 1.7 至 2.1 倍；KISS-GS 的文件大小相近，但峰值信噪比最多低 0.7 分贝；剪枝并量化基元的方法则要大 3.5 至 12 倍。在更稀疏的锚点密度下，相较于 HAC++ 的优势会进一步扩大（第 5.5节）。

5.4 定性结果

图 3 比较了 SplatTok 与 HAC++ 在 Mip-NeRF 360 数据集上的两种码率表现。SplatTok 能够保留细微的高频内容，而 HAC 会将其++ 平滑掉，例如叶片、细结构以及掠射角表面。图 5至 8 进一步展示了两种码率下四个数据集上的结果：在 DL3DV 上，潜变量搜索以输入 splat 的渲染图像为目标；在 Mip-NeRF 360、Tanks and Temples 和 Deep Blending 上，潜变量搜索则以训练图像为目标。从 8 位降至 6 位可将文件大小减少约三分之一，同时几乎不会产生可见变化。

5.5 码率控制

由于每个锚点都使用相同数量的令牌进行编码，因此编码的大小由两个量决定：令牌的位深和锚点的数量。对于这两者的所有设置，都使用同一个训练模型；由于锚点的数量取决于输入高斯的数量，因此预先压缩输入泼溅可作为第三个控制因素。

位深。 表。 3 改变了 卡车上令牌的精度。将精度从 8 位降至 6 位和 5 位时，仅损失 0.15 和 0.35 分贝，而降至 4 位时损失 1.1 分贝；同一个训练模型和潜变量搜索即可覆盖整个范围。

锚点密度。表。4采用以每 128 个输入高斯配置一个锚点训练的模型，不进行重新训练，而是将锚点密度减半，使每个锚点概括约两倍数量的高斯，且解码器输出的高斯数量约为输入的 0.6 倍。模型能够很好地处理较稀疏的锚点：在 8 位下，每 256 个高斯配置一个锚点，其峰值信噪比与 6 位下每 128 个高斯配置一个锚点的结果相当或几乎相当（差距在 0.02 分贝以内，且在 Tanks and Temples 上高出 0.08 分贝），文件大小则减少 19% 至 23%。因此，在低码率下，相比采用更多低精度锚点，采用更少的高精度锚点能更有效地利用位数。在 6 位下，较稀疏的锚点使每个场景的文件大小达到 1.2 至 4.0 MB，且峰值信噪比仍优于 HAC++ 的低码率设置（表。2）：在 Mip-NeRF 360 上高出 0.29 分贝，在 Tanks and Temples 上高出 0.41 分贝，在 Deep Blending 上高出 0.07 分贝，同时文件大小减少 53% 至 60%。锚点数量减半后，潜变量搜索的每一步解码的高斯数量也减半，搜索时间因此大致减半。精细细节的损失

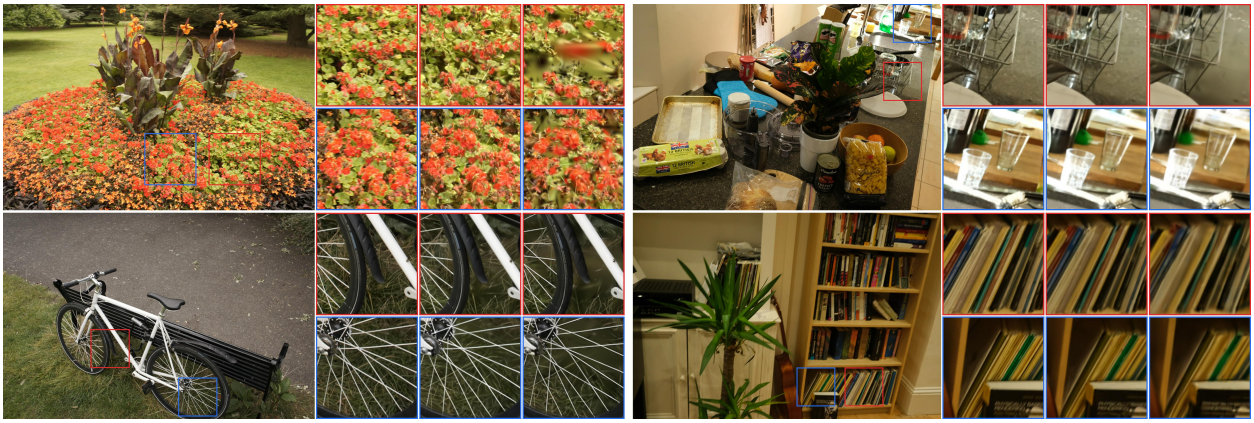

图3。 在 Mip-NeRF 360 上进行定性比较。 左侧（花朵 和*Bicycle*）：高码率。右侧（房间和 柜台）：低码率。每组中，最左侧的图像是真实图像，随后是对真实图像、SplatTok 和 HAC 的放大裁剪图++。与表 2中的设置相同，SplatTok 的潜变量搜索以训练图像为目标。与 HAC 相比，SplatTok 在两种码率下都能更好地保留叶片、倾斜表面和细结构的细节++ 。

表3。 位深 在 卡车 (Tanks and Temples) 上的结果，潜变量搜索以训练图像为目标。每值比特数仅指编码令牌。

位深	峰值信噪比↑	大小↓	比特/值↓
$b=4$	25.46	2.24	2.40
$b=5$	26.21	3.01	3.31
$b=6$ （低码率）	26.38	3.71	4.14
$b=7$	26.49	4.59	5.18
$b=8$ （高码率）	26.51	5.47	6.23

表4。 锚点密度：每128个输入高斯设置一个锚点（与训练时相同），或在表2的3DGS-MCMC splat上每256个设置一个锚点，2，潜变量搜索指向训练图像。时间单位为每个场景的秒数。

数据集	G./锚点 位	峰值信噪比↑	结构相似性指数↑	学习感知图像块相似度↓	大小↓	时间↓
Mip-NeRF 360	128	8	28.09	0.820	0.215	10.89	890
	128	6	27.93	0.813	0.222	7.46	810
	256	8	27.93	0.810	0.236	5.85	531
	256	6	27.76	0.802	0.243	4.04	559
Tanks and	128	8	24.77	0.854	0.177	5.65	347
Temples	128	6	24.67	0.847	0.183	3.94	385
	256	8	24.75	0.843	0.200	3.03	188
	256	6	24.57	0.836	0.206	2.14	202
Deep Blending	128	8	30.34	0.911	0.258	3.20	198
	128	6	30.18	0.903	0.271	2.08	220
	256	8	30.16	0.903	0.278	1.68	106
	256	6	30.07	0.901	0.281	1.19	116

然而，与整体保真度相比，精细细节受损更为明显：由于每个区域解码的高斯数量更少，在 6 位下，结构相似性指数和学习感知图像块相似度相较默认密度的下降幅度比峰值信噪比更大。这表明可以在场景内调整锚点密度，让复杂区域获得比简单区域更多的令牌（第6节）。

压实后的输入。 SplatTok 可处理任意标准 splat，包括已通过压缩方法缩减的泼溅；压实输入会按比例缩小其比特流。表。 5 将 SplatTok 应用于 KISS-GS 在自身编码前生成的 512k 高斯 POPSpa 泼溅 [31] 。在 8 位下，SplatTok 可将其压缩至 1.74 MB，达到 26.72 分贝，与 KISS-GS 在其 128k 工作点的表现相当（1.84 MB 时为 26.69 分贝），且比 KISS-GS 对同一批 512k 泼溅进行编码后的结果小 4.1 倍。在 6 位下，它以 1.25 MB 达到 26.50 分贝，优于 KISS-GS 曲线（该曲线在此大小处插值为 26.19 分贝），同时结构相似性指数更高、学习感知图像块相似度更低。因此，SplatTok 使用压实程度较低的泼溅，也能达到或超越 KISS-GS 的权衡表现，并可作为压缩流水线的最终编码阶段。

5.6 潜变量搜索与摊销差距

图 4 展示了潜变量搜索过程中质量和编码的变化情况。搜索能快速改善编码器生成的编码：在 Mip-NeRF 360 上，8位编码在前 100 步内提升了 3.7 分贝，经过 500 步后趋于稳定，达到 28.09 分贝，而编码相较于编码器的输出变化不到 10%。将搜索得到的 8位编码舍入为 6 位，并再优化 100 步后，可达到 27.93 分贝，高于从头开始进行 6 位搜索的结果（27.85 分贝），且成本仅为其一小部分；由此得到的 6位编码变化幅度也在相同范围内。搜索对码率的影响微乎其微：熵编码后的比特流大小变化不到

表 5。压缩压实后的泼溅在 Mip-NeRF 360 上的表现。使用 POPSpa（KISS-GS 的压实阶段）将输入泼溅压实为每个场景 512k 个高斯体，作者将其分享给了我们。与 KISS-GS 的微调一样，SplatTok 使用面向训练图像的潜变量搜索，以 8 位和 6 位对其进行压缩。†数据读取自 KISS-GS 的率失真图（采用编码感知微调的 SOG-XT 编码）。‡在 SplatTok 对应的大小处，根据其 64k 和 128k 工作点之间的对数大小进行线性插值。大小以 MB 为单位（2^{20} 字节）。

方法	#高斯体	峰值信噪比↑	结构相似性指数↑	学习感知图像块相似度↓	大小↓
POPSpa 泼溅（未压缩输入）	512 k	27.79	0.813	0.239	–
KISS-GS†，512k 预算	512 k	27.75	0.809	0.239	7.17
KISS-GS†，128k 预算	128 千	26.69	0.754	0.322	1.84
KISS-GS‡，插值	–	26.62	0.750	0.327	1.74
KISS-GS‡，插值	–	26.19	0.725	0.355	1.25
POPSpa 泼溅 +SplatTok（8 位）	–	26.72	0.751	0.325	1.74
POPSpa 泼溅 +SplatTok（6 位）	–	26.50	0.730	0.332	1.25

比特流变化小于 1%。

以输入泼溅在留出测试视图上的渲染图像为基准进行衡量（搜索过程中从未使用这些视图），增益约为 8 分贝，但编码按相对均方根误差计算仅变化了 7% 至 8%；不同数据集的变化幅度从 DL3DV 上的约 4% 到 Tanks and Temples 上的 10% 不等。因此，使用相同的冻结解码器，搜索消除了相对于输入泼溅的编码器平方误差的 84%。解码器所能接受的最佳编码至少与搜索得到的编码一样好，因此，编码器误差中至少这一部分属于摊销差距，而最多只有 16% 可归因于解码器。此外，效果更好的编码与编码器给出的编码相近：编码器必须同时服务于所有场景，但对每个场景的编码都几乎正确，这也解释了为何在网络冻结的情况下进行短暂搜索就足够。两种搜索目标表现相近，而朝着训练图像进行搜索还可额外提升 0.4 分贝，因为真实图像中也包含输入泼溅本身无法再现的细节。

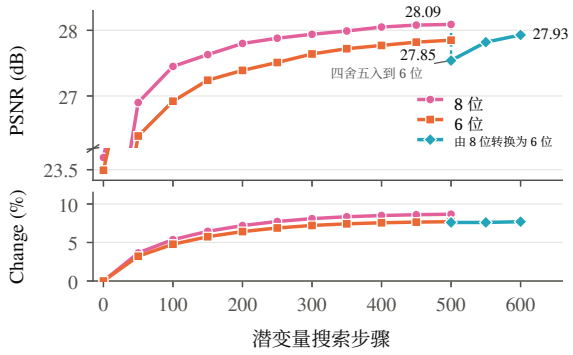

图4。 在 Mip-NeRF 360 上进行潜变量搜索，对其各个场景取平均值，朝训练图像方向进行。上：留出视图上的峰值信噪比。下：潜在编码相对于编码器输出的相对均方根变化。

搜索是否需要编码器？。 为测试这一点，我们将卡车 的令牌初始化为随机值，这些值来自 50 个随机 DL3DV 火车场景中令牌的逐通道均值和方差，保留输入 splat 的锚点、锚点大小和量化步长，并使用相同的协议运行潜变量搜索。编码从 7.25 分贝开始，在相同的步数后仅达到 21.44 分贝，而从编码器的输出开始时则达到 26.51 分贝。冻结解码器

表6。 熵编码 应用于 卡车 的高码率编码。各变体均无损，仅大小不同。

概率表	大小↓	比特/值↓
单一直方图	6.17	7.06
逐位置	5.49	6.26
逐位置 +邻域上下文	5.47	6.23

alone is thus not en 足够：编码器将搜索置于一个良好的编码附近，而搜索弥合了剩余的差距 .

5.7 消融实验

全局注意力。 为了衡量编码器本身的性能，我们比较不经过潜变量搜索的编码，因为潜变量搜索可能会弥补编码器较弱的问题。在大小相同的情况下，将全局注意力层替换为每个锚点内部的注意力，会使 DL3DV 测试场景上的纯编码器峰值信噪比降低 0.82 分贝，这证实了场景级上下文能够提供超出各锚点自身高斯所包含的信息，从而改善编码。潜变量搜索缩小了这一差距，但经过相同次数的搜索步骤后，仍相差 0.26 分贝。仅局部编码器的前向传播成本更低，但它无法加快最慢的阶段——潜变量搜索；该阶段仅使用冻结解码器。

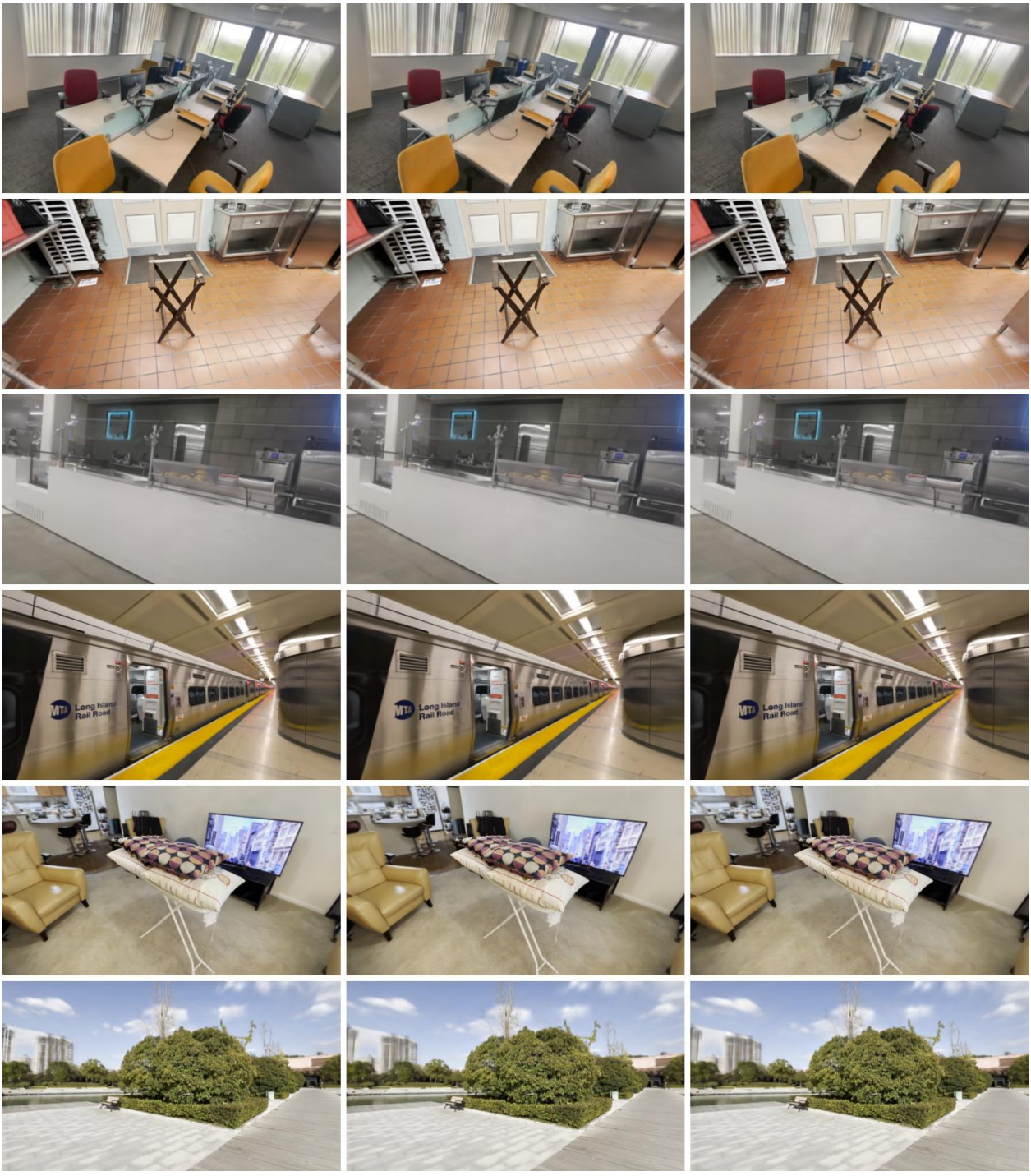

图 5. DL3DV 测试场景。 左：输入 splat 的渲染图像；中：8 位的 SplatTok；右：6 位的 SplatTok。潜变量搜索以输入 splat 的渲染图像为搜索目标。

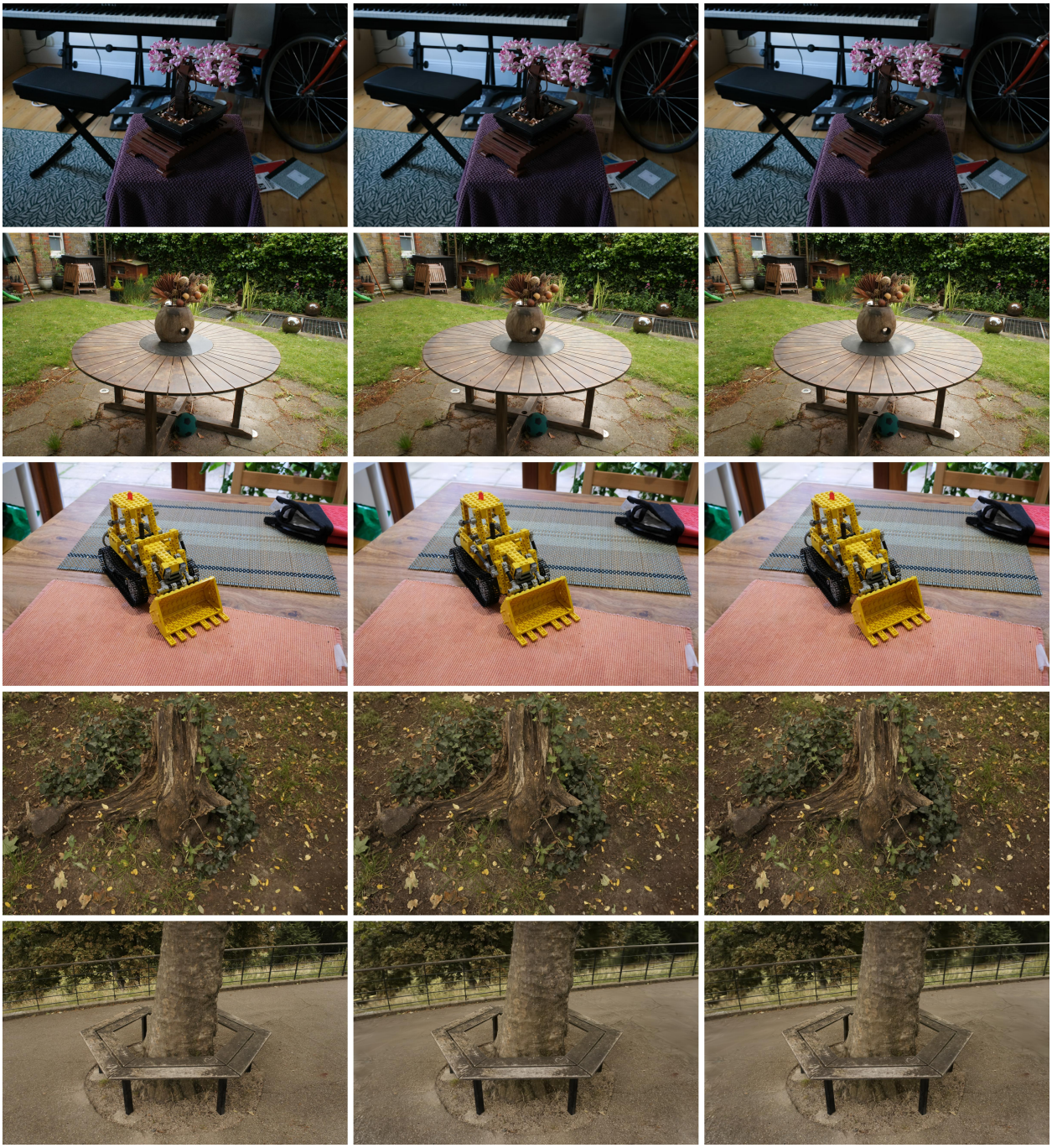

图6. Mip-NeRF 360. 左：真实图像；中：8 位的 SplatTok；右：6 位的 SplatTok。潜变量搜索的目标是训练图像。



图 7。 **Tanks and Temples**。 左：真实图像；中：8 位的 SplatTok；右：6 位的 SplatTok。潜变量搜索的目标是训练图像。



图 8. **Deep Blending.** 左：真实图像；中：8 位的 SplatTok；右：6 位的 SplatTok。潜变量搜索的目标是训练图像。

循环量化。 对连续令牌运行潜变量搜索、仅在最后进行量化，会使 Mip-NeRF 360 上的峰值信噪比在 8 位时降低 0.33 分贝，在 6 位时降低 0.67 分贝，因为搜索找到的连续编码无法经受舍入；而在循环中进行量化时，搜索找到的编码在量化后仍然表现良好。

熵编码。 根据锚点内的位置选择表，相比单一直方图可节省 11% 的位；邻近锚点上下文则仅能额外节省 0.5%，且质量没有变化，因为编码是无损的（表 6）。因此，计数直方图仅捕捉到锚点之间少量的依赖关系，这为学习式熵模型留出了空间（第 6节）。

5.8 计时与渲染速度

表 7 分解了不同规模场景中各阶段的耗时。潜变量搜索和解码所需时间大致随锚点数量线性增长，因而也随输入高斯数量线性增长；而聚类和编码的耗时增长更快，其中编码因全局注意力而呈二次增长。在最大的场景中，潜变量搜索仍占总耗时的约 80%；GPU 加速聚类还能进一步缩短其余部分。即使对于最大的场景，解码已存储的编码也远不到一秒。解码器每个锚点输出 160 个高斯，而每 128 个输入高斯放置一个锚点，因此解码后的泼溅所含高斯数量约为输入的 1.25 倍；其渲染成本因此与输入泼溅相当：在 Mip-NeRF 360 的 花园场景中，每秒 171 帧，对比 184 帧（高斯数量分别为 440 万和 350 万，每次渲染一帧宽 1600 像素的画面）。

表 7. 各阶段耗时 不同大小场景的耗时，单位为秒，测试平台为单块 GH200 GPU。聚类将场景划分为锚点；编码和解码各需一次网络前向传播；潜变量搜索按默认步数运行。

高斯	锚点	聚类	编码	潜变量搜索	熵解码	解码
350 k	2736	3.1	0.10	48	0.05	0.04
1 M	7813	16.6	0.51	179	0.13	0.11
3.5 M	27559	143	5.1	672	0.48	0.44

在压缩比相近的方法中，SplatTok 的压缩耗时相当或更短。对于相同的输入泼溅和图形处理器，HAC++ 在输入泼溅的渲染图像上重新训练，在 Mip-NeRF 360 上每个场景耗时约 33 分钟，而 SplatTok 耗时 10 至 13 分钟（表 1）。KISS-GS [31]报告了单个场景 *Bicycle*在一块 A100 图形处理器上的耗时，该场景来自 Mip-NeRF 360：在较高码率下，压实、编码和编码感知微调耗时约 950 秒，不包括输入泼溅的重建；而 SplatTok 处理含 350 万个高斯的泼溅时耗时约 820 秒（表 7）。由于 KISS-GS 的数据是从论文中的运行时间图表读取的，且测量所用硬件不同，因此这一比较仅为近似值。FCGS 和 FlexGaussian 等前馈且无需训练的方法可在数秒内压缩一个场景（表 1），但生成的文件大 3 至 6.5 倍。

6 讨论

局限性。 潜变量搜索是最耗时的阶段：在包含一百万个高斯的场景中，耗时近三分钟，远超编码器的一次前向传播。对于大型场景，编码器也需要大量显存：处理四百万个高斯时约需 30GB，因为其全局注意力覆盖场景中的所有令牌；采用分块实现可以降低这一需求。

加速潜变量搜索。 在包含大量高斯的场景中，每个搜索步骤中最耗费计算的部分显然是解码所有锚点。由于每一步只渲染少数几个相机视角，只解码这些相机视角可见的锚点便能降低这项开销。更少的锚点也能缩短搜索时间，正如第 5.5 节所示。

自适应锚点密度。 锚点密度实验（第 5.5 节）表明，同一模型能够很好地处理更稀疏的锚点；但当每个区域分配的令牌都减少时，精细细节受到的影响会大于整体保真度。与对每个输入高斯均匀分配锚点相比，根据局部复杂度分配锚点，能将位用在最关键之处。

学习式熵模型。 统计得到的概率表很少捕捉相邻锚点之间的依赖关系（第 5.7 节）。一种以已解码令牌为条件的学习式熵模型，如学习式图像压缩中所采用的模型 [4,29], 可以进一步降低码率，而在潜变量搜索中加入码率项，则能够让模型在每个场景中权衡大小与质量。

随数据规模扩展。 作为一种学习型编解码器，SplatTok 应能进一步受益于更多训练数据：在不进行潜变量搜索的情况下，编码器在训练场景上的得分比未见场景高 1.8 分贝，这表明 9,101 个场景尚未覆盖真实采集的多样性，而潜变量搜索能够缩小未见场景上的这一差距。改进用于训练自编码器的数据增强，例如在场景尺度、密度和采集风格方面进行增强，是减少这一差距的另一种互补途径。由于潜变量搜索主要用于补偿摊销编码中的误差，因此，在规模更大、更多样化的数据上训练出更好的编码器，也可能减少达到给定质量所需的搜索步骤数。

超越压缩的场景标记。除了压缩之外，这些令牌还构成了对整个场景的紧凑三维锚定描述：只需数千个锚点，而非数百万个基元。这使它们成为场景生成、编辑、检索和语义理解等下游任务的天然接口。

7 结论

SplatTok 并非对给定泼溅的基元进行编码，也不是从头开始为每个场景优化新表示，而是重新合成场景：它从数千个泼溅中学习空间锚定令牌的潜空间，并在网络冻结的情况下，通过在该空间中搜索对应编码来压缩泼溅。单个模型无需训练图像，即可按多种码率压缩来自不同数据集的泼溅；在可用这些图像时，它以更小的尺寸实现了优于逐场景方法的峰值信噪比，并可解码为标准 splat。由于它压缩的是任意给定的泼溅，因此也能继承未来泼溅重建技术的改进。

致谢

本研究始于 Sai Karthikey Pentapati 在 Dolby实验室 实习期间。我们还感谢德克萨斯高级计算中心（TACC）授予我们访问 Vista服务器 的权限。感谢 KISS-GS 的作者分享其经 POPSpa 压缩的泼溅。

参考文献

[1] Mohammad Asim、Christopher Wewer 和 Jan Eric Lenssen。SceneTok：面向三维场景的压缩式、可扩散令牌空间。发表于*CVPR*，2026 年。[2] Milena T. Bagdasarian、Paul Knoll、Yi-Hsin Li、Florian Barthel、Anna Hilsmann、Peter Eisert 和 Wieland Morgenstern。3DGS.zip：三维高斯泼溅压缩方法综述。发表于《计算机图形学论坛》，2025 年。[3] Johannes Ballé、Valero Laparra 和 Eero P. Simoncelli。端到端优化图像压缩。发表于*ICLR*，2017 年。[4] Johannes Ballé、David Minnen、Saurabh Singh、Sung Jin Hwang 和 Nick Johnston。基于尺度超先验的变分图像压缩。发表于*ICLR*，2018 年。[5] Jonathan T. Barron、Ben Mildenhall、Dor Verbin、Pratul P. Srinivasan 和 Peter Hedman。Mip-NeRF 360：无界抗锯齿神经辐射场。发表于*CVPR*，2022 年。[6] Yoshua Bengio、Nicholas Léonard 和 Aaron Courville。条件计算中随机神经元梯度的估计或传播。arXiv 预印本 arXiv:1308.3432，2013 年。[7] Joaquim Campos、Simon Meierhans、Abdelaziz Djelouah 和 Christopher Schroers。神经图像压缩的内容自适应优化。发表于*CVPRW*，2019 年。[8] Yihang Chen、Qianyi Wu、Weiyao Lin、Mehrtash Harandi 和 Jianfei Cai。HAC：用于三维高斯泼溅压缩的哈希网格辅助上下文。发表于*ECCV*，2024 年。[9] Yihang Chen、Qianyi Wu、Mengyao Li、Weiyao Lin、Mehrtash Harandi 和 Jianfei Cai。快速前馈三维高斯泼溅压缩。发表于*ICLR*，2025 年。[10] Yihang Chen、Qianyi Wu、Weiyao Lin、Mehrtash Harandi 和 Jianfei Cai。HAC++：迈向三维高斯泼溅的 100 倍压缩。发表于*IEEE TPAMI*，47(11):10210–10226，2025 年。[11] Yihang Chen、Mengyao Li、Qianyi Wu、Weiyao Lin、Mehrtash Harandi 和 Jianfei Cai。PCGS：三维高斯泼溅的渐进式压缩。发表于*AAAI*，2026 年。

[12] Xuan Deng、Xiandong Meng、Hengyu Man、Qiang Zhu、Tiange Zhang、Debin Zhao 和 Xiaopeng Fan。LG-HCC：面向三维高斯泼溅的局部几何感知层次化上下文压缩。arXiv 预印本 arXiv:2603.28431，2026 年。[13] Jarek Duda。非对称数制系统：结合霍夫曼编码速度与算术编码压缩率的熵编码方法。arXiv 预印本 arXiv:1311.2540，2013 年。[14] Emilien Dupont、Adam Goli´nski、Milad Alizadeh、Yee Whye Teh 和 Arnaud Doucet。COIN：基于隐式神经表示的压缩。发表于 ICLR 神经压缩研讨会，2021 年。[15] Zhiwen Fan、Kevin Wang、Kairun Wen、Zehao Zhu、Dejia Xu 和 Zhangyang Wang。LightGaussian：无界 3D 高斯压缩，体积缩减 15 倍，并实现 200+ FPS。发表于 NeurIPS，2024 年。[16] Quankai Gao、Iliyan Georgiev、Tuanfeng Y. Wang、Krishna Kumar Singh、Ulrich Neumann 和 Jae Shin Yoon。Can3Tok：场景级三维高斯的规范三维分词与潜在建模。发表于 ICCV，2025 年。[17] Sharath Girish、Kamal Gupta 和 Abhinav Shrivastava。EAGLES：采用轻量级编码的高效加速 3D 高斯。发表于 ECCV，2024 年。[18] Peter Hedman、Julien Philip、True Price、Jan-Michael Frahm、George Drettakis 和 Gabriel Brostow。用于自由视点图像渲染的 Deep Blending。发表于 ACM TOG，37(6)，2018 年。[19] Qiang Hu、Zhenlong Wu、Lei Huang、Zihan Zheng、Xiaoyun Zhang 和 Wenjun Zhang。GenSplatCodec：通过单步扩散实现前馈式高斯泼溅压缩。arXiv 预印本 arXiv:2607.24403，2026 年。[20] Keller Jordan、Yuchen Jin、Vlado Boza、Jiacheng You、Franz Cesista、Laker Newhouse 和 Jeremy Bernstein。Muon：用于神经网络隐藏层的优化器。https://kellerjordan.github.io/posts/muon/，2024 年。[21] Bernhard Kerbl、Georgios Kopanas、Thomas Leimkuehler 和 George Drettakis。用于实时辐射场渲染的三维高斯泼溅。发表于 ACM TOG，42(4)，2023 年。[22] Shakiba Kheradmand、Daniel Rebain、Gopal Sharma、Weiwei Sun、Yang-Che Tseng、Hossam Isack、Abhishek Kar、Andrea Tagliasacchi 和 Kwang Moo Yi。将三维高斯泼溅视为马尔可夫链蒙特卡洛。发表于 NeurIPS，2024 年。[23] Arno Knapitsch、Jaesik Park、Qian-Yi Zhou 和 Vladlen Koltun。Tanks and Temples：大规模场景重建基准测试。发表于 ACM TOG，36(4)，2017 年。[24] Joo Chan Lee、Daniel Rho、Xiangyu Sun、Jong Hwan Ko 和 Eunbyung Park。用于辐射场的紧凑三维高斯表示。发表于 CVPR，2024 年。[25] Lu Ling、Yichen Sheng、Zhi Tu、Wentian Zhao、Cheng Xin、Kun Wan、Lantao Yu、Qianyu Guo、Zixun Yu、Yawen Lu、Xuanmao Li、Xingpeng Sun、Rohan Ashok、Aniruddha Mukherjee、Hao Kang、Xiangrui Kong、Gang Hua、Tianyi Zhang、Bedrich Benes 和 Aniket Bera。DL3DV-10K：用于基于深度学习的三维视觉的大规模场景数据集。发表于 CVPR，2024 年。[26] Xiangrui Liu、Xinju Wu、Pingping Zhang、Shiqi Wang、Zhu Li 和 Sam Kwong。CompGS：通过压缩高斯泼溅实现高效三维场景表示。发表于 ACM MM，2024 年。[27] Zhening Liu、Rui Song、Yushi Huang、Yingdong Hu、Xinjie Zhang、Jiawei Shao、Zehong Lin 和 Jun Zhang。基于长上下文建模的前馈式三维高斯泼溅压缩。arXiv 预印本 arXiv:2512.00877，2025 年。[28] Tao Lu、Mulin Yu、Linning Xu、Yuanbo Xiangli、Limin Wang、Dahua Lin 和 Bo Dai。Scaffold-GS：用于视点自适应渲染的结构化三维高斯。发表于 CVPR，2024 年。[29] David Minnen、Johannes Ballé 和 George D. Toderici。学习式图像压缩的联合自回归与层次先验。发表于 NeurIPS，2018 年。[30] Wieland Morgenstern、Florian Barthel、Anna Hilsmann 和 Peter Eisert。通过自组织高斯网格实现紧凑三维场景表示。发表于 ECCV，2024 年。[31] Wieland Morgenstern、Friedrich Elias Branschke、Florian Fleischmann、Adrian Szatmari、Paul Schlack、Florian Barthel、Peter Eisert 和 Anna Hilsmann。KISS-GS：化繁为简的三维高斯泼溅压缩。发表于 ECCV，2026 年。[32] K. L. Navaneet 、 Kossar Pourahmadi Meibodi 、 Soroush Abbasi Koohpayegani 和 Hamed Pirsiavash。CompGS：通过矢量量化实现更小、更快的高斯泼溅。发表于 ECCV，2024 年。

[33] Simon Niedermayr、Josef Stumpfegger 和 Rüdiger Westermann。用于加速新视角合成的压缩 3D 高斯泼溅。发表于 *CVPR*，2024 年。[34] Panagiotis Papantonakis、Georgios Kopanas、Bernhard Kerbl、Alexandre Lanvin 和 George Drettakis。减少 3D 高斯泼溅的内存占用。发表于 *ACM* 计算机图形学与交互技术会议录，7(1)，2024 年。[35] Jiawei Ren、Michal Jan Tyszkiewicz、Jiahui Huang 和 Zan Gojcic。TokenGS：利用可学习令牌将 3D 高斯预测与像素解耦。发表于 *CVPR*，2026 年。[36] Barbara Roessle、Norman Müller、Lorenzo Porzi、Samuel Rota Bulò、Peter Kontschieder、Angela Dai 和 Matthias Nießner。L3DG：潜在 3D 高斯扩散。发表于 *SIGGRAPH Asia 2024* 会议论文，2024 年。[37] Jianlin Su、Murtadha Ahmed、Yu Lu、Shengfeng Pan、Wen Bo 和 Yunfeng Liu。RoFormer：采用旋转位置嵌入的增强型 Transformer。发表于 *Neurocomputing*，568：127063，2024 年。[38] Boyuan Tian、Qizhe Gao、Siran Xianyu、Xiaotong Cui 和 Minjia Zhang。FlexGaussian：灵活且经济高效的免训练 3D 高斯泼溅压缩。发表于 *ACM MM*，2025 年。[39] Nicolas von Lützow、Barbara Rössle、Katharina Schmid 和 Matthias Nießner。GaussianGPT：迈向自回归 3D 高斯场景生成。发表于 *ECCV*，2026 年。[40] Henan Wang、Hanxin Zhu、Tianyu He、Runsen Feng、Jiajun Deng、Jiang Bian 和 Zhibo Chen。端到端率失真优化 3D 高斯表示。发表于 *ECCV*，2024 年。[41] Jianyuan Wang、Minghao Chen、Nikita Karaev、Andrea Vedaldi、Christian Rupprecht 和 David Novotny。VGGT：视觉几何基础 Transformer。发表于 *CVPR*，2025 年。[42] Yufei Wang、Zhihao Li、Lanqing Guo、Wenhan Yang、Alex C. Kot 和 Bihan Wen。ContextGS：采用锚点级上下文模型的紧凑型 3D 高斯泼溅。发表于 *NeurIPS*，2024 年。[43] Shuzhao Xie、Weixiang Zhang、Chen Tang、Yunpeng Bai、Rongwei Lu、Shijia Ge 和 Zhi Wang。MesonGS：通过高效属性变换对 3D 高斯进行后训练压缩。发表于 *ECCV*，2024 年。[44] Hao Xu、Xiaolin Wu 和 Xi Zhang。学习用于 3DGS 压缩的分层稀疏变换编码。arXiv 预印本 arXiv:2505.22908，2025 年。[45] Runjie Yan、Yan-Pei Cao、Peng Wang、Ding Liang 和 Yuan-Chen Guo。采用学习式密度控制的生成式 3D 高斯。发表于*ACMSIGGRAPH*，2026 年。[46] Yibo Yang、Robert Bamler 和 Stephan Mandt。改进神经图像压缩的推理。发表于 *NeurIPS*，2020 年。[47] Vickie Ye、Ruilong Li、Justin Kerr、Matias Turkulainen、Brent Yi、Zhuoyang Pan、Otto Seiskari、Jianbo Ye、Jeffrey Hu、Matthew Tancik 和 Angjoo Kanazawa。gsplat：用于高斯泼溅的开源库。发表于 机器学习研究杂志，26(34)：1–17，2025 年。[48] Pengpeng Yu、Yueru Chen、Fei Song、Tai Qin、Qi Zhang、Jing Wang 和 Yulan Guo。迈向实用的 3D 高斯泼溅压缩。arXiv 预印本 arXiv:2609.30245，2026 年。[49] Pengpeng Yu、Runqing Jiang、Qi Zhang、Dingquan Li、Jing Wang 和 Yulan Guo。CodecSplat：用于前馈式 3D 高斯泼溅的超紧凑潜在编码。发表于 *NeurIPS*，2026 年。[50] Yu-Ting Zhan、Cheng-Yuan Ho、Hebi Yang、Yi-Hsin Chen、Jui Chiu Chiang、Yu-Lun Liu 和 Wen-Hsiao Peng。CAT-3DGS：一种用于率失真优化 3DGS 压缩的上下文自适应三平面方法。发表于 *ICLR*，2025 年。[51] Biao Zhang、Jiapeng Tang、Matthias Nießner 和 Peter Wonka。3DShape2VecSet：面向神经场和生成式扩散模型的 3D 形状表示。发表于 *ACM TOG*，42(4)，2023 年。[52] Jun-Teng Zhang、Tong Chen、Yu-Xin Zhao、Yi-Bo Shi、Jing Wang 和 Zhan Ma。Speedygs：通过两阶段优化实现内容感知的 3D 高斯泼溅压缩。发表于 第 *34* 届 *ACM* 多媒体国际会议，2026 年。[53] Longwen Zhang、Ziyu Wang、Qixuan Zhang、Qiwei Qiu、Anqi Pang、Haoran Jiang、Wei Yang、Lan Xu 和 Jingyi Yu。CLAY：用于创建高质量 3D 资产的可控大规模生成模型。发表于 *ACM TOG*，43(4)，2024 年。

A 逐场景结果

表格。8 至10 列出了 SplatTok 在 Mip-NeRF 360、Tanks and Temples 和 Deep Blending 上的逐场景结果，输入为 3DGS-MCMC 泼溅，并针对训练图像进行潜变量搜索，如表2所示。对于每个场景，我们报告低码率和高码率下留出测试视图的峰值信噪比、结构相似性指数和学习感知图像块相似度；每个表格的最后一行给出各场景的平均值。

表 8. 在 Mip-NeRF 360 上的逐场景结果 使用 3DGS-MCMC 输入泼溅，并针对训练图像进行潜变量搜索。

	低码率			高码率		
场景	峰值信噪比↑	结构相似性指数↑	学习感知图像块相似度↓	峰值信噪比↑	结构相似性指数↑	学习感知图像块相似度↓
Bicycle	25.18	0.734	0.264	25.35	0.747	0.256
花朵	21.88	0.598	0.351	21.95	0.605	0.346
花园	27.01	0.824	0.175	27.22	0.835	0.165
树桩	26.98	0.783	0.238	27.15	0.792	0.229
树丘	23.32	0.653	0.327	23.34	0.659	0.319
盆景	32.85	0.948	0.171	33.08	0.952	0.166
柜台	29.61	0.914	0.174	29.73	0.920	0.168
厨房	31.64	0.928	0.121	31.90	0.933	0.115
Room	32.95	0.936	0.176	33.10	0.940	0.171
Mean	27.93	0.813	0.222	28.09	0.820	0.215

表 9. Tanks and Temples 上的逐场景结果，与表8相同。

	低码率			高码率		
场景	峰值信噪比	结构相似性指数	学习感知图像块相似度	峰值信噪比	结构相似性指数	学习感知图像块相似度
火车	22.97	0.814	0.214	23.02	0.820	0.208
卡车	26.38	0.881	0.152	26.51	0.887	0.145
Mean	24.67	0.847	0.183	24.76	0.854	0.177

表10。Deep Blending上的逐场景结果，如表8所示。

	低码率			高码率		
场景	峰值信噪比	结构相似性指数	学习感知图像块相似度	峰值信噪比	结构相似性指数	学习感知图像块相似度
Dr Johnson	29.78	0.897	0.275	29.82	0.906	0.263
游戏室	30.59	0.909	0.267	30.85	0.917	0.254
Mean	30.18	0.903	0.271	30.33	0.911	0.258