arXiv:2611.31918v1 [cs.CV] 24 Nov 2026

HiGS：用于快速可微分三维高斯溅射的分层架构

Dawid Paj.ak Martin Bisson Rodolfo Lima François Trudel
NVIDIA

摘要

*3D*高斯泼溅（*3DGS*）将一个屏幕瓦片同时作为分区单元、光栅化单元和并行工作单元。分区倾向于采用较大的瓦片，而光栅化则倾向于采用较小的瓦片；由于每个瓦片对应一个工作单元，最密的瓦片决定了帧时间。我们提出分层瓦片高斯泼溅（*HiGS*），这是一种用于渲染和重建的统一架构，为每种职能赋予各自的粒度：高斯在粗粒度的宏瓦片上进行分区，在其中的细粒度渲染瓦片上进行光栅化，并以有界的深度排序批次进行处理。密集区域会派发更多批次，而不是一个更长的工作单元，因此帧时间取决于视图中的工作量，而非其中最密的瓦片。同一批次也用于重建：合成过程提供独立计算每个批次梯度所需的透射率和颜色，使优化器的内存流量成为训练的主要开销。我们将优化器融合进反向传播，将其存储并直接更新的参数和矩估计减半，并将外观更新推迟到视图处理之外。与我们评估的所有已发表方法相比，*HiGS* 渲染场景的速度都更快；对于负载不均衡的视图，其优势最为显著；同时，它能以更快的速度和更少的内存完成重建，并保持相同的图像质量和精确的由前向后的 α 合成。

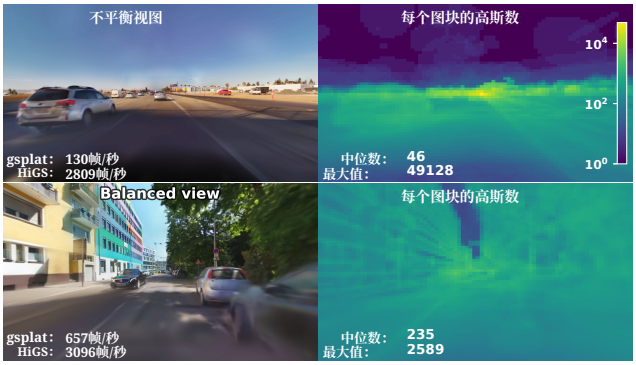


图 1. **每图块负载不均衡** 在两个驾驶视图上（1080p，16 × 16 图块），以及 gsplat 的吞吐量[52]以及 HiGS（第 3.1节）在各图块上的吞吐量。中位数根据非空图块计算。最密图块决定了 gsplat 的帧时间。

1. 引言

3D高斯泼溅 (3DGS) [21] 已成为实时逼真渲染捕获场景的主流方法，兼具神经辐射场的保真度 [36]以及一种显式、可微的表示形式，商用GPU可通过光栅化以高吞吐量处理它。重建过程会数万次运行光栅化器的前向和反向，因此光栅化成本决定了重建和显示的开销。

大多数 3DGS 变体都采用同一条流水线：将高斯投影到屏幕上，分配到其覆盖的图块中，按图块和深度进行全局排序，为每个图块生成一个高斯列表，然后为每个图块分配一个线程块进行光栅化。图块承担着三个相互冲突的角色：分区单元、光栅化单元和并行工作单元。分区倾向于使用较大的图块，因为这样可以减少（图块、高斯）配对的数量并进行排序。光栅化倾向于使用较小的图块，因为这样可以减少高斯有效支撑区域之外的像素–高斯计算。在任何尺寸下，少数密集图块都包含远多于其余图块的高斯，而为每个图块分配一个线程块会产生 尾部效应，从而主导帧时间（图 1）。单一图块尺寸无法同时满足这三个要求；在 4K 分辨率下，3DGS 每帧需要对数千万个配对执行全局基数排序，而且帧时间由最密图块而非视图中的工作量决定。

以往工作在未分离这三种角色的情况下降低了这些成本。它们减少配对、压缩模型并加速单个阶段 [6, 7, 9, 15, 16, 31, 33, 44, 46, 48], ，而未改变流水线的方法则以近似合成 [18, 23] 或定制硬件 [29, 51] 为代价。以比光栅化更粗的粒度进行排序 [3, 19, 27] 能缩小排序规模，但每个图块仍对应一个无界的高斯列表；重新调度图块 [12], ，并将最密图块的列表拆分到多个线程块中 [56]，虽然缩短了处理尾部，却保留了全局排序。这些补救措施各自改变了图块的一种或两种角色，却都没有将这种改变延伸到训练的反向传播中。

我们提出了 HiGS，这是一种面向通用 GPU 的统一架构，为每种角色设定了各自的粒度，并以一个工作单元同时负责渲染和重建。高斯会在粗粒度的 宏图块上进行分箱和深度排序，每个宏图块都是一个

细粒度 渲染瓦片的组，高斯将在这些渲染瓦片上进行光栅化；工作单元则是一个（宏瓦片、高斯批次）对：这是一个大小受限的深度排序批次，会被独立光栅化，然后按照深度顺序与其他批次进行合成。这种单一分解方式带来了三个结果。排序被拆分为相互独立的宏瓦片片段。每个批次，无论是否属于密集瓦片，都能装入快速本地存储，在其中仅加载一次，并由其包含的渲染瓦片共享。密集宏图块会按比例发出更多单元，因此不会有任何单元处理一个密集瓦片的完整高斯列表，从而消除了尾部效应的成因。

相同的单元负责重建，因此训练步骤的两个遍次都采用这一调度。合成会为每个批次保留其前方的透射率以及从该处开始的颜色，而批次在求导时也彼此独立，就像它们在渲染时一样。随后，优化器成为训练的主要负担；我们减少其内存流量，直到它同样能够根据视图所观察到的内容进行扩展：优化器被融合进反向传播中，直接更新低精度参数和矩估计，并推迟更新视图所剔除的高斯的外观。与标准基准场景上的近期优化光栅化器相比，HiGS 的渲染速度提升了 1.5× 至 5.6× 倍；在负载高度不均衡的驾驶视图上，速度提升了 6.8× 至 20.0× 倍。与我们评测的已发表方法中速度最快者相比，它重建场景的速度提升了 2.3× 至 3.0× 倍；在图像质量相同的情况下，其内存占用比内存效率最高的方法少 1.9× 。

我们的贡献包括该架构及其三个组成部分：

- 一种用于渲染和重建的**统一架构**，围绕一个单元——宏瓦片的有界深度批次——对分区、光栅化和并行工作进行分离；
- 基于这些单元的**渲染器**，采用分段深度排序、具有内联可见性的负载均衡批次光栅化，以及稳定的半精度（FP16）数据通路；
- 在相同批次上执行**反向传播**，每个批次都根据合成留下的边界状态独立进行微分；
- 与反向投影融合的**低带宽优化器**，直接更新采用随机舍入的 16 位状态，并延迟更新视图不需要的外观。

2. 相关工作

Kerbl 等人。 [21] 确立了基于图块的 3D 高斯斑点流水线，而 gsplat [52] 则通过广泛采用的开源库使其得到普及。此后的大多数工作都保留了渲染模型，并降低了该流水线的成本。

成对约简。许多配对是假阳性：高斯的边界框与某个图块重叠，但高斯实际并未延伸到该图块。StopThePop [44]引入了感知不透明度的边界，并精确测试椭圆会延伸到哪些图块。FlashGS [9], 、AdR-Gaussian [48], 和 Speedy-Splat [16]推导出了类似的边界，其中前者和后者还各自采用了精确测试。Faster-GS [15] 也采用了这些方法。HiGS 在分箱和光栅化期间测试椭圆重叠，但其成对约简主要来自在宏图块上进行分箱。

模型尺寸缩减。更少的高斯也意味着更少的配对。与我们的方法互补的工作通过缩减模型来提升性能：Speedy-Splat 和 RadSplat [39]，通过剪枝，Taming 3DGS [33]，通过预算式致密化，Mini-Splatting [8], 通过重要性加权采样，FastGS [45], 通过多视图一致致密化和剪枝，以及 LightGaus-sian [7], 、EAGLES [10], 、Compact-3DGS [28], 和 Com-pressed 3DGS [38]通过量化和码本方案。Kerbl等人 [22] 针对超大规模采集数据，在场景上构建细节层次结构，而非针对渲染工作构建层次结构。

训练加速。其中一些方法在训练期间发挥作用，而训练耗时主要由反向传播和优化器决定。3DGS 的反向传播会按像素累积梯度，并因原子更新而产生竞争；Taming 3DGS 通过对斑点并行处理来避免这一问题，而 DIST-WAR [6] 则先在寄存器中聚合一个线程束对同一高斯的更新，再为每个参数执行一次原子更新，从而减少竞争。LiteGS [30]在拥有某个图块的线程束内归约梯度，并以簇为单位剔除和更新按莫顿序排列的高斯。在优化器方面，Faster-GS 将逐斑点反向传播与其自有的 Adam 实现配对，并可将其融合进该传播过程，据报告，端到端训练速度最高可提升5× 。GS-Scale [26]仅将几何数据保留在 GPU 上；当高斯位于视锥体之外时，其他参数则存放在主机内存中，其 Adam 更新会延后执行。其他方法降低的是一次运行的工作量，而非单步成本：DashGaussian [2]调度渲染分辨率和高斯数量，SAD-GS [32]利用多尺度图像结构引导致密化以缩短收敛时间，而 Grendel [57] 则将训练分配到多块 GPU 上。这些方法与 HiGS 相辅相成；HiGS 重新设计了训练步骤本身。

渲染加速。 在前向传播中，GPU 实现逐阶段优化流水线，大多保留全局排序，并以屏幕瓦片作为工作单元。Flash GS 融合预处理内核，并在软件中对有依赖关系的高斯数据获取进行流水线处理；TC-GS [31]将 alpha 混合指数重新表述为 Tensor Core 矩阵乘法，在集成到 3DGS 后报告最高达 2.2× 的加速比。Splatshop [46] 在创建瓦片片段之前，按 32 位深度对可见斑点排序，从而缩短排序键。TileGS [47] 用每个图块内的深度分箱取代全局排序，并仅对选定分箱进行精确排序。为应对不均衡负载，StopThePop 均衡各高斯阶段的负载，而 AdR-Gaussian 采用一种损失函数进行训练，使每个像素的高斯数量趋于均衡。在光栅化器中，Balanced 3DGS [12]通过允许线程块在运行时认领图块、将图块缩小至四个像素，并将同一像素的高斯分散到一个线程束的多个线程上，来应对负载不均衡。

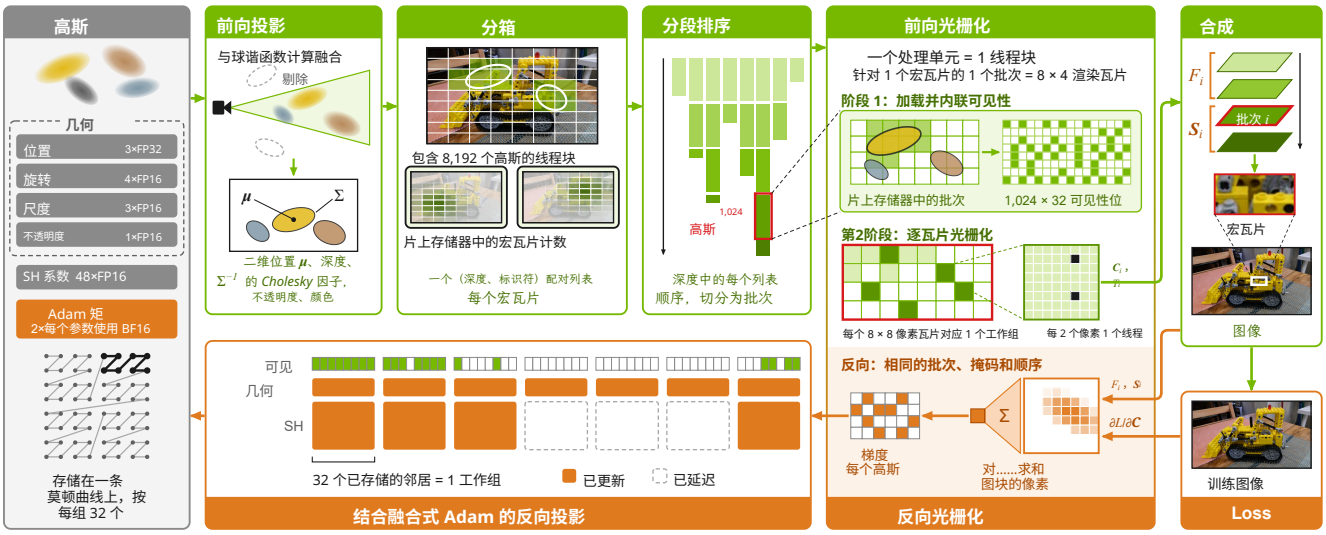


图 2. **HiGS流水线**：每帧包含的阶段（绿色）、训练步骤额外包含的阶段（橙色）以及高斯（灰色）。光栅化面板展示了两个遍次共用的一个处理单元：对于每个像素，它将批次 i 单独混合为颜色 $\mathbf{C}_i$ 和透射率 T_i。合成会为每个批次赋予其前方各批次的透射率 F_i，以及它自身和后方各批次共同贡献的颜色 $\mathbf{S}_i$。

CaT-GS [56] 将负载高于平均负载两倍的图块的深度排序列表划分为多个切片，由不同线程块分别进行光栅化，再按顺序合成；这一做法融入了每图块流水线及其全局排序。对于超高分辨率，SHARP-GS [3]采用随分辨率增大的图块尺寸进行分箱，通过每批次 32 个高斯对应的位掩码，针对更精细的图块进行剔除并完成光栅化，同时按 Morton 顺序存储高斯。GS-TG [19], 是一种也在 GPU 上进行评估的加速器设计，它对多个图块组成的组进行排序，并通过每个高斯对应的位掩码对原始图块进行光栅化；其他硬件方案 [27, 29, 51] 则不再使用商品级 GPU。在 HiGS 中，一个有界批次是所有列表的工作单元，适用于渲染以及重建的两个遍次。

其他方法。 其余工作会改变渲染模型本身，而 HiGS 保留了该模型。无排序高斯泼溅 [18] 放弃精确的前向到后向合成，改用可交换的加权和；StochasticSplats [23]则采用随机透明度。其他方法通过逐光线排序（StopThePop）、带限基元 [53], 以及结合混合透明度的透视校正求值 [14], 来提升保真度，或将模型扩展至次级光线 [37]和畸变相机 [50]。

3. 渲染架构

HiGS流水线（图2，实现细节见第 B 节）由多个层级的工作单元构成，并采用近期光栅化器中的融合投影与球谐函数（SH）计算 [9, 15], , 输出每个高斯的屏幕空间中心、深度、逆协方差因子、颜色和可见性。

3.1. 宏瓦片层次结构与工作分解

该层级结构建立在标准光栅化基元渲染图块之上，此处每个图块包含 8×8 个像素。HiGS 将 M=32个相邻渲染图块组合成一个宏瓦片，排列为 8×4 个图块，每个图块覆盖 64×32 个像素。分箱和排序为每个宏瓦片生成一个按深度排序的高斯列表（第3.2节），而可见性判断和混合则针对每个渲染图块进行（第3.3节）。在每个宏瓦片内，高斯数据只需加载到本地存储一次，即可供所有 M 个所属渲染图块重复使用，从而将数据读取开销分摊到细粒度输出上。

每个列表被拆分为 高斯批次，每批最多包含B_r=1个,024 个高斯，每个（宏瓦片，批次）对都构成一个独立的处理单元。无论列表是否稠密，这一上限对所有列表都成立，因为只有这样批次才能驻留在本地存储中。包含 n 个高斯的宏瓦片会生成 $\lceil n/B_r \rceil$ 个单元。在每个处理单元内，渲染图块由工作组从共享的 图块工作队列 $\mathcal{Q}$中动态认领，而每个工作组的 调度缓冲区 $\mathcal{R}$ 则吸收每个图块可见高斯数量的差异，使混合流水线以固定速率运行。因此，调度与密度成正比，而非与屏幕面积成正比：计算量随高斯数量变化；与每个图块分配一个线程组时最慢的图块会拖慢整帧不同，不会有单个渲染图块拖慢整帧（图3）。

这一层级结构的每一层都映射到 GPU 的一个层级（图 2）。处理单元是一个线程块，其批次驻留在片上存储器（即我们的本地存储）中；工作组由 P=32 个锁步执行的线程组成；每个线程则将其像素累加器保存在寄存器中。预分配的宏图块列表和批次部分缓冲区使各单元能够独立运行，存储需求随批次数量增长。

3.2. 分区与分段深度排序

分箱会为每个宏图块构建一个深度键控高斯列表。列表长度取决于数据，因此并行构建列表时，必须协调写入位置，同时避免通过全局原子操作进行串行化。Kerbl 等人 [21] 将此任务交给排序处理：复合键（图块，深度）会按图块对无序记录分组，并按深度排序，因此排序是全局的。

我们的分箱过程会在分箱期间生成每个宏瓦片的列表，因此排序时只需进行段内排序。高斯以 B_p=8,192 个为一批进行分区，每批由一个线程组协同处理。该线程组在本地存储中维护一个私有直方图，其中帧中的每个宏瓦片对应一个计数器，并使用本地原子操作而非全局原子操作来累积重叠计数。由于帧的宏瓦片数量适中，直方图可以容纳：1080p 时约有 1,000 个条目，4K 时约有 4,000 个条目；而渲染瓦片直方图则包含 30,000 至 130,000 个条目，比本地存储容量高出一个数量级，因此只能采用全局原子操作路径。分层前缀扫描会将批次局部计数转换为写入偏移量，随后填充阶段会再次遍历各批次，并将（深度、高斯标识符）配对直接写入每个宏瓦片的范围。

遵循 Ac- cuTile [16]更粗一级的做法，通过逐行保守扫描不透明度阈值椭圆，将高斯与宏图块配对。因此，HiGS 在 8×8 个图块上进行光栅化，但在 Mip-NeRF 360 上，1080p 分辨率下平均仅有 1.51M 个配对，4K 分辨率下平均仅有 2.21M 个配对；在 16×16 个图块上，配对数也比 gsplat 的不透明度感知包围盒少 $4.3\times$，而后者已经形成 $2.5\times$，且比 8×8 少 $3.1\times$。宏图块分箱贡献了超过一半的削减，其余部分则来自椭圆测试。

随后，每个宏图块都成为分段排序中的一个段，而且这些段在设计上彼此独立。段本身已标识其所属的宏图块，因此键只需使用 32 位深度，而非 gsplat 的 64 位复合键。移除图块位后，基数排序所需处理的键更短，记录大小也从 12 字节缩减至 8 字节。Splatshop [46] 在全局排序中缩短键的长度，而在这里，图块维度则从键中消失。随着分辨率提高，渲染图块配对数量会增加，但段的数量仍有上限，因为更多宏图块会将高斯分配到更多、更小的段中。随着段数量增加，通用分段排序仍会变慢。因此，我们采用由六个阶段组成的流水线进行排序，整个过程无需主机同步：先对每类段大小分别执行块基数排序，共四次级联 [35], ；对于超出最大类别的段，则先对固定大小的运行进行排序，再使用 merge path 将其合并 [11]。这样一来，从 1080p 到 4K，排序所需时间都保持在相同范围内（第 5.3 节）。

3.3. 批量光栅化与合成

3DGS 在光栅化开始前，会为每个图块显式生成高斯列表。我们的光栅化器将这一阶段替换为与高斯负载同步进行的可见性判定，额外开销很低。每个处理单元由 W 个工作线程组成 P，并分两个阶段处理其（宏瓦片、批次）对，随后由所有单元共享一个合成过程（图2）。

阶段 1：负载与内联可见性。该单元以每批 P 个高斯、每个线程一个的方式，通过协作将其批次加载到本地存储中。加载期间，会测试每个高斯的不透明度阈值椭圆是否与所有 M 个渲染图块重叠，从而生成逐高斯瓦片掩码。GS-TG [19] 会在光栅化之前生成这些掩码；在 GPU 上，这会使掩码生成与排序串行执行。SHARP-GS [3] 也会为每批 32 个高斯生成一个掩码；在协作加载期间计算这些掩码，可避免这次内存往返。加载过程会确定每个高斯与哪些图块重叠，但混合所需的信息恰好相反：每个图块能看到哪些高斯。对每个小批次的 $P\times M$ 位矩阵进行转置，即可实现两者之间的转换：随后，每个渲染图块都会为每个小批次持有一个 P位掩码，而将这些掩码串接起来，便得到该图块在整个批次中的可见性。

第2阶段：逐瓦片光栅化。 工作组从 $\mathcal{Q}$认领活动渲染瓦片，因此只要还有瓦片，就不会有工作组闲置。对于认领的瓦片，工作组遍历其掩码，并从本地存储中提取可见高斯。调度缓冲区 $\mathcal{R}$ 累积这些高斯的索引，并通过展开式混合核心按 B_d=64 个一批排空它们，从而将变速的掩码迭代与定速的混合运算解耦。每次排空后，一致投票会测试瓦片中的每个像素是否都已达到透射率饱和；如果是，工作组就认领下一个瓦片。**合成**。 宏瓦片的列表可能跨越多个批次，并由彼此独立的单元处理，因此逐瓦片结果是部分结果：批次 i 为每个像素生成局部颜色$\mathbf{C}_i$ 和局部透射率 T_i，仅与其自身的高斯进行混合。正如 Balanced 3DGS 和 CaT-GS 所采用的那样，前向到后的透明度混合在连续深度范围 [43], 上具有结合性，因此，若 $F_i=\prod_{h<i}T_h$ 表示批次 i前方的透射率，则像素颜色为

$$\mathbf{C}=\sum_i F_i\,\mathbf{C}_i, \tag{1}$$

加上由最终透射率加权的背景。每个单元仅为接收到高斯数据的图块写入其部分结果，这些图块会在每个批次的活动瓦片掩码中标记；而合成过程则按照深度顺序遍历每个渲染图块的各个批次，累加公式（1）。该过程在单个图块的各批次之间按顺序执行，但并不会形成瓶颈：它读取紧凑的部分结果流，在图块达到饱和后跳过剩余批次，并且在 1080p 分辨率下耗时仅为 0.01 到 0.03 毫秒。因此，每个像素都会按照 3DGS 的深度顺序并依据其 alpha 混合递推，从前到后完成合成；这正是我们称 HiGS 的合成过程为精确的含义。

3.4. 数值表示与存储顺序

光栅化器以 FP16 存储并计算投影属性。这样做是出于内存方面的考虑，而非算术运算：

在我们面向的 GPU 上，打包后的 FP16 与单精度（FP32）的数学运算吞吐量相同，但 16 位属性可将内存和本地存储占用减半，因此处理单元的负载速度提高一倍，可同时驻留的高斯数量增加一倍，并且每个线程所需携带的累加器状态减半。在逐像素马氏距离 $q = (\mathbf{p}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{p}-\boldsymbol{\mu})$中存在两个隐患，其中像素为 $\mathbf{p}$，投影中心为$\boldsymbol{\mu}$，屏幕空间协方差为 Σ。HiGS 通过数据表示而非逐像素处理来解决这两个问题。

当 Σ^{-1} 以圆锥系数的形式存储时，其二次型包含一个正负号混合的交叉项，因此会产生抵消。HiGS 改为存储 $\Sigma^{-1} = LL^\top$ 的 Cholesky因子，即一个下三角 L，其元素为 l_0、 l_1和 l_2，这样对于偏移量（dx，dy），$= \boldsymbol{\mu} - \mathbf{p}$

$$q = (l_0\,dx + l_1\,dy)^2 + (l_2\,dy)^2, \tag{2}$$

即使发生舍入，平方和也始终非负。坐标若将其屏幕空间绝对幅值带入这些乘积，就会导致溢出；一个单元仅相对于其宏瓦片中心，以渲染瓦片单位为基准，将位置缩窄为 FP16。TC-GS [31] 也识别出同一风险，并按图块平移坐标；而此处只需在负载时平移一次，即可用于该宏瓦片的所有渲染瓦片。对于渲染，还可以通过 YCoCg 去相关 [34] 和内联解码的非均匀量化，将球谐函数系数压缩至每个高斯 32 字节，仅为其 FP32 大小的六分之一 [42]。

存储顺序也会影响内存流量。投影仅为视图未剔除的高斯读取 SH 系数，分箱也仅为这些高斯读取投影属性。对高斯均值进行 Morton排序（SHARP-GS 和 LiteGS 也采用这种方式），会将被剔除的高斯分组成连续区段。它们的数据会以完整的内存扇区为单位被跳过，而不是与可见邻居一起读取，从而缩短帧（第 5.3节）。该顺序独立于每帧的深度排序，并且相同的局部性也会传递给优化器（第4.4节）。

4. 重建

一个训练步骤使用第 3节的流水线渲染一个视图，通过该流水线对图像损失求导，并更新参- 数。我们保留 3DGS 在 gsplat 中的目标函数和致密化策略。独立的处理单元负责反向传播，之后优化器的内存流量成为该步骤的主要开销；我们持续降低这一开销，直到外观更新的规模取决于视图所看到的内容，而不是场景大小。

4.1. 独立批次微分

高斯不透明度在像素处的梯度 α 取决于其前方的透射率以及其后方合成的颜色。3DGS 通过将每个图块的列表从后向前遍历，并将透射率除以 $1-\alpha$来恢复它们；StopThe- Pop [44] 则从前向后遍历，并从最终结果中减去颜色，而 Taming 3DGS [33] 和 Faster-GS 则从每 32 个高斯存储一次的逐像素状态开始从前向后遍历。由于一个图块的列表跨越了独立渲染的批次，没有一种方法能直接适用于 HiGS，因为在这种密度下按图块存储状态会牺牲共享负载所带来的内存节省。

我们为每个批次存储一次此状态。训练前向过程使用 FP32 累积颜色，同时将混合权重和透射率保存在 FP16 中，而第 3.3 节的合成过程会完成每个批次的边界状态。其从前到后的遍历已经计算出批次 F_i 前方的透射率 i；再进行一次从后到前的遍历，将每个颜色部分结果替换为后缀部分

$$\mathbf{S}_i = \sum_{h \geq i} F_h\,\mathbf{C}_h, \tag{3}$$

即从批次 i 起合成的颜色。两者都会原位覆盖部分结果，因此边界状态无需占用额外存储；每个像素、每个批次仅需 14 字节的部分结果缓冲区。随后，反向处理单元无需其他批次，即可根据 F_i、$\mathbf{S}_i$ 和该批次，对其（宏瓦片、批次）对进行微分。

在单元内部，会重新计算前向传播，而不是对其求逆。该单元将同一批次载入局部存储，复用前向传播保存的可见性掩码，并由每个工作组使用前向传播的算术运算、排空边界和饱和度测试，使其图块从前到后进行混合，初始局部透射率为 1。这样便能精确复现每个高斯 j 的舍入后混合权重 $\widehat{w}_j = \alpha_j t_j$，其中 α_j 是其在该像素处的不透明度，t_j 是其前方的局部透射率；在接近饱和时，通过 FP16 透射率相除无法做到这一点。令$\mathbf{c}j$ 为高斯 j 的颜色，$\mathbf{g} = \partial\mathcal{L}/\partial\mathbf{C}$为像素处损失 $\mathcal{L}$ 的梯度，$\mathbf{g}_i = F_i\mathbf{g}$、背景 $\mathbf{b}$ 和最终透射率 T_{end}，该组将标量残差 $r = \mathbf{g}^\top(\mathbf{S}_i + T_{\text{end}}\mathbf{b})$ 初始化为批次处及其后的损失加权颜色，并依次对每个高斯计算

$$r \leftarrow r - \widehat{w}_j\,\mathbf{g}_i^\top\mathbf{c}_j, \tag{4}$$

$$\frac{\partial\mathcal{L}}{\partial\alpha_j} = t_j\,\mathbf{g}_i^\top\mathbf{c}_j - \frac{r}{1-\alpha_j}, \qquad \frac{\partial\mathcal{L}}{\partial\mathbf{c}_j} = \widehat{w}_j\,\mathbf{g}_i, \tag{5}$$

这样，r 始终保存当前高斯后方的贡献。

从 α_j 到投影中心、Cholesky 因子和不透明度的其余链式法则，在每个像素上计算，并在工作组内归约。因此，每个高斯和图块执行五次双字原子加法，以更新其几何和颜色梯度的九个累加器，且无需使用逐像素梯度缓冲区。

4.2. 与投影融合的优化器

光栅化加速后，Adam 的带宽开销便显现出来：每一步都要读取和写入所有参数及其两个矩，采用 FP32 和 3 阶球谐函数时，每个高斯的状态占 708 字节。在我们的光栅化器中，PyTorch 的 Adam 所需时间甚至超过了前向和反向传播

两者的总耗时（第 5.4 节），而下文讨论的三种措施可减少其流量。

第一项措施将梯度移出内存。与 Faster-GS 一样，更新在反向传播中进行：反向投影为每个高斯分配一个线程，由该线程计算投影和 SH 计算的导数，因此会将高斯几何结构的梯度以及每个 SH 梯度的两个因子都保存在寄存器中。线程在此处应用 Adam，工作组中的线程协同更新 SH 系数，并且每个参数和矩估计只写入一次。唯一会进入内存的梯度是光栅化器累加器，每个高斯占 40 字节；由于多个图块会对同一个高斯产生贡献，因此需要这些累加器。

4.3. 低精度参数和矩

融合消除了梯度传输开销，但并未减少状态占用。第二项措施将其减半：位置仍采用 FP32，其他所有参数（旋转、对数尺度、不透明度对数几率和 SH 系数）均以 FP16 存储，Adam 的两个矩估计则以 bfloat16（BF16）存储，[20], 每个高斯占 360 字节，而非 708 字节。这些数组是优化器自身的状态，并非 FP32 参数的舍入副本，因此渲染器读取的内容与优化器写入的内容完全一致。投影、其导数以及 Adam 运算仍采用 FP32；只有存储时才降低精度。

将存储精度收窄并采用就近舍入，会丢弃所有小于末位半个单位的更新；而斑点优化中较小的学习率使这种情况十分常见。我们像 [13], Zamirai 等人 [54] 在 BF16 训练权重时所做的那样采用随机舍入，也对矩估计进行舍入：存储值是精确结果的两个可表示邻值之一，按与精确结果的接近程度设定选取概率，因此存储值的期望等于精确值。采用就近舍入时，重建质量会明显下降；采用随机舍入时，重建质量则与 FP32 存储相当（第5.4节）。

4.4. 与可见性成比例的外观更新

剩下的大多是外观：球谐函数系数占一个高斯 59 个标量中的 48 个，但在 Mip-NeRF 360 的花园场景中，训练视图平均会剔除 66% 的高斯，它们的梯度为零。正如 Taming 3DGS 所建议的那样跳过这些高斯，或像 Faster-GS 那样对其进行评估，以及像 LiteGS 那样跳过整个簇，并不等同于更新它们，因为即使梯度消失，Adam的一阶矩仍会继续推动参数变化；Ding等人 [4] 将此分析为隐式更新。然而，零梯度步只取决于存储的矩估计和步数索引。我们像稀疏在线学习在正则化中所做的那样，[5, 25], 延迟执行这些步：被剔除的系数保持不变，直到某个视图需要它，此时再以闭式解补上错过的所有步。

令 (m, v) 表示该系数 θ 在其上一次更新于第 a 步后的矩估计。经过 k 个梯度为零的步后，采用衰减率 β_1 和 β_2 的 Adam 会将它们衰减为 $(\beta_1^k m,\ \beta_2^k v)$，并且使用学习率 η 和 $\rho = \beta_1/\sqrt{\beta_2}$，我们将 θ 移动

$$\Delta\theta = -\frac{\eta\, m\, \Phi}{\sqrt{v} + \varepsilon_A\, \Phi/\Psi}, \tag{6}$$

$$\Phi = \sum_{j=1}^{k} \rho^j \frac{\sqrt{1-\beta_2^{a+j}}}{1-\beta_1^{a+j}}, \qquad \Psi = \sum_{j=1}^{k} \frac{\beta_1^j}{1-\beta_1^{a+j}},$$

其中 Φ和 Ψ 分别是在 Adam 的 ε_A 相较于$\sqrt{v}$ 可以忽略和占主导的情况下，偏差校正步因子的总和。对于球谐函数系数的固定学习率，式（6）在这两种情况下以及 k=1时，都与 Adam 的位移相等；而在两者之间，从第 100 步起，差异始终小于 0.05%。每个总和都来自每次运行时构建的一张表中的两个条目，因此无需限制滞后：只保留最后一次更新的步数，而任何视图都不需要的系数既不会被读取或写入，也不会被计数。前向传播读取滞后的系数时，会在寄存器中应用式（6）；融合优化器则使用当前步数应用该式，并将结果存储一次。GS-Scale [26]也会延迟更新其主机端参数的 Adam 更新，对 ε_A=0 的恢复精确无误，并使用一个四位计数器，确保至少每 16 步更新一次；而花园 中的滞后中位数为 2 步，第 99 百分位数为 38 步。

我们按组记录一次最后更新，每组包含 P=32 连续的高斯，由优化器的一个工作组负责更新。没有可见高斯的组会跳过其球谐函数更新；这是工作组统一作出的决定，因此该组中每个高斯的 288 字节系数和矩估计都保持不变；其他组则更新其所有成员。几何信息每一步都会更新，因为它决定可见性。为每个高斯单独记录并不能多跳过多少更新，因为存储采用 Morton 顺序，已能缩短渲染时间（第 3.4节）：在 花园场景中，即使致密化结束后，98% 的被剔除高斯仍位于完全剔除的组中。而在 gsplat 致密化后保留的顺序中（新高斯追加在末尾），这一比例为 11%。Faster-GS 每 5,000 次迭代会重新排列高斯，以减少缓存未命中和分歧控制流；LiteGS 则在致密化步骤后重新排列。在这里，顺序决定了有多少外观计算可以延后；每个致密化步骤都会通过一次收集操作恢复顺序，该操作涵盖所有逐高斯数组，并将每个副本或一对拆分得到的子项写入其父项所在的位置。这样，优化器每个高斯、每一步只需移动 375 字节；而稠密 FP32 更新需要移动 1,416 字节，因为它会完整读写一次整个状态（详细推导见第 C 节）。

5. 实验

5.1. 设置

我们使用一台配备 NVIDIA RTX PRO 6000 Blackwell GPU 和 AMD Ryzen 7 7800X3D CPU 的 PC 评估我们的方法。

数据集	方法	时间（秒）	内存（GiB）	峰值信噪比	结构相似性指数	学习感知图像块相似度
Mip-NeRF 360	3DGS	650.9		3.76	27.81 .8245	.1693
	gsplat	309.4		3.87	27.82 .8254	.1671
	Faster-GS	130.5		5.73	27.81 .8253	.1676
	HiGS	49.3		2.02	27.81 .8254	.1666
Tanks and Temples	3DGS	399.8		2.23	23.39 .8386	.1409
	gsplat	213.8		2.30	23.41 .8389	.1402
	Faster-GS	85.4		3.41	23.41 .8389	.1405
	HiGS	36.5		1.19	23.40 .8387	.1398
Deep 混合	3DGS	765.2		3.73	29.83 .9076	.1650
	gsplat	293.8		3.86	29.82 .9075	.1644
	Faster-GS	118.6		5.74	29.84 .9076	.1645
	HiGS	40.1		2.01	29.80 .9072	.1632

表1. **重建**：各数据集场景的平均值，以及十次运行的平均值（时间和内存为三次运行）。

我们将方法与有公开实现的方法进行比较。**重建**。 我们使用同一训练框架，让每种方法采用相同的初始化流程、训练视图序列、损失、调度方案和 3DGS稠密化策略。渲染器、反向传播、优化器和存储顺序则各不相同。我们将方法与 3DGS、gsplat 以及采用其融合优化器的 Faster-GS 进行比较。所有方法的不透明度学习率均为 0.025。每种方法都在 13 个常用于评估 3DGS 的场景上训练 30,000 步，分辨率采用各数据集的标准分辨率：Mip-NeRF 的九个场景 360 [1], Tanks and Temples 的两个场景 [24], 以及 Deep Blending 的两个场景 [17]。我们报告训练循环的时间（不包括启动阶段解码训练图像并将其缓存到 GPU内存所需的时间）、不含该缓存时的峰值 GPU内存，以及测试集上的峰值信噪比（单位：dB）、结构相似性指数 [49],和学习感知图像块相似度 [55]（采用 AlexNet 特征）。

渲染。 我们使用每个测试相机，以 1080p 和 4K 分辨率渲染七个 Mip-NeRF 360 场景公开发布的 3DGS 训练模型（球谐函数阶数为 3，高斯数量为 120 万至 610 万），并报告每个场景在所有相机计时帧上的中位数；每个相机取 10 帧预热后再计时 100 帧。我们使用 CUDA 事件对渲染一帧所需的所有 GPU 内核进行计时。为测试不均衡负载，我们还以 1080p 分辨率加入 120 个驾驶视图，每秒一帧，来自 PhysicalAI Autonomous Vehicles数据集的六段片段，这些片段由 NuRec 重建 [40], , 每个场景包含一个由 250 万个高斯构成的模型。HiGS 采用默认配置：8×8 渲染瓦片、Morton序存储和 32 字节球谐函数。每个先前的光栅化器均采用其公开版本的默认配置，使用 16×16 图块（Faster-GS 使用16×12），深度排序采用与我们的方法相同的图块级排序，且不进行抗锯齿。

5.2. 重建

在这三个数据集上，HiGS 完成重建的速度比次快的方法快 2。3× 到 3。0× 倍，比 gsplat 快 5。9× 到 7。3× 倍（表 1）。它的内存用量也比内存效率最高的替代方法少 1。9×；这一比例接近优化器状态所占的比例，优化器会直接更新优化器状态，无需进行 FP32 拷贝（第 4.3 节）。数据集平均质量

	吞吐量（每秒帧数）		HiGS 加速比	
方法	1080p	4K	1080p	4K
HiGS	2756	1533	—	—
SHARP-GS	1250	1055	2.2×	1.5×
Faster-GS	949	578	2.9×	2.7×
FlashGS	893	670	3.1×	2.3×
TC-GS	765	499	3.6×	3.1×
Speedy-Splat	643	385	4.3×	4.0×
StopThePop	573	293	4.8×	5.2×
gsplat	541	275	5.1×	5.6×
3DGS	286	102	9.6×	15.0×

表 2. **渲染**吞吐量，即 1000 除以每个场景帧时间中位数的平均值，单位为毫秒。

相当：各方法间的峰值信噪比差异小于 HiGS 多次运行间的波动。表 1中的 HiGS 是唯一一种光栅化器在训练步骤的两个遍次中，工作量都与高斯密度成正比的方法，因此重建也获得了渲染器带来的加速。其余优势源自第 4 节：其他方法会在每一步更新每个高斯的外观状态，而没有融合优化器的方法还会移动完整的参数梯度。Faster-GS 是唯一使用融合优化器的基线，因此速度次之；第 5.4 节在 花园 上测量了这两部分（各阶段的细分结果见第 D 节）。

5.3. 渲染

HiGS 在所有场景的两种分辨率下都是最快的光栅化器（表 2）。它的领先幅度会随着方法对每图块流水线的改动程度而缩小：相比仅收紧 3DGS 图块边界的 gsplat、Speedy-Splat 和 StopThePop，领先 4。0× 至 5。6×；相比重构其阶段的 FlashGS、Faster-GS 和 TC-GS，领先 2。3× 至 3。6×；而相较于唯一一个分箱粒度比光栅化粒度更粗的基线 SHARP-GS，领先幅度最小。不过，随着负载增加，这种差距优势并不能维持：在 花园场景中，当预算为 90M 个高斯时，SHARP-GS 的表现落后于 FlashGS，而 HiGS 分别领先 3。4× （1080p）和 3。3×（4K）。

在 Mip-NeRF 360 上，HiGS 的单帧平均耗时为 0.36 ms（1080p），其中光栅化和合成占 38% 至 52%，分箱与排序占 25% 至 43%，投影与球谐函数计算占 9% 至 32%，且这些耗时会随着高斯数量增加。深度排序本身在两种分辨率下均耗时 0.03 至 0.06 ms；相比之下，gsplat 的全局基数排序在 4K 下耗时 0.44 至 1.10 ms，而 HiGS 在 4K 下比针对相同段的通用分段排序快 4.6× 至 6.0× 倍。对模型进行一次性的 Morton 排序置换后，1080p 下的帧时间降低 24%，4K 下则降低 18%；即使采用预训练模型的存储顺序，HiGS 仍比 gsplat 快 3.9× 倍，并且快 4.6× 倍。球谐系数打包可使帧时间降低 11% 和 7%；即使不进行打包，HiGS 仍然领先表 2中的所有光栅化器：比 SHARP-GS 快 2.0×倍和 1.4×倍。

图 3 展示了每个图块使用一个单元在不均衡负载下的代价。从最均衡的驾驶视图到最不均衡的视图，瓦片–高斯对增加了 1。

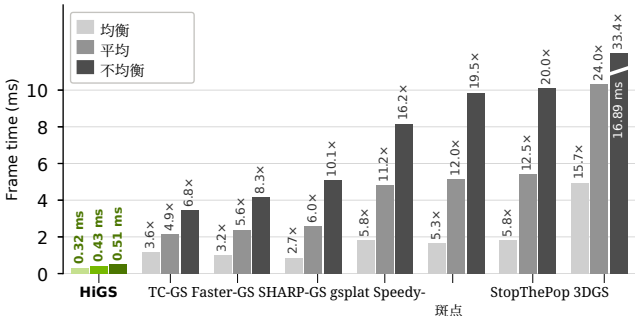


图3. **负载不均衡情况下的帧时间** 在120个驾驶视图上：分别统计最均衡的30个视图、所有视图以及最不均衡的30个视图的平均值；视图按最密集图块与中位数已占用图块的比值排序（图 1）。标签给出 HiGS 的帧时间及其相对于各光栅化器的加速比；被截短的柱形标示其帧时间。未能完成基准测试的 FlashGS 未列出。

配置	时间（s）	内存（GiB）	峰值信噪比（dB）
独立优化器，FP32	194.4	4.14 ±.02	27.83 ±.03
+ 空间有序存储	176.8	3.99 ±.01	27.82 ±.04
+ BF16 矩 ，至最近值	136.7	3.20 ±.01	27.68 ±.02
+随机矩估计舍入	137.0	3.18 ±.01	27.82 ±.05
+ 融合优化器	100.6	3.38 ±.01	27.83 ±.03
+ FP16 参数，舍入到最近值	82.0	2.69 ±.01	27.77 ±.06
+ 随机参数舍入	82.9	2.72 ±.01	27.81 ±.04
+延迟外观（HiGS）	61.5	2.72 ±.01	27.82 ±.05
HiGS，无随机舍入	60.1	2.65 ±.01	27.45 ±.03
HiGS，存储顺序未定	91.2	2.72 ±.01	27.84 ±.05

表3。 **消融实验** 重建流水线在 花园 场景上的表现，十次运行的平均值和标准差。每一行在上一行的基础上增加第 4 节中的一个要素，从搭配独立 FP32 Adam 和 gsplat 致密化的 HiGS 光栅化器开始。最后两行分别移除了 HiGS 中的一个要素。

在最不均衡的视图中，HiGS 的帧时间增加了 1。5×，而七种基线方法的帧时间增加了 3。6×至 5。0× 。9×在所有视图中，HiGS 的帧时间与视图中的工作量相符：对配对数量进行线性拟合可以解释其 90% 的方差，而对最密瓦片的高斯数量进行线性拟合可以解释 53% 的方差。基线方法的帧时间则取决于单个最密瓦片，该瓦片的高斯数量可以解释其 87% 至 98% 的方差，而配对数量可以解释 30% 至 40% 的方差。更粗粒度分箱不会改变这一点：表 2 中与其最接近的是 SHARP-GS，它与最密瓦片的关联最强，因为其每个瓦片始终保持一个单元。重建通过设计继承了这种均衡性，因为相同的单元承载训练步骤的两个遍次（第 4.1 节）。

固定模型后，HiGS 与 gsplat 的结果一致，峰值信噪比为 67.2 dB；使用打包球谐时为 55.6 dB。合成过程完全精确，因此差异来自 FP16 算术、每处理 B_d 个高斯进行一次饱和度测试，以及将不透明度上限设为 1 而非 0.99。与测试集相比，HiGS 和六种基线方法的峰值信噪比差异不超过 0.03 dB，结构相似性指数和学习感知图像块相似度的差异均不超过 0.001（更多详情见第 D 节）。

5.4. 消融实验

表 3的首行仅在光栅化器方面与 gsplat 不同：它训练 *garden* 耗时 194.4 秒，而 gsplat 耗时 404.8 秒，且优化器占据了其循环的 59%。接着，五项改动将训练时间缩短至 61.5 秒，其中两项恢复了质量。与上一行相比，我们的致密化方法保持存储在空间上有序（第 4.4 节），可减少 9% 的训练时间；单独使用带 BF16 矩估计的 Adam 可减少 23%；融合优化器可减少 27%，但峰值内存增加 0.20 GiB；使用 FP16 参数可减少 18%；最后，延后剔除组的外观更新可再减少 26%。总计，训练速度提升 3。2× 倍，内存减少 34%，峰值信噪比与首行相差 0.01 dB。采用最接近值舍入时，矩估计导致 0.14 dB 的损失，参数导致 0.06 dB 的损失；随机舍入可将损失恢复至仅相差各次运行之间的差异，而不采用该方法的 HiGS 会损失 0.37 dB，超过前两项损失之和。不使用空间有序存储时，HiGS 耗时会增加 48%；相比之下，Faster-GS 报告称，训练过程中不进行周期性重排序时，耗时仅增加 2% 至 6%。在这里，有序存储的作用不止于改善内存局部性：新增耗时的 90% 来自反向传播，此时当仅有少数组被完全剔除后，融合优化器便抵消了延后处理带来的收益（第 4.4 节）；其余部分则来自前向传播（更多结果见第 D 节）。

6. 结论

HiGS 基于一个观察：分区、光栅化和并行工作无需共用同一个图块。为每项工作设定不同的粒度后，排序便缩减为小型独立片段，数据只需加载一次即可共享，也不会再由某个单一密集瓦片限制整帧性能；这些都是同一种分解方式的结果，而非彼此独立的优化。重建沿用相同的单元及其背后的原则：只在视图需要之处投入计算，直至优化器的内存流量亦是如此。与我们评估的所有已发表方法相比，该方法的渲染和重建速度更快，重建所需内存更少，同时实现精确合成并保持相同的图像质量。

这种并行性是以内存和带宽为代价换来的。每个批次都会为自己的输出预留空间并写入输出，而且即使不透明前景将其遮挡，也仍会进行光栅化。因此，在所有高斯都覆盖整帧的退化场景中，这些开销会随着每个隐藏层而增长。捕获场景远没有这么极端，而在我们的所有测试中，均衡调度所带来的收益都超过了合成所丢弃的工作，尤其是在负载极不均衡的视图上（图3）。由于一个单元只需要自己的批次及其图块的边界状态，且外观参数在不可见时不会被改动，我们认为，能够超出单个 GPU 内存容量的场景——分布在多个 GPU 上，或部分存放在核外——将是顺理成章的下一步。

参考文献

[1] 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。载于 IEEE/CVF计算机视觉与模式识别会议（CVPR）论文集，2022。7[2] Youyu Chen、JunJun Jiang、Kui Jiang、Xiao Tang、Zhihao Li、Xianming Liu 和 Yinyu Nie。DashGaussian：在 200 秒内优化 3D高斯泼溅。载于 IEEE/CVF计算机视觉与模式识别会议（CVPR）论文集，第 11146–11155 页，2025。2[3] JunRan Ding、WeiHang Liu、YuKe Li、YiZhou Wang、An-Tong Li、QiHan Ding、Xin Lou 和 JingYi Yu。SHARP-GS：面向超高分辨率 3D高斯泼溅的可扩展高保真加速渲染流水线。载于 SIGGRAPH 2026 会议论文集，美国纽约，2026。计算机协会。1、3、4[4] Renjie Ding、Yaonan Wang、Min Liu、Jialin Zhu、Jiazheng Wang、Jiahao Zhao、Wenting Shen、Feixiang He 和 Xiang Chen。3DGS 优化解耦的一步。载于国际学习表征会议（ICLR）论文集，第 6858–6892 页，2026。6[5] John Duchi 和 Yoram Singer。使用前向后向分裂进行高效在线与批次学习。《机器学习研究杂志》，10(99):2899–2934，2009。6[6] Sankeerth Durvasula、Adrian Zhao、Fan Chen、Ruofan Liang、Pawan Kumar Sanjaya 和 Nandita Vijaykumar。DIST-WAR：基于光栅的渲染管线上的快速可微渲染。arXiv 预印本 arXiv:2401.05345，2024。1、2[7] Zhiwen Fan、Kevin Wang、Kairun Wen、Zehao Zhu、Dejia Xu 和 Zhangyang Wang。LightGaussian：无界 3D 高斯压缩，压缩率达 15 倍且每秒帧数提升 200+ 。载于神经信息处理系统进展（NeurIPS），2024。1、2[8] Guangchi Fang 和 Bing Wang。Mini-splatting：用数量受限的高斯表示场景。载于欧洲计算机视觉会议（ECCV），2024。2[9] Guofeng Feng、Siyan Chen、Rong Fu、Zimu Liao、Yi Wang、Tao Liu、Boni Hu、Linning Xu、Zhilin Pei、Hengjie Li、Xiuhong Li、Ninghui Sun、Xingcheng Zhang 和 Bo Dai。FlashGS：面向大规模高分辨率渲染的高效 3D 高斯泼溅。载于 IEEE/CVF计算机视觉与模式识别会议（CVPR）论文集，第 26652–26662 页，2025。1、2、3[10] Sharath Girish、Kamal Gupta 和 Abhinav Shrivastava。EAGLES：采用轻量级编码的高效加速 3D 高斯。载于欧洲计算机视觉会议（ECCV），2024。2[11] Oded Green、Robert McColl 和 David A. Bader。GPU 合并路径：一种 GPU合并算法。载于第 26 届 ACM国际超级计算会议论文集，第 331–340 页。ACM，2012。4、12[12] Hao Gui、Lin Hu、Rui Chen、Mingxiao Huang、Yuxin Yin、Jin Yang、Yong Wu、Chen Liu、Zhongxu Sun、Xueyang Zhang 和 Kun Z han。Balanced 3DGS：采用细粒度分块的逐高斯并行渲染。arXiv 预印本 arXiv:2412.17378，2024。1、2[13] Suyog Gupta、Ankur Agrawal、Kailash Gopalakrishnan 和 Pritish Narayanan。有限数值精度下的深度学习。载于第 32 届国际机器学习会议论文集，第 1737–1746 页。PMLR，2015。6[14] Florian Hahlbohm、Fabian Friederichs、Tim Weyrich、Linus Franke、Moritz Kappel、Susana Castillo、Marc Stamminger、Martin Eisemann 和 Marcus Magnor。利用混合透明度实现高效的透视校正 3D高斯泼溅。Computer Graphics Forum，44(2)，2025。3[15] Florian Hahlbohm、Linus Franke、Martin Eisemann 和 Marcus Magnor。Faster-GS：分析并改进高斯泼溅优化。载于 IEEE/CVF计算机视觉与模式识别会议论文集，第 18946–18957 页，2026。1、2、3[16] Alex Hanson、Allen Tu、Geng Lin、Vasu Singla、Matthias Zwicker 和 Tom Goldstein。Speedy-splat：利用稀疏像素和稀疏基元实现快速 3D高斯泼溅。载于计算机视觉与模式识别会议（CVPR）论文集，第 21537–21546 页，2025。1、2、4、16[17] Peter Hedman、Julien Philip、True Price、Jan-Michael Frahm、George Drettakis 和 Gabriel Brostow。面向自由视点基于图像的渲染的深度混合。ACM Transactions on Graphics，37(6)，2018。7[18] Qiqi Hou、Randall Rauwendaal、Zifeng Li、Hoang Le、Farzad Farhadzadeh、Fatih Porikli、Alexei Bourd 和 Amir Said。通过加权和渲染实现无排序高斯泼溅。载于国际学习表征会议（ICLR），2025。1、3[19] Joongho Jo 和 Jongsun Park。GS-TG：采用图块分组的 3D高斯泼溅加速器，在保持光栅化效率的同时减少冗余排序。载于 2025 年第 62 届 ACM/IEEE 设计自动化会议（DAC），第 1 7 页，2025 1、3、4 – 。[20] Dhiraj Kalamkar、Dheevatsa Mudigere、Naveen Mellem-pudi、Dipankar Das、Kunal Banerjee、Sasikanth Avancha、Dharma Teja Vooturi、Nataraj Jammalamadaka、Jianyu Huang、Hector Yuen、Jiyan Yang、Jongsoo Park、Alexander Heinecke、Evangelos Georganas、Sudarshan Srinivasan、Abhisek Kundu、Misha Smelyanskiy、Bharat Kaul 和 Pradeep Dubey。BFLOAT16 深度学习训练研究。arXiv 预印本 arXiv:1905.12322，2019。6[21] Bernhard Kerbl、Georgios Kopanas、Thomas Leimkühler 和 George Drettakis。用于实时辐射场渲染的 3D高斯泼溅。ACM Transactions on Graphics，42(4)，2023。1、2、4[22] Bernhard Kerbl、Andreas Meuleman、Georgios Kopanas、Michael Wimmer、Alexandre Lanvin 和 George Drettakis。用于超大型数据集实时渲染的分层三维高斯表示。ACM Transactions on Graphics，43(4)，2024。2[23] Shakiba Kheradmand、Delio Vicini、George Kopanas、Dmitry Lagun、Kwang Moo Yi、Mark Matthews 和 Andrea Tagliasacchi。StochasticSplats：用于无排序 3D高斯泼溅的随机光栅化。载于 IEEE/CVF国际计算机视觉会议（ICCV）论文集，第 26326–26335 页，2025。1、3[24] Arno Knapitsch、Jaesik Park、Qian-Yi Zhou 和 Vladlen Koltun。Tanks and Temples：大规模场景重建基准测试

。ACM Transactions on Graphics，36(4)，2017。7 [25] John Langford、Lihong Li 和 Tong Zhang。通过截断梯度实现稀疏在线学习。机器学习研究期刊，10(28)：777–801，2009。6[26] Donghyun Lee、Dawoon Jeong、Jae W. Lee 和 Hongil Yoon。GS-Scale：通过主机卸载释放大规模 3D高斯泼溅训练能力。载于第 31 届 ACM 编程语言与操作系统体系结构支持国际会议论文集，第 2 卷，第 860–875 页。计算机协会，2026。2、6、15[27] Junseo Lee、Seokwon Lee、Jungi Lee、Junyong Park 和 Jae-woong Sim。GSCore：通过面向 3D高斯泼溅的体系结构支持实现高效辐射场渲染。载于第 29 届 ACM 编程语言与操作系统体系结构支持国际会议论文集，第 3 卷（ASPLOS '24），2024。1、3[28] Joo Chan Lee、Daniel Rho、Xiangyu Sun、Jong Hwan Ko 和 Eunbyung Park。用于辐射场的紧凑型3D高斯表示。载于 IEEE/CVF计算机视觉与模式识别会议（CVPR）论文集，第 21719–21728 页，2024。2[29] Sixu Li、Ben Keller、Yingyan Lin 和 Brucek Khailany。GauRast：增强 GPU三角形光栅化器以加速 3D高斯泼溅。载于第 62 届 ACM/IEEE 设计自动化会议（DAC）论文集，2025。1、3[30] Kaimin Liao、Hua Wang、Zhi Chen、Luchao Wang 和 Yaohua Tang。LiteGS：通过系统与算法协同设计，在数分钟内训练 3DGS 的高性能框架。载于欧洲计算机视觉会议（ECCV），第 574–591 页，2026。2[31] Zimu Liao、Jifeng Ding、Siwei Cui、Ruixuan Gong、Boni Hu、Yi Wang、Hengjie Li、Hui Wang、Xingcheng Zhang 和 Rong Fu。TC-GS：利用张量核心的更快高斯泼溅模块。载于 SIGGRAPH Asia 2025 会议论文集，2025。1、2、5[32] Linjie Lyu、Ayush Tewari、Jianchun Chen、Thomas Leimkühler 和 Christian Theobald。通过结构感知致密化加速 3D高斯泼溅收敛。载于 SIGGRAPH 2026 会议论文集，美国纽约州纽约市，2026。计算机协会。2、18[33] Saswat Subhajyoti Mallick、Rahul Goel、Bernhard Kerbl、Markus Steinberger、Francisco Vicente Carrasco 和 Fernando De La Torre。Taming 3DGS：有限资源下的高质量辐射场。载于 SIGGRAPH Asia 2024 会议论文集，美国纽约州纽约市，2024。计算机协会。[34]

Henrique S. Malvar、Gary J. Sullivan 和 Sridhar Srinivasan。用于图像压缩的基于提升的可逆颜色变换。载于 Proc. SPIE 7073，《数字图像处理应用 XXXI》，第 707307 页，2008。5[35] Duane Merrill 和 Andrew S. Grimshaw。重新审视 GPGPU流架构中的排序。技术报告 CS2010-03，弗吉尼亚大学计算机科学系，2010。4、12[36] Ben Mildenhall、Pratul P. Srinivasan、Matthew Tancik、Jonathan T. Barron、Ravi Ramamoorthi 和 Ren Ng。NeRF：

将场景表示为神经辐射场以合成视图。载于《计算机视觉——ECCV 2020》，第 405–421 页。施普林格，2020 年。1[37] Nicolas Moenne-Loccoz、Ashkan Mirzaei、Or Perel、Riccardo de Lutio、Janick Martinez Esturo、Gavriel State、Sanja Fidler、Nicholas Sharp 和 Zan Gojcic。3D 高斯光线追踪：粒子场景的快速追踪。《ACM Transactions on Graphics》，43(6)，2024 年。3[38] Simon Niedermayr、Josef Stumpfegger 和 Rüdiger Westermann。用于加速新视角合成的压缩 3D 高斯泼溅。载于《IEEE/CVF计算机视觉与模式识别会议（CVPR）论文集》，第 10349–10358 页，2024 年。2[39] Michael Niemeyer、Fabian Manhardt、Marie-Julie Rakoto-saona、Michael Oechsle、Daniel Duckworth、Rama Gosula、Keisuke Tateno、John Bates、Dominik Kaeser 和 Federico Tombari。RadSplat：辐射场引导的高斯泼溅，实现稳健的实时渲染，达到 900+ 每秒帧数。载于《2025 年国际 3 144，2025 3 134 2 D 视觉（DV）会议》，第 – 页。[40] 英伟达公司。PhysicalAI-Autonomous-Vehicles-NuRec。https://huggingface.co/datasets/nvidia/PhysicalAI-Autonomous-Vehicles-NuRec，2025 年。数据集，版本 26.02。7[41] 英伟达公司。《CUDA 编程指南：程序化依赖启动与同步》。https://docs.nvidia.com/cuda/cuda-programming-guide/，2025 年。第 4.5 节。12[42] 英伟达公司。《并行线程执行指令集架构》第 9.3 版。https://docs.nvidia.com/cuda/parallel-thread-execution/，2025 年。5[43] Thomas Porter 和 Tom Duff。数字图像合成。载于《第 11 届年度计算机图形学与交互技术会议论文集》（SIGGRAPH '84），第 253–259 页。计算机协会，1984 年。4[44] Lukas Radl、Michael Steiner、Mathias Parger、Alexander Weinrauch、Bernhard Kerbl 和 Markus Steinberger。StopThePop：用于视图一致实时渲染的排序高斯泼溅。《ACM Transactions on Graphics》，43(4)，2024 年。1、2、5、16[45] Shiwei Ren、Tianci Wen、Yongchun Fang 和 Biao Lu。FastGS：100 秒内完成 3D 高斯泼溅训练。载于《IEEE/CVF计算机视觉与模式识别会议（CVPR）论文集》，第 26094 26103 页，2026 年。2 –。[46] Markus Schütz、Christoph Peters、Florian Hahlbohm、Elmar Eisemann、Marcus Magnor 和 Michael Wimmer。Splatshop：高效编辑大型高斯泼溅模型。《Computer Graphics Forum》（HPG 会议论文），44(8)，2025 年。1、2、4[47] Wei Tan、Matias Turkulainen、Lauri Ilola、Hamed Rezazadegan Tavakoli 和 Juho Kannala。TileGS：用于高斯泼溅光栅化的图块局部深度分箱。载于《Pacific Graphics 2026——会议论文与海报》。欧洲图形学协会，2026 年。2[48] Xinzhe Wang、Ran Yi 和 Lizhuang Ma。AdR-Gaussian：利用自适应半径加速高斯泼溅。载于《SIGGRAPH Asia 2024 会议论文集》，2024 年。1、2

[49] Zhou Wang、Alan C. Bovik、Hamid R. Sheikh 和 Eero P. Simoncelli。图像质量评估：从误差可见性到结构相似性。《IEEE 图像处理汇刊》，13(4)：600–612，2004 年。7[50] Qi Wu、Janick Martinez Esturo、Ashkan Mirzaei、Nicolas Moenne-Loccoz 和 Zan Gojcic。3DGUT：在高斯泼溅中支持畸变相机与次级光线。载于《IEEE/CVF计算机视觉与模式识别会议（CVPR）论文集》，2025 年。3[51] Mengtian Yang、Yipeng Wang、Chieh-Pu Lo、Xiuhao Zhang、Sirish Oruganti 和 Jaydeep P. Kulkarni。GSAcc：通过深度推测和以高斯为中心的光栅化加速 3D 高斯泼溅。载于第 62 届 ACM/IEEE 设计自动化会议（DAC）论文集，2025 年。1、3[52] 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。《机器学习研究期刊》，26(34)：1–17，2025 年。1、2、16[53] Zehao Yu、Anpei Chen、Binbin Huang、Torsten Sattler 和 Andreas Geiger。Mip-splatting：无混叠的 3D 高斯泼溅。载于《IEEE/CVF计算机视觉与模式识别会议（CVPR）论文集》，第 19447–19456 页，2024 年。3[54] Pedram Zamirai、Jian Zhang、Christopher R. Aberger 和 Christopher De Sa。重新审视 BFloat16 训练。arXiv 预印本 arXiv:2010.06192，2021 年。6[55] Richard Zhang、Phillip Isola、Alexei A. Efros、Eli Shechtman 和 Oliver Wang。深度特征作为感知度量的惊人效能。载于《IEEE/CVF计算机视觉与模式识别会议（CVPR）论文集》，第 586–595 页，2018 年。7[56] Tingjia Zhang、Bo Chen、Shengzhong Liu、Fan Wu 和 Guihai Chen。CaT-GS：通过帧间缓存和图块调度实现大规模场景的高效 3DGS 渲染。载于《IEEE/CVF计算机视觉与模式识别会议（CVPR）论文集》，第 37391–37400 页，2026 年。1、3[57] Hexu Zhao、Haoyang Weng、Daohan Lu、Ang Li、Jinyang Li、Aurojit Panda 和 Saining Xie。论如何扩展 3D 高斯泼溅训练。载于《国际学习表征会议（ICLR）》，第 35642–35661 页，2025 年。2

HiGS：一种分层 架构，用于快速可微分三维高斯溅射

补充材料

符号	含义
M	每个宏瓦片中的渲染瓦片数
B_p, B_r, B_d	分区、光栅化、排空批大小
P, W	每个工作组的线程数、每单元组数
K	一个批次的小批次
τ, w	线程索引，工作线程组索引
$\mathcal{S}$, $\mathcal{Q}$, $\mathcal{R}$	本地存储，图块队列，调度缓冲区
Σ, L, l_k	协方差， Σ^{-1}的Cholesky因子，条目
$\boldsymbol{\mu}$, $\mathbf{p}$	高斯分布的投影中心，像素
dx, dy; u_0, u_1; q	$\boldsymbol{\mu}-\mathbf{p}$; 在 L上的投影； $u_0^2+u_1^2$
o_j, d_j	高斯分布的不透明度， $\partial\mathcal{L}/\partial \ln \alpha_j$
ε	透射率阈值
$\mathbf{c}_i$, T_i	颜色, 批次 i 单独的透射率
F_i, $\mathbf{S}_i$	之前的透射率 , 从批次 i 开始的颜色
T_{end}, $\mathbf{b}$	最终透射率、背景颜色
t_j, α_j	局部透射率，像素处的不透明度
$\mathbf{c}_j$, $\widehat{w}_j$	颜色，舍入后的混合权重
$\mathcal{L}$,$\mathbf{g}$, $\mathbf{g}_i$, r	损失，其梯度， $F_i\mathbf{g}$，残差
θ, m, v	一个参数及其 Adam 矩
η, β_1, β_2, ε_A	学习率和 Adam常数
a, k, ρ	上次更新的步，滞后量， $\beta_1/\sqrt{\beta_2}$
Φ, Ψ	滞后期间步因子的总和

表S1. 符号说明。

A. 引言

本附录提供渲染流水线（第 B 节）和重建流水线（第 C 节）的实现细节，包括伪代码、数据布局和推导。随后报告有关性能和图像质量的其他结果（第 D 节）。我们沿用主论文中的符号，表 S1 对此进行了总结。我们将在论文发表后发布实现的源代码。

B. 渲染流水线

B.1. 打包的球谐系数

在主论文第 3.4 节的 32 字节球谐表示中，常数项保留 16 位亮度和 YCoCg 基每个色度通道 15 位。平方根压扩后，15 个高阶系数中的每个系数都量化为 6 位亮度和每个色度通道 4 位，并为每个球谐基函数和通道使用一个尺度，即其在不透明度不低于 0.005 的高斯中的幅值第 99.99 百分位数。相较于流式传输未压缩的 FP16 系数，直接从该表示中解码可使 1080p 下的投影和球谐函数计算时间缩短 35%，4K 下缩短 30%。

B.2. 分段深度排序

排序流水线的每个阶段对应一个内核，主机连续启动这六个内核。

阶段		线程	输入
级联	T0	64	包含 1 至 1,024 个键的段
	T1	128	包含 1,025 至 2,048 个键的段
	T2	256	包含 2,049 至 4,096 个键的段
	T3	512	包含 4,097 至 8,192 个键的段
运行排序		512	包含 8,192 个键的运行
合并		256	单个段的已排序运行

级联。 每个层级为每个段启动一个线程块，除非其段属于该层级的大小类别，否则线程块会立即返回。线程块使用 CUB [35] 的块基数排序，在 4 个 8 位遍次中，对 32 位深度键（即正 FP32 深度值的位模式）及其高斯 ID 进行排序。各层级不会读取其他层级写入的内容，因此四个层级通过程序化依赖启动 [41] 实现执行重叠。

更大段。 最后一个层级不会对超过 8,192 个键的段进行排序，而是将该段划分为每段 8,192 个键的运行，并铺设用于连接这些运行的成对合并二叉树。运行排序随后像层级对段进行排序一样，对每个运行进行排序，而合并则执行这些二叉树。每次合并都通过 GPU 合并路径 [11] 划分为均衡子问题；父合并会在其子合并完成后于设备上调度。这两个阶段读取前一阶段写入的内容，并按顺序运行。每个阶段都会运行固定数量的线程块，由这些线程块在设备上认领其工作：运行从柜台中获取，合并从队列中获取，因此主机无需读回一帧包含多少个大段。

B.3. 批量光栅化

算法S1 给出了处理单元及算法 S2对一个渲染图块的遍历过程，反向分支的推导见第 C.1节。单元中的每个线程都执行以下伪代码：工作组 $\tau \in [0, P)$ 中的 $w \in [0, W)$各线程都持有每个变量各自的副本；在算法S2 中，该副本的值对应于该线程所处理的像素对。以下原语在线程间执行：

- 屏障会等待，直到该单元的所有工作组都到达。
- VOTE(c) 是一个 P 位掩码，标示组中满足 c 的线程；如果组中所有线程都满足 c，则 UNANIMOUS(c) 为真。
- COMPACT(掩码) 返回掩码中置位比特的位置。

算法 S1. PROCESSINGUNIT($batch$, $pass$)：一个处理单元，由前向传播和反向传播共享（主论文第 3.3 节和第 4.1 节）。阶段 1 为每个线程分配一个高斯，以隐藏内存延迟；工作组 0 在第2阶段之前构建图块工作队列 $\mathcal{Q}$。

输入： 批次中的 $n \le B_r$ 个高斯，按深度顺序排列；
$K = \lceil n/P \rceil$ 个高斯的小批次；
$pass$ = 前向或反向

// 阶段 *1*：加载并内联可见性
对于 $k \leftarrow w$ **到** $K-1$，**步长为** W，**执行** ▷ 组跨小批次步进
  $j \leftarrow kP + \tau$； ▷ 每个线程处理一个高斯
  // 宏图块坐标系中的中心、因子、颜色
  if $j < n$ **then** $\mathcal{S}.gauss[j] \leftarrow$ 加载($batch$, j)；
  如果前向 **则**
    // M 位：如果高斯 j 与图块 t重叠，则设置 *bit* t
    $b \leftarrow$ OVERLAPTEST($\mathcal{S}, j$)，如果 $j \ge n$则为 0；
    // 线程 τ 接收图块 τ的 P 位
    $\mathcal{S}.mask[k][\tau] \leftarrow$ TRANSPOSE(b)；
  else
    // 前向传播保存的图块 τ 掩码
    $\mathcal{S}.mask[k][\tau] \leftarrow$ 已保存掩码(k, τ)；
  end
结束
屏障；
// 图块工作队列：组 *0* 的线程 τ 负责图块 τ
如果 $w = 0$ **那么**
  // 图块 τ处批次中的高斯
  $h \leftarrow \bigvee_{k=0}^{K-1} \mathcal{S}.mask[k][\tau]$；
  $a \leftarrow$ VOTE($h \ne 0 \wedge$ 图块 τ位于图像中)；
  // 反向：保留合成结果到达批次的图块
  if 反向 **then** $a \leftarrow a \wedge$ VOTE(已合成的图块 τ)；
  $\mathcal{Q} \leftarrow$ COMPACT(a)；
结束
屏障；
// 第*2*阶段：逐图块工作
当 $t \leftarrow$ 认领($\mathcal{Q}$)**时执行** ▷组动态认领图块
  RENDERTILE(t, $pass$)； ▷ 算法 *S2*
结束

• TRANSPOSE(b) 对由组中线程的值 b构成的 $P \times M$ 位矩阵进行转置，使线程 τ接收列 τ。

• 认领($\mathcal{Q}$)会以原子方式从队列中取出一个图块并将其返回；队列为空时则不返回任何内容。
• ATOMICADD会对一个值进行加法操作，而其他组和其他单元中的线程也会同时更新该值。
在此配置中，加载高斯小批次与宏瓦片的 M =32个渲染瓦片是两个恰好大小相同的维度，因此转置可以在一个工作组内完成。渲染时，我们每单元使用 W=10个工作组，训练时使用 W=8 个工作组。**数值格式。** Cholesky 因子 $L = \left[\begin{smallmatrix} l_0 & 0 \\ l_1 & l_2 \end{smallmatrix}\right]$ 经过预缩放，以吸收从自然指数到以 2 为底的指数的转换，以及图块单位坐标尺度的变化；对于大小为 s 像素的渲染瓦片，缩放因子为 $l'_k = l_k\, s\sqrt{\log_2(e)}/2$。因此，像素处的不透明度等于高斯的不透明度乘以 2^{-q}，并通过一次 FP16 以 2 为底的指数运算求得。在主论文第 3.4 节的宏图块坐标系中，像素坐标满足 $|x| \le 4$和 $|y| \le 2$。

算法 S2. 渲染图块渲染图块(t, $pass$)：一个工作组处理算法 S1 中认领的渲染图块，每个线程处理其像素对。两次遍次以相同方式遍历图块中的高斯，通过调度缓冲区 $\mathcal{R}$，按 B_d 的批次排空，并对每个排空批次应用不同的例程。饱和会提前结束遍历。

输入： $\mathcal{S}$ 和 K 算法S1；
反向：边界状态 F_i,$\mathbf{S}_i$, T_{end} 的图块
以及像素对处的损失梯度 $\mathbf{g}$

$T \leftarrow 1$； ▷ 批次内的透射率
如果前向 **则**
  $\mathbf{c} \leftarrow$ 0<style 标识符='2'>； ▷批次颜色
否则
  $\mathbf{g}_i \leftarrow F_i\,\mathbf{g}$； $r \leftarrow \mathbf{g}^\top(\mathbf{S}_i + T_{\text{end}}\mathbf{b})$； ▷ 残差
结束
$\mathcal{R} \leftarrow \varnothing$； ▷容量 $2B_d$
对 $k \leftarrow 0$**至** $K-1$**执行** ▷小批次，按深度顺序排列
  // 与图块 t重叠的小批次 k 中的高斯
  $\mathcal{R}$.ENQUEUE(kP +COMPACT($\mathcal{S}.mask[k][t]$))；
  // 完整排空，以及最后一个微批次之后的剩余部分
  当 $|\mathcal{R}| \ge B_d$ 或（$k = K-1$ 且 $|\mathcal{R}| > 0$）**时**
    $J \leftarrow \mathcal{R}$.出队(min(B_d, $|\mathcal{R}|$))；
    如果前向**则**
      前向到后向混合(J)；
    else
      前向后梯度(J)；
    end
    // 图块的每个像素都已饱和
    如果 UNANIMOUS($T < \varepsilon$) **则跳出**两个循环；
  end
结束
如果前向 **则将** ($\mathbf{C}$ T) 存储为图块 t 的部分 ($\mathbf{C}_i$, T_i)；

过程FRONTTOBACKBLEND(J)
  遍历高斯 j J，按深度顺序**执行**
    $\alpha \leftarrow$ min(1, $o_j\, 2^{-q}$)，低于 1/255时设为 0；
    $\widehat{w} \leftarrow \alpha T$；
    $\mathbf{c} \leftarrow \mathbf{c} + \widehat{w}\mathbf{c}_j$<style 标识符='4'>； $T \leftarrow T - \widehat{w}$；
  end
结束
过程FRONTTOBACKGRADIENT(J)
  对高斯 j 按深度顺序遍历 J**执行**
    $\alpha \leftarrow$ min(1, $o_j\, 2^{-q}$)，在 1/255以下设为 0；
    $\widehat{w} \leftarrow \alpha T$；
    $r \leftarrow r - \widehat{w}\,\mathbf{g}_i^\top \mathbf{c}_j$；
    // 若 d_j 为零，则 $\alpha = 1$或 $T = 0$
    $d_j \leftarrow \alpha(T\mathbf{g}_i^\top \mathbf{c}_j - r/(1-\alpha))$； $T \leftarrow T - \widehat{w}$；
    // 对图块中所有像素求和，公式(10)
    ATOMICADD(j的梯度, 求和结果)；
  end
结束

混合与输出。 FRONTTOBACKBLEND 是 3DGS 的混合递推关系，不透明度上限为 1，并在每次排空后测试透射率阈值 $\varepsilon = 10^{-4}$。每个单元将其部分结果写入一个按批次和渲染图块索引的稠密缓冲区；每个批次占用 16 KB，每个像素使用四个 FP16 通道。

算法 S3. 复合训练(t)：一个工作组
合成渲染图块 t，每个线程处理其像素对，并完成图块中各批次的边界状态。

输入： 图块中活动批次的部分结果 $(\mathbf{C}_i, T_i)$ $i = 0, \ldots, n_b-1$，按深度顺序排列
输出： F_i，$\mathbf{S}_i$ 合成批次 $0, .., last$；T_{end}；图像的像素

$F \leftarrow 1$；　$last \leftarrow n_b - 1$；
对于 $i \leftarrow 0$ **到** $n_b - 1$，**执行**　▷从前到后
　$F_i \leftarrow F$；　$F \leftarrow F\,T_i$；
　// 图块中的每个像素都已饱和
　如果 U全体一致$(F < \varepsilon)$**则** $last \leftarrow i$;**跳出**;
结束
$T_{\text{end}} \leftarrow F$; $\mathbf{S} \leftarrow \mathbf{0}$;
对于 $i \leftarrow last$ **到** 0**执行**　▷ 由后向前
　$\mathbf{S} \leftarrow \mathbf{S} + F_i\,\mathbf{C}_i$<style 标识符='4'>; S
　$_i \leftarrow$ S<style 标识符='7'>;
结束
图像 $\leftarrow \mathbf{S}_0 + T_{\text{end}}\,\mathbf{b}$;

C. 重建流水线

C.1. 反向光栅化

边界状态。 训练前向过程使用 FP32 累积颜色，因为 FP16 累积会淹没细微增量并使图像变暗，而且会让反向传播得到一个已经丢失了必须减去的项的后缀部分。算法 S3给出了完成图块各批次边界状态的合成过程。位于最后一个被合成批次之后的批次中的高斯将获得零梯度。

推导。 考虑一个像素以及深度顺序下批次 i 的高斯 j =1，...，n，其局部透射率为$t_1 = 1$，$t_{j+1} = t_j$($1 - \alpha_j$)，混合权重为 $w_j = \alpha_j t_j$。像素颜色为

$$\mathbf{C} = \sum_{h<i} F_h \mathbf{C}_h + F_i \sum_j w_j \mathbf{c}_j + \sum_{h>i} F_h \mathbf{C}_h + T_{\text{end}} \mathbf{b}. \quad (7)$$

第一项求和与批次 i 无关。高斯 j 后方的每一项，包括后续批次和背景，都会在其透射率中带有因子 $(1 - \alpha_j)$，因此

$$\frac{\partial \mathbf{C}}{\partial \alpha_j} = F_i t_j \mathbf{c}_j - \frac{1}{1-\alpha_j}\left(F_i \sum_{j'>j} w_{j'} \mathbf{c}_{j'} + \sum_{h>i} F_h \mathbf{C}_h + T_{\text{end}} \mathbf{b}\right). \quad (8)$$

与损失梯度$\mathbf{g}$ 做内积并写作$\mathbf{g}_i = F_i \mathbf{g}$，即可得到主论文中的公式（5），其中 $\widehat{w}_j$ 是 w_j的舍入值，其残差为

$$r_j = \mathbf{g}^\top \left(\mathbf{S}_i + T_{\text{end}} \mathbf{b}\right) - \sum_{j' \le j} w_{j'}\, \mathbf{g}_i^\top \mathbf{c}_{j'}. \quad (9)$$

主论文公式（4）的递推关系维持了这一点。该导数对应于前向传播所渲染的函数，其中将舍入视为恒等映射。在 $\alpha_j = 1$ 或 $t_j = 0$ 由不透明度导出的梯度被设为零，而前向传播在某个像素处跳过的高斯在该处恰好接收到零值。

从 α_j 开始，链式求导继续传递到光栅化器的输入。令 $(dx, dy) = \boldsymbol{\mu} - \mathbf{p}$ 表示中心相对于像素的偏移，l_k 表示 B.3 节中的预缩放因子B.3。令 $u_0 = l_0\,dx + l_1\,dy$ 且 $u_1 = l_2\,dy$，$q = u_0^2 + u_1^2$，并令 $\alpha_j = o_j 2^{-q}$表示不透明度 o_j，则相对于 $\ln \alpha_j$ 的梯度为 $d_j = \alpha_j\, \partial \mathcal{L}/\partial \alpha_j$，而逐高斯梯度是图块中各像素 p 上的求和：

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial o_j} &= \frac{1}{o_j} \sum_p d_j, & \frac{\partial \mathcal{L}}{\partial l_0} &\propto \sum_p u_0\,dx\,d_j, \\
\frac{\partial \mathcal{L}}{\partial l_1} &\propto \sum_p u_0\,dy\,d_j, & \frac{\partial \mathcal{L}}{\partial l_2} &\propto \sum_p u_1\,dy\,d_j, \\
\frac{\partial \mathcal{L}}{\partial \mu_x} &\propto l_0 \sum_p u_0\,d_j, & \frac{\partial \mathcal{L}}{\partial \mu_y} &\propto l_1 \sum_p u_0\,d_j + l_2 \sum_p u_1\,d_j,
\end{aligned} \quad (10)$$

每项都乘以常数 $-2\ln 2$，再乘以 B.3 节中用于因子的预缩放常数 B.3 以及用于中心的 $1/s$，因为偏移的单位是渲染图块。一个工作组会将这些和以及 $\widehat{w}_j\,\mathbf{g}_i$ 对颜色的贡献，以原子方式累加到高斯中心、其因子和不透明度以及颜色的梯度中，共执行五次每次两个字的加法。

C.2. 融合优化器与随机舍入

融合处理过程使用 Adam 和 $\beta_1 = 0.9$、$\beta_2 = 0.999$、$\varepsilon_A = 10^{-15}$。参数根据未舍入的矩估计进行更新。颜色对视线方向的梯度会影响平均值的梯度，因此平均值最后更新，在球谐函数系数之后。由于梯度不会保留，3DGS 的致密化统计量——视图数量和中心屏幕空间梯度的范数——会在同一处理过程中累积。

存储时，会将均匀分布的随机比特加到要舍弃的比特上，然后截断：BF16 矩估计舍弃 16 位，FP16 参数舍弃 13 位。这些比特由运行种子、步和元素索引的哈希值生成，因此无需存储生成器状态，也无需在致密化过程中传递该状态。因此，在该格式的正规数范围内，存储值是无偏的。将 FP16 参数扩展为 FP32 是精确的，因此，采用这种布局训练的场景，其渲染结果与存储相同数值的 FP32 场景逐位一致。

C.3. 延迟外观更新

调度。 在步骤 s，包含可见高斯的组首先根据主论文中的公式 (6)，将其所有高斯的系数推进至步骤 $s - 1$，然后执行步骤 s，同时其剔除成员的梯度为零。步骤 s 的前向和反向传播均使用恢复至步骤 $s - 1$的系数。每个滞后组都会在此之前更新到最新状态

场景	HiGS	SHARP-GS	Faster-GS	FlashGS	TC-GS	Speedy-Splat	StopThePop	gsplat	3DGS
	1080p吞吐量（每秒帧数）								
自行车	2053	806	640	582	468	415	374	357	163
盆景	4219	2137	1481	1546	1443	1156	1036	979	557
柜台	3546	1730	1370	1212	1242	925	789	725	343
花园	1901	839	583	591	466	418	375	361	222
厨房	2632	1230	1021	852	916	679	615	554	295
room	3906	2008	1410	1466	1232	934	809	781	331
树桩	2688	1235	996	912	682	625	560	535	371
	4K吞吐量（每秒帧数）								
自行车	1309	756	465	508	356	296	218	205	60
盆景	1934	1715	644	1121	839	608	481	450	167
柜台	1698	1366	720	756	643	456	317	299	110
花园	1292	735	441	480	336	278	225	212	94
厨房	1443	972	568	544	488	343	265	246	106
room	1736	1460	733	913	708	497	348	327	104
树桩	1534	1087	615	796	492	407	341	312	140

表 S2。在七个 Mip-NeRF 360 场景的预训练 3DGS 模型上，每个场景的渲染吞吐量，单位为帧每秒：1000 除以中位数帧时间（毫秒）。TC-GS 使用其与 Speedy-Splat 光栅化器集成发布的版本运行；StopThePop 使用原始预设，该预设按每个图块的深度排序；SHARP-GS 使用预计算的协方差，并在 1080p 下采用大小为 32×32 像素的分箱图块，在 4K 下采用大小为 64×64 。FlashGS 在 4K 下无法处理一个厨房相机，因此该相机被排除在其中位数计算之外。

进行致密化、执行不会紧跟优化器步的渲染（例如评估）以及创建检查点，从而确保训练步骤之外的任何操作都不会观察到滞后的系数。在花园中，一次步骤的视图在致密化期间平均会剔除 65% 的高斯，在致密化后则会剔除 66%。

闭式解的推导。 按照主论文公式 (6) 的符号说明，零梯度 Adam 步 $a+j$ 会将 m缩放 β_1 倍，将 v 缩放 β_2 倍，并使系数 θ 移动

$$-\frac{\eta}{1-\beta_1^{a+j}} \cdot \frac{\beta_1^j m}{\beta_2^{j/2}\sqrt{v}/\sqrt{1-\beta_2^{a+j}}+\varepsilon_A}. \qquad (11)$$

当 ε_A 相对于 $\sqrt{v}$可忽略时， k 步的总和为$-\eta\,\Phi\,m/\sqrt{v}$; 而当它占主导时，总和为 $-\eta\,\Psi\,m/\varepsilon_A$。当分母中的另一项可忽略时，主论文公式 (6) 分别简化为这两个极限；当 11 时，则简化为公式 ($k=1$。在这两个极限之间，该结果从不会低估移动的幅值，因为单步移动关于它的两个项是凹且齐次的。对于在训练第一步最后一次更新的系数，高估幅度最大，为 4.5%；从第 600 步起，高估幅度始终低于 0.003%。当 $\varepsilon_A = 0$时，闭式解与 GS-Scale 的恢复结果一致；GS-Scale [26], 按滞后最多 15 步的情况列出因子 $\eta\,\Phi$ 。当 ε_A 占主导时，该恢复结果与步数总和相差 Φ/Ψ 倍：第 100 步时为 0.31 至 0.32，而从第 5,000 步起与 1 的差距不超过 0.5%。当 $\Phi(a)$ 和 $\Psi(a)$ 时，同样的总和延伸至无穷大，

$$\Phi = \bar{\Phi}(a) - \rho^k\,\bar{\Phi}(a+k), \qquad \Psi = \bar{\Psi}(a) - \beta_1^k\,\bar{\Psi}(a+k), \qquad (12)$$

因此，每步包含两个条目的表就足够了。对于 $\beta_1^2 < \beta_2$, 这些总和会收敛。恢复操作会对系数进行舍入以及矩估计进行一次处理，而不是对每个跳过的步分别处理一次。在独立设定种子的舍入中，滞后 8 步时，恢复系数的方差是密集更新系数方差的 0.16 倍；在滞后 38 步时则为 0.05 倍。

C.4. 空间排序下的致密化

致密化步骤会获取均值的莫顿码，并将其限制在一个网格中。该网格每个轴包含 2^{10} 个单元，在训练开始时，根据初始均值的边界框确定，并向外扩展 10%。收集操作会移除被剪枝的高斯，并将矩估计与参数一并传递；随后，将每个组记录的步设为当前步。在相同状态和随机序列下，得到的高斯集合及优化器状态与 gsplat 的致密化结果一致。

D. 附加结果

D.1. 渲染性能

表S2按场景扩展了主论文表 2 中的渲染对比。HiGS 在高斯数量最多的场景（自行车、花园）中的优势最大；在按阶段计时的两种基线方法 gsplat 和 FlashGS 中，投影、分箱和排序耗时约占一半帧时间甚至更多。而在 4K 分辨率的小型室内场景中，相较于 SHARP-GS 的优势最小。

成对约简有两个来源（表S3）。通过椭圆相交来细化 gsplat 在渲染瓦片上的边界框，可在 1080p 下减少约四分之一的配对，在 4K 下减少三分之一以上。将相同的测试移至宏瓦片粒度后，可在 1080p 下进一步减少剩余配对的一半，在 4K 下减少近三分之二，因此它在两种分辨率下都是更大的来源。它的扩展性也更好：从 1080p 提升到 4K 时

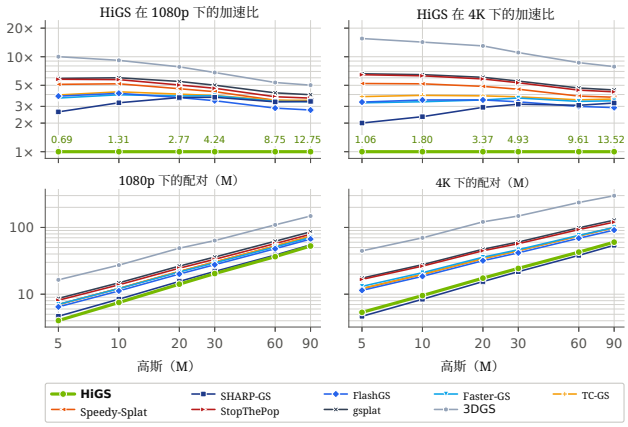


图 S1。 ***在garden* 场景**上的扩展：该场景使用从 5M 到 90M 的六种高斯数量配置进行重建，并从其测试集的第一台相机进行渲染。 顶部： HiGS 相对于每种光栅化器的加速比，即在相同配置和分辨率下，对 400 帧的帧时间中位数之比。HiGS 线上方的数字表示其在各配置下的帧时间（毫秒）。 底部： 每种方法所排序的瓦片–高斯对：HiGS 使用宏图块，其他方法使用 16×16 渲染图块（16×12 用于 Faster-GS）。SHARP-GS 的计数方式有所不同：它统计的是其分箱图块（表 S2）中的数量，这些图块来自包围矩形，并且统计发生在其排序之前的椭圆测试之前。

	1080p（M 配对）			4K（M 配对）		
场景	gsplat	椭圆	HiGS	gsplat	椭圆	HiGS
自行车	5.16	3.97	2.15	11.08	7.01	2.95
盆景	2.37	1.73	0.74	6.38	4.05	1.21
柜台	3.71	2.62	1.05	10.25	6.21	1.80
花园	5.50	4.47	2.60	10.82	7.58	3.41
厨房	4.79	3.74	1.81	11.32	7.72	2.69
room	3.03	2.06	0.77	8.83	5.19	1.39
树桩	3.53	2.72	1.45	7.79	5.04	2.03
Mean	4.01	3.04	1.51	9.49	6.11	2.21

表 S3。在七个 Mip-NeRF 360 场景中，各测试相机下 Tile–Gaussian 对计数的平均值（单位：百万）。*gsplat*：gsplat 库在评估版本 [52] 中的渲染图块分区；该方法使用感知不透明度的包围盒 [44], 将每个高斯限定在 16×16 个渲染图块内。椭圆：在相同图块上，使用逐行椭圆相交对相同包围盒进行细化 [16], 。*HiGS*：在宏瓦片粒度下进行椭圆细化测试，包含 8×8 个渲染图块。

宏瓦片对增长了 1.3× 到 1.8×，在每个场景中都低于 gsplat。

段数同时增加四倍，而排序的响应有所不同。在我们的 32 位键和宏瓦片片段上，通用分段排序 CUB 的速度下降了 1.9×至 2.7×，在每个场景中，降幅都超过了配对数量的增长幅度。我们的排序流水线（第 B.2节）的速度下降幅度小于配对数量的增长幅度：在五个场景中，其时间变化最多为 2%；在配对数量最少的两个场景中，最多增加 1.6× 。gsplat 的全局基数排序

	1080p（毫秒）				4K（毫秒）			
	8×8		16×16		8×8		16×16	
场景	gsplat	我们的方法	gsplat	我们的方法	gsplat	我们的方法	gsplat	我们的方法
自行车	3.51	0.49	2.81	0.62	8.13	0.76	4.88	0.87
盆景	1.41	0.24	1.02	0.29	4.94	0.52	2.22	0.50
柜台	2.22	0.28	1.38	0.35	7.36	0.59	3.35	0.63
花园	3.46	0.53	2.77	0.68	7.54	0.77	4.72	0.93
厨房	2.66	0.38	1.81	0.50	7.90	0.69	4.07	0.79
room	1.94	0.26	1.28	0.32	6.75	0.58	3.05	0.57
树桩	2.24	0.37	1.87	0.44	5.85	0.65	3.20	0.65
Mean	2.49	0.36	1.85	0.46	6.92	0.65	3.64	0.71

表 S4. 在七个 Mip-NeRF 360 场景中，gsplat 和 HiGS 在两种渲染图块尺寸下的中位帧时间。

速度下降了 2.9×至 4.6×。

无论选择何种渲染图块，分解都很稳健（表 S4）。使用 16×16 像素的图块后，gsplat 在每个场景中的帧时间都缩短了 17% 至 55%。HiGS 则没有类似的优化空间，因为其分区开销已经很小：在 1080p 下，较大的图块会拖慢所有场景；在 4K 下，帧时间变化 −4% 至 +21%。因此，主论文中的比较采用 gsplat 的 16×16设置，即其较快的设置。在图块尺寸相同的情况下，HiGS 在 1080p 下快 8×8，在 4K 下快 6.9× 。 10.6×

在 Mip-NeRF 360 的 花园 场景中（图 S1），HiGS 在所有预算和两种分辨率下都领先于所有方法。从最小模型到最大模型，高斯数量增加 18 倍，其帧时间在 1080p 下增长 18.4× ，在 4K 下增长 12.8× 。随着模型增大，HiGS 对 gsplat 的领先优势缩小，这是因为所有方法都要承担的逐高斯投影所占帧时间比例不断增加：在 1080p 下，逐高斯投影占 HiGS 帧时间的比例从 5M 时的 29% 增至 90M 时的 38%。在逐瓦片流水线中，各方法的帧时间排序与配对数量一致：在图 S1的两行中，每个预算下 3DGS 都慢于 gsplat 和 StopThePop，而这两者又慢于配对数量最少的 FlashGS、Faster-GS、TC-GS 和 Speedy-Splat。SHARP-GS 在两个最小模型上是最快的基线方法，之后则落后于 FlashGS；HiGS 对它的领先优势随着模型增大而扩大。原因在于分箱和光栅化：从 5M 增至 90M 时，它们在 4K 下耗时增加 21.7× ，而 HiGS 的对应耗时增加 10.2× ；两者的投影耗时则增长相近。

D.2. 训练性能

在每个场景中，HiGS 在稳态训练步骤的每个阶段都最快（图 S2）。其每步平均耗时为 1.64 ms，而 Faster-GS、gsplat 和 3DGS 分别为 4.45 ms、10.20 ms 和 20.44 ms。分箱和两遍光栅化的工作量取决于瓦片–高斯对，速度快 3×至 19 倍5× 。在其他方法中则更慢；而在 3 DGS 中，仅反向光栅化就占了该步骤一半以上的时间。两次投影和优化器的工作量均与高斯数量成正比，

		Mip-NeRF 360							Tanks and Temples			Deep Blending		
	方法	自行车	花卉	花园	树桩	Treehill	盆景	柜台	厨房	房间	火车		卡车	Dr. Johnson		游戏室
PSNR	3DGS	25.62	21.93	27.81	26.88	22.97	32.48	29.21	31.56	31.80	21.23		25.54	29.48		30.18
	gsplat	25.63	21.95	27.82	26.93	22.97	32.41	29.20	31.66	31.81	21.29		25.53	29.49		30.16
	Faster-GS	25.62	21.96	27.82	26.91	22.99	32.43	29.21	31.58	31.81	21.28		25.55	29.47		30.20
	HiGS	25.63	21.95	27.82	26.93	22.98	32.38	29.19	31.65	31.75	21.25		25.55	29.45		30.14
SSIM	3DGS	.7725	.6179	.8725	.7798	.6509	.9482	.9165	.9336	.9289	.7952		.8820	.9060		.9093
	gsplat	.7751	.6195	.8728	.7815	.6522	.9481	.9166	.9340	.9291	.7957		.8820	.9058		.9092
	Faster-GS	.7742	.6197	.8728	.7813	.6518	.9482	.9167	.9336	.9290	.7955		.8823	.9058		.9094
	HiGS	.7751	.6196	.8729	.7817	.6521	.9482	.9165	.9337	.9288	.7954		.8821	.9055		.9090
学习感知图像块相似度	3DGS	.1652	.3252	.0727	.1480	.3155	.1166	.1427	.0854	.1525	.1862		.0956	.1704		.1595
	gsplat	.1595	.3227	.0722	.1452	.3088	.1164	.1421	.0849	.1520	.1851		.0954	.1699		.1588
	Faster-GS	.1610	.3228	.0722	.1455	.3111	.1162	.1423	.0851	.1524	.1856		.0953	.1699		.1591
	HiGS	.1595	.3223	.0721	.1449	.3074	.1155	.1418	.0847	.1513	.1848		.0948	.1687		.1576
高斯 (M)	3DGS	4.60	2.70	3.58	4.12	3.04	1.06	1.03	1.57	1.26	1.04		2.01	3.14		1.87
	gsplat	4.78	2.76	3.64	4.15	3.11	1.09	1.06	1.60	1.31	1.06		2.04	3.27		1.94
	Faster-GS	4.73	2.75	3.62	4.12	3.08	1.08	1.05	1.59	1.30	1.05		2.04	3.25		1.92
	HiGS	4.79	2.75	3.64	4.13	3.12	1.09	1.06	1.60	1.30	1.06		2.04	3.27		1.93
内存 (GiB)	3DGS	6.57	3.90	5.13	5.90	4.32	1.65	1.79	2.55	2.06	1.57		2.90	4.72		2.74
	gsplat	6.98	4.06	5.34	6.06	4.57	1.69	1.66	2.43	2.00	1.60		3.01	4.83		2.89
	Faster-GS	10.34	6.05	7.96	9.14	6.76	2.48	2.39	3.53	2.93	2.33		4.49	7.20		4.28
	HiGS	3.52	2.07	2.72	3.06	2.32	1.04	1.04	1.29	1.13	0.84		1.55	2.53		1.50
Time (s)	3DGS	759.9	537.9	659.9	600.4	571.6	551.5	663.2	792.1	721.7	373.7		425.8	858.3		672.1
	gsplat	481.0	296.3	404.8	395.4	324.6	182.0	205.7	274.9	219.9	177.1		250.6	348.5		239.2
	Faster-GS	191.1	122.7	170.3	160.1	130.6	84.1	93.8	126.9	95.1	69.6		101.1	139.4		97.9
	HiGS	60.1	44.4	61.2	47.5	46.3	38.4	44.9	58.9	41.6	33.3		39.7	44.6		35.6

表 S5。**各场景重建**，均值为十次运行的平均值；时间和内存则为三次运行的平均值。测量指标与主论文中的表 1 相同，最后一个数字表示高斯数量（百万）。

	相较于 gsplat			相较于测试集		
方法	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
Ours 我们的方法，打包球谐	67.16 55.60	.9999 .9995	.0000 .0007	28.94 28.93	.8721 .8718	.1285 .1286
FlashGS	49.74	.9983	.0010	28.92	.8713	.1282
Faster-GS	73.86	1.0000	.0000	28.94	.8721	.1287
Speedy-Splat	94.43	1.0000	.0000	28.94	.8721	.1287
StopThePop	94.37	1.0000	.0000	28.94	.8721	.1287
3DGS	75.04	1.0000	.0000	28.94	.8721	.1287
gsplat	—	—	—	28.94	.8721	.1287

表 S6。对七个 Mip-NeRF 360 场景的预训练 3DGS 模型进行渲染时的图像质量，取各场景平均值。左：在 1080p 下与 FP32 gsplat 渲染器进行对比，此时模型保持不变，从而单独评估渲染器。右：与每个场景标准分辨率下的测试集进行对比。

它们的耗时取决于方法处理其参数的方式。Faster-GS 的融合优化器会在每一步更新每个高斯，因此在这些操作上的耗时是 HiGS 的 3.7× 倍。gsplat 和 3DGS 的耗时均为 9.8× 倍，因为它们还会拼接球谐函数系数、写入完整的参数梯度，并将 Adam 作为单独的一遍运行。采用融合优化器的反向投影占 HiGS 步骤耗时的 35%，占 Faster-GS 步骤耗时的 54%。

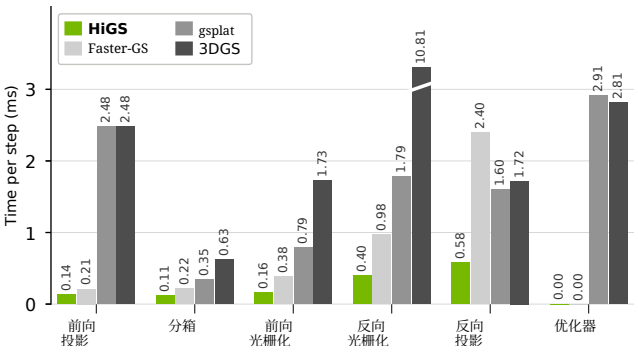


图 S2。**稳态训练步骤的阶段**：在致密化结束后的各个步骤中，以及主论文表 1 所列 13 个场景上测得的每个阶段的平均时间（单位：毫秒）；每个场景运行一次，并在阶段边界处使用 CUDA 事件计时。前向投影包括 SH 计算，分箱深度排序，前向光栅化HiGS 的合成过程，以及反向投影SH 梯度；对于 HiGS 和 Faster-GS，还包括融合优化器，因此只有 gsplat 和 3DGS 具有优化器阶段。未显示的是损失及其梯度（所有方法均使用相同代码计算），以及步骤中这些阶段之外的剩余部分，平均分别为 0.25 毫秒和 0.01 毫秒。

在 花园 上的稳态训练步骤中，融合优化器的运行速度达到

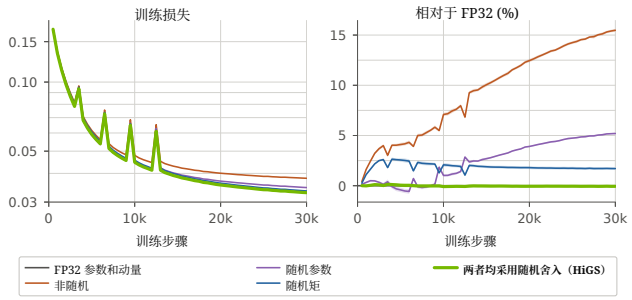


图 S3. **花园场景上采用和不采用随机舍入时的训练损失**，其中 FP16 参数和 BF16 矩分别采用随机舍入或舍入到最近值。左：以 500 步为窗口的损失，取十次运行的平均值。右：相对于 FP32 参数和矩的相同损失；每次运行均与相同随机种子的 FP32 运行结果比较，并显示均值的 95% 区间。

GPU 峰值内存带宽的 80%，而前向和反向光栅化受指令吞吐量限制，而非内存限制。每一步都会恢复 13.1 % 的高斯外观，即前一步未更新的组中的高斯；其中 96.0% 的高斯最多滞后 16 步。

对于每个场景（表 S5），HiGS 使用的内存比其他方法中内存效率最高的方法少 1.6× 到 1.9×，最终的高斯数量与 gsplat 相差不超过 0.02M。在每个场景中，其峰值信噪比与 3DGS 和 gsplat 相差不超过 0.10 dB；在部分场景中更高，在其他场景中则更低。

训练损失（图 S3）在整个运行过程中呈现的趋势，与主论文表 3 的图像指标在运行结束时所衡量的情况一致。若每次存储都舍入到最近值，最后 5,000 步的损失比 FP32 参数和矩高 14.9%，对应表中的第二行，而且差距仍在扩大。漂移源于矩：若只有矩舍入到最近值，差距仍会扩大，最后 5,000 步高出 5.0%。二阶矩在每一步都会乘以因子 β_2，即变化 0.1%；而 BF16 数值间距为 0.4% 至 0.8%。因此，每一步都更新的参数，其二阶矩会被舍入回此前的值，只会不断增大。这样一来，它记录的便是该参数遇到过的最大梯度平方，而非近期平均值；随着训练推进，Adam 除以该值所得的步长也随之缩小。若只有参数舍入到最近值，差距则稳定在 1.7%，这是存储时舍弃小于 FP16 数值间距一半的更新所造成的代价。两者的影响并非简单相加：参数和矩都舍入到最近值的代价约为两者之和的两倍。若两者都采用随机舍入，整个运行过程中的损失与 FP32 相差不超过 0.2%；我们将剩余差异归因于每次存储引入的噪声。第 15,000 步之前，相对损失的波动与不透明度重置相对应。

HiGS 不受 3DGS 致密化策略的限制，并能与更新的方法结合。采用 SAD-GS 的结构感知致密化训练 3,000 步后 [32], it

	方法	Mip-NeRF 360	坦克与神庙	Deep 混合
Time (s)	SAD-GS	31.2	19.0	15.6
	HiGS + SAD-GS	7.1	4.6	4.1
PSNR (dB)	SAD-GS	27.15	22.50	28.97
	HiGS + SAD-GS	27.29	22.52	29.08
高斯 (M)	SAD-GS	3.06	1.35	0.88
	HiGS + SAD-GS	2.90	1.29	0.78

表 S7. **采用 SAD-GS 致密化的重建**：使用主论文的测试框架，并以 SAD-GS 的致密化和训练设置替代 3DGS 策略，运行 3,000 步。*HiGS + SAD-GS* 采用抗锯齿进行光栅化。时间不包括两种方法在训练前对训练图像进行的结构分析。对每个数据集中的场景和三次运行取平均值。

是 3。8× 是 4。4× 在相同峰值信噪比下比 SAD-GS 本身快S7。SAD-GS 减少一次运行所需的步数，而 HiGS 降低每一步的成本。

D.3. 图像质量

在测试集上，各光栅化器之间的差异低于训练模型的残差误差（表 S6）。与 FP32 gsplat 渲染器相比，差异达数十分贝，而较低的数值则可追溯到特定的设计选择。Speedy-Splat 和 StopThePop 能以数值精度复现参考结果。3DGS 的固定 3σ 图块边界可能会遗漏高于 $1/255$ 阈值的贡献，正如 StopThePop 所指出的；Faster-GS 在合成后才测试终止条件，且不会将不透明度限制在上界以内；FlashGS 则写入 8 位像素缓冲区。每通道 8 位时，HiGS 的误差水平对应于约 0.1 个代码值的均方根误差。