3D Gaussian Splatting：从原始论文到公式链路与应用发展

申家芮

组会汇报

2026 年 5 月 28 日



Source: Kerbl et al., 3D Gaussian Splatting project page

报告结构

1. 从原始 3DGS 论文入手
2. 从论文回到公式：相机投影、渲染与可微优化
3. 原始 3DGS 公式与训练机制
4. 3DGS 数学链路与实验理解
5. 3DGS 的学术发展与代表应用
6. 总结：3DGS 作为显式三维表示平台
7. 补充讲解：按视频逻辑回看原始 3DGS 方法

Part 1. 原始 3DGS 论文的任务设定

- **输入：**多视角训练照片，以及每张照片对应的 camera 信息。
- **学习对象：**一组带有位置、形状、透明度和颜色的 3D Gaussians。
- **目标：**在新的 camera 视角下生成高质量图像，并达到实时渲染速度。

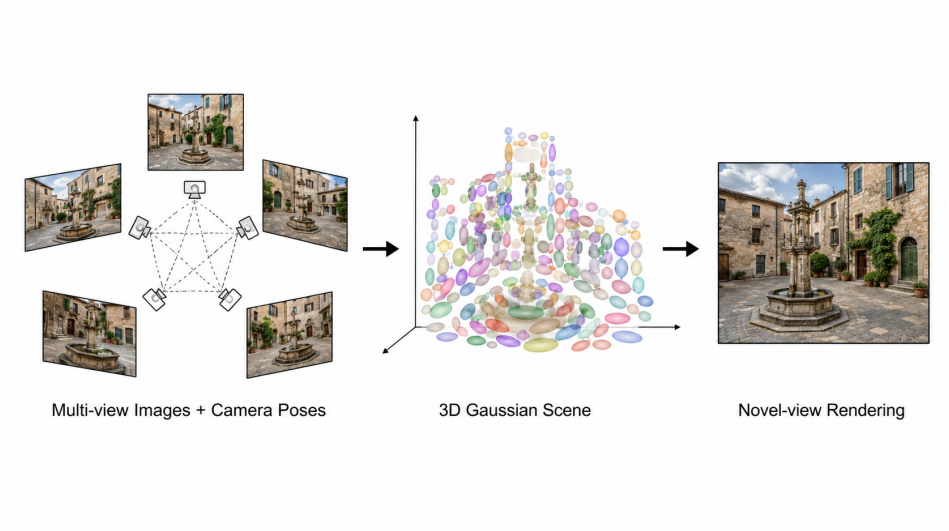
原始论文的核心问题可以概括为：如何把真实场景学习成显式、可渲染、可优化的 Gaussian scene representation。



Source: Kerbl et al., 3DGS project page

论文任务设定：多视角照片到实时新视角渲染

3D Gaussian Splatting for Real-Time Radiance Field Rendering 的任务可以先理解为：

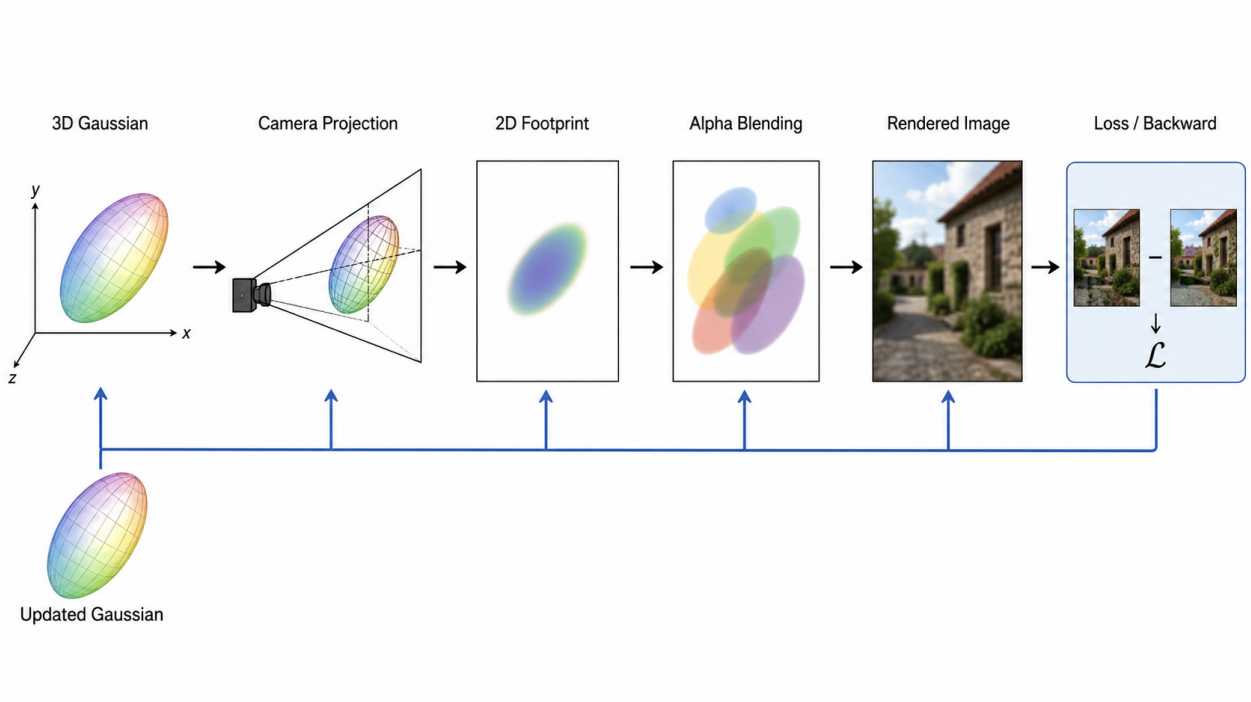


Source: Generated teaching diagram

- **输入：**多视角照片、camera 信息和较粗初始几何。
- **学习对象：**带位置、形状、透明度和颜色的 Gaussian scene。
- **监督与输出：**渲染图像接近训练照片，并支持新视角实时渲染。

这篇论文关心的是：把多视角照片中的场景学习成一组可实时渲染的 Gaussian。

3DGS 技术链路：从 Gaussian 到图像监督



Source: Generated teaching diagram

- Gaussian 在三维世界中有位置、尺度、旋转、透明度和颜色。
- 相机模型把 3D Gaussian 投影到图像平面，形成 2D footprint。
- 多个 footprint 通过 alpha blending 合成图像；loss 再反向更新 Gaussian 参数。

这条链路描述的是 3DGS 技术本身：表示、投影、合成和反向更新。

Part 2. 投影、渲染与可微优化的必要背景

- **相机投影：**把世界坐标中的 Gaussian 对齐到训练图像的像素位置。
- **渲染直觉：**理解 footprint、遮挡、颜色累积和屏幕像素之间的关系。
- **可微优化：**渲染图像与真实图像的误差，需要能反向传回 Gaussian 参数。

这些背景共同服务于 3DGS 的一条路径：三维 Gaussian 经过相机和渲染器变成二维图像监督。

3DGS 前置知识：相机投影

- 3DGS 的监督来自二维图片，但 Gaussian 本身在三维世界坐标中。
- 每个训练视角都需要说明相机从哪里看、3D Gaussian 投影到哪里，以及它覆盖哪些像素。
- 因此相机模型承担 3DGS 渲染和训练中的几何桥梁作用。



Source: MIT Vision Book

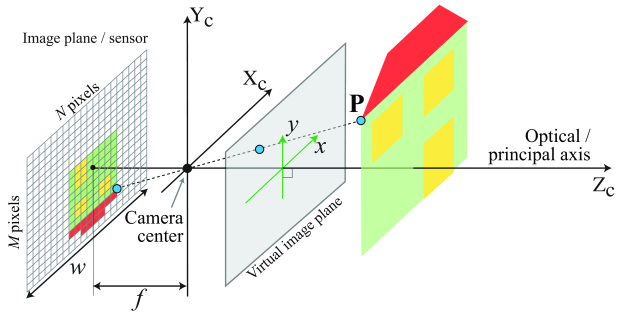


相机几何决定 3D Gaussian 如何和真实照片对齐。

从世界坐标到像素坐标：够用版链路



$$X_c = RX_w + t, \qquad s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R|t]\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$



Source: MIT Vision Book

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

- $X_c = R_{cw}X_w + t_{cw}$ 表示 world-to-camera 约定。
- t_{cw} 是坐标变换中的平移项，**并非相机中心**。
- 透视除法回答“近大远小”；内参回答“落到哪个像素”。

Gaussian 的中心、形状和 footprint 都通过这条投影链路进入图像坐标。

参考：Lei Mao, Camera Intrinsics and Extrinsics; PyTorch3D Cameras

相机位姿和初始化来源

- 经典多视角 3DGS 训练需要相机几何和较粗的初始表示。
- 这些信息可以来自多种来源：
  - 数据集标注或仿真环境；
  - RGB-D、LiDAR、SLAM 或多视角估计；
  - 深度/几何基础模型；
  - 网络直接预测 Gaussian。
- 关键在于理解：**位姿如何参与投影，初始化如何影响优化稳定性。**

3DGS 的训练依赖相机几何和可优化 3D representation，具体来源可以多样化。

论文背景：radiance field 与新视角合成

- 新视角合成的目标是：给定已有视角的观测，生成未见过相机视角下的图像。
- Radiance field 方法把三维场景表示成“从空间位置和观察方向到颜色/密度”的场。
- 原始 3DGS 延续新视角合成任务，但把场景表示从函数查询转向显式 Gaussian primitives。



这一背景把问题收束到 representation 选择：场景如何存，决定后续如何渲染和训练。

一种常见表示：隐式 neural field

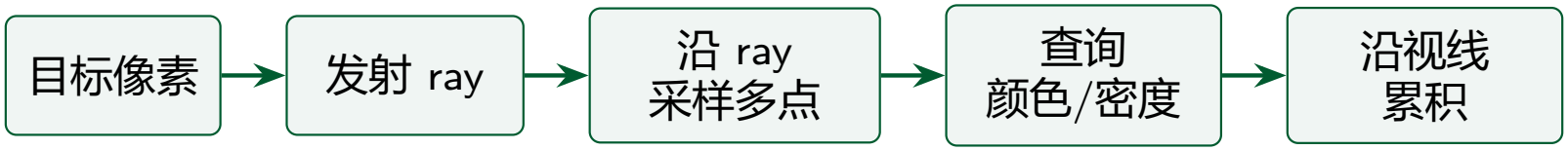
$$F_\theta(x, d) \to (c, \sigma)$$

- 在 radiance field 思路中，场景可以写成一个可查询函数。
- x：三维空间位置；d：viewing direction。
- c：当前点从该方向观察到的颜色；σ：密度或遮挡强度。
- θ：神经网络参数，训练后承载场景信息。
- 渲染一个像素时，通常沿 camera ray 采样多个点，多次查询函数，再把颜色和密度累积成像素颜色。
- 这种表示的场景信息主要藏在函数参数中；3DGS 采用显式 Gaussian，让可学习元素直接位于三维空间中。

对比的重点是 representation：函数式表示依赖查询；显式 Gaussian 表示可以直接投影到屏幕。

参考：PyImageSearch neural field tutorial

ray-based 渲染：从像素发出光线



- 每个像素对应一条 camera ray。
- ray 上的多个采样点提供颜色和密度。
- 前方密度或透明度会影响后方颜色的可见程度。
- 对 3DGS 有用的部分是颜色累积直觉：前面的贡献会遮挡或削弱后面的贡献。

3DGS 也要处理遮挡和颜色累积；贡献来自屏幕上的 Gaussian footprint，并与沿 ray 的密集采样形成对照。

Rasterization：从空间元素到屏幕像素

核心动作

给定一组三维场景元素，先把它们投影到屏幕上，再判断每个元素覆盖哪些像素，并计算这些像素的颜色。

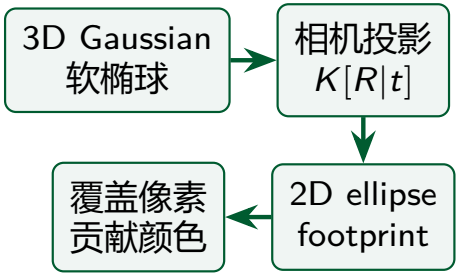
场景元素	屏幕上变成什么	需要计算什么
mesh triangle	屏幕上的三角形区域	哪些像素在三角形内部
3D Gaussian	屏幕上的椭圆 footprint	哪些像素受这个 Gaussian 影响

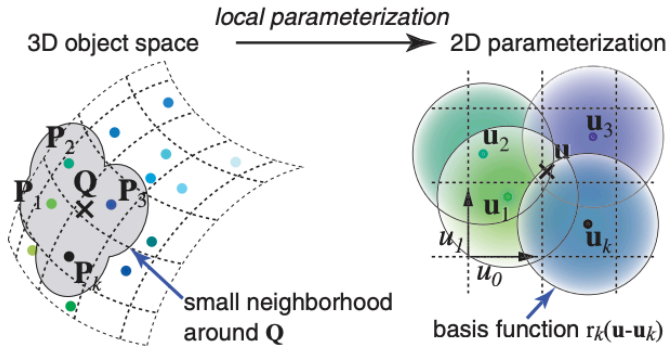
- Rasterization 回答的问题是：一个已经存在于三维空间中的 primitive，会影响屏幕上的哪些像素？
- 3DGS 中的 primitive 是 Gaussian，投影到屏幕后形成椭圆 footprint。
- 找到受影响像素之后，再计算权重、透明度和颜色混合。

Rasterization 给 3DGS 提供屏幕空间组织方式：先找到受影响像素，再进行 splatting 和 alpha blending。

参考：Hugging Face 3DGS blog

Splatting：从 3D Gaussian 到屏幕 footprint





Source: Shi Yan, How to Render a Single Gaussian Splat

- Splatting 可以理解成把软的三维小云团“摊”到屏幕上。
- Gaussian 在图像上形成有面积、有软边界的椭圆 footprint。
- footprint 覆盖到的像素都会受到颜色和透明度影响。

Splatting 是 3DGS 把显式 3D representation 变成图像的核心桥梁。

参考：Chauffier, Exploring 3DGS; Scthe, Rendering in 3DGS

Alpha blending：多 Gaussian 的前后合成

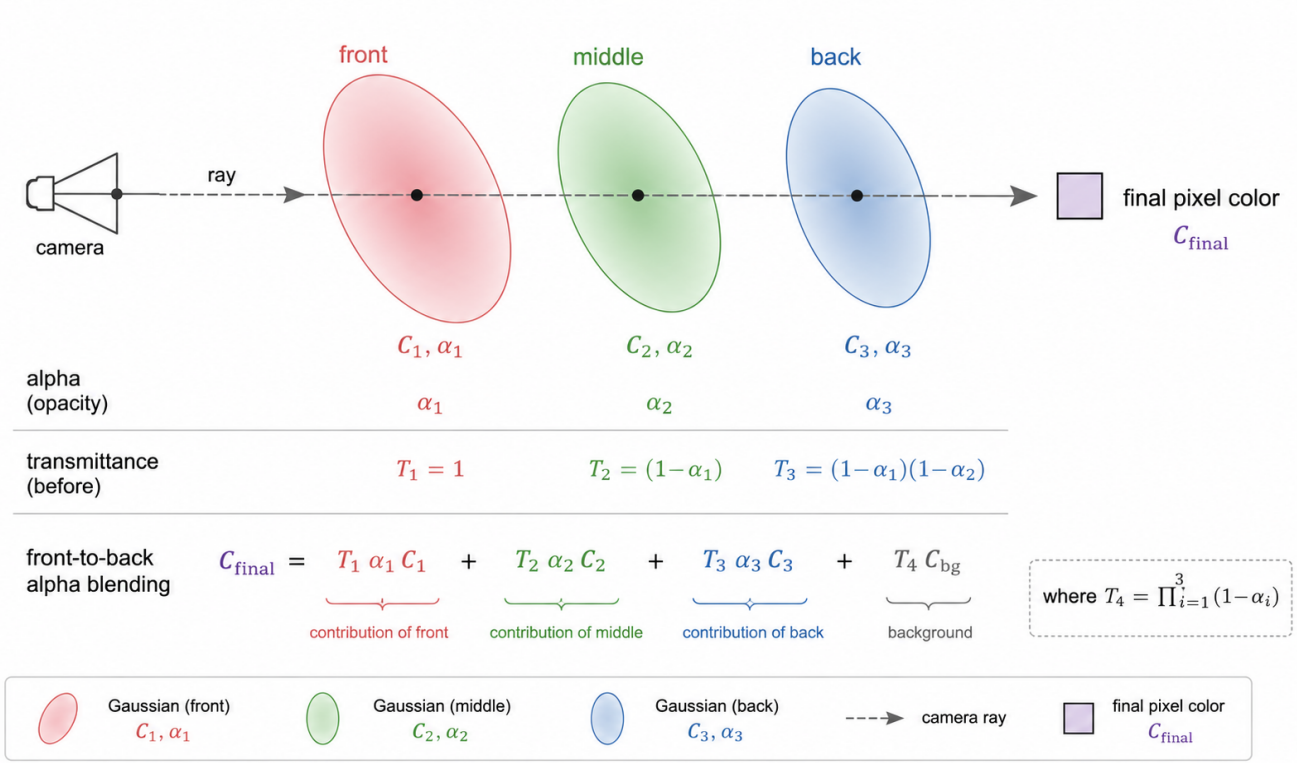
$$C(p) = \sum_{i=1}^{N} T_i(p)\alpha_i(p)c_i + T_{\mathrm{bg}}(p)c_{\mathrm{bg}}, \qquad T_i(p) = \prod_{j<i}(1-\alpha_j(p))$$

符号	直观含义
c_i	第 i 个 Gaussian 的颜色贡献
$\alpha_i(p)$	第 i 个 Gaussian 对像素 p 的像素级不透明度
$T_i(p)$	前面 Gaussian 透过之后，后面还能被看见的程度

- 多个 Gaussian 可能覆盖同一个像素，因此需要按深度 **front-to-back** 合成。
- Gaussian 自身学习的是 opacity；像素级 $\alpha_i(p)$ 还会乘上 2D footprint 上的权重。
- $T_{\mathrm{bg}}c_{\mathrm{bg}}$ 表示所有 Gaussian 之后剩余透射看到的背景颜色。

Alpha blending 解决的是“多个软元素叠在一起，一个像素到底是什么颜色”。

Alpha blending 的前后遮挡示意



Source: Generated teaching diagram

- 每个 Gaussian 的颜色先乘以当前像素级 $\alpha_i(p)$，再乘以前方剩余透射率 $T_i(p)$。
- 从前到后累积时，前面的透明度越高，后面的颜色贡献越小。

这张图对应公式中的 $T_i\alpha_i c_i$：颜色贡献同时取决于自身透明度和前方遮挡。

可微渲染：让渲染结果参与反向传播



- 核心要求很简单：渲染输出要能参与 loss，并把梯度传回参数。
- 神经网络训练也是这个逻辑；3DGS 的特殊之处在于可学习参数对应 Gaussian 的位置、尺度、旋转、透明度和颜色。
- 如果渲染器不可微，图像误差就无法告诉这些 Gaussian 参数如何调整。

可微渲染的作用就是把图像 loss 接到 Gaussian 参数上。

Part 3. 原始 3DGS 的表示与训练机制

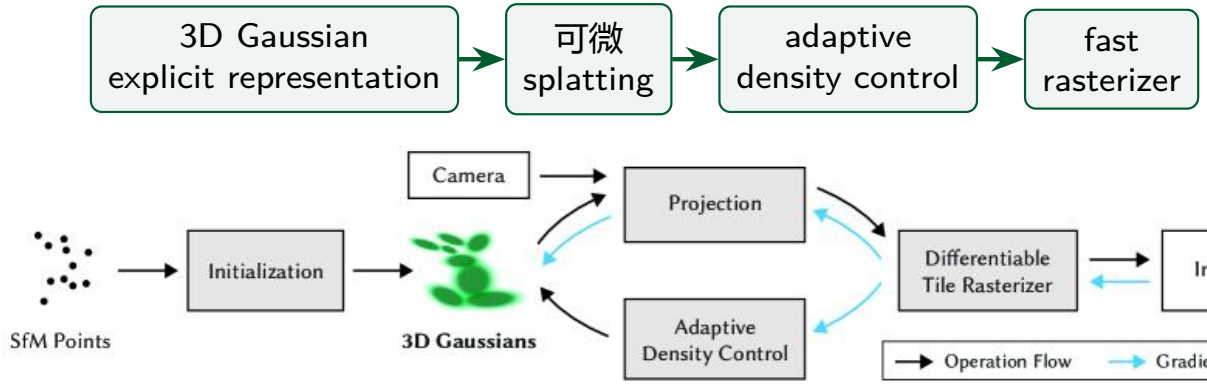
- 相机投影、footprint、alpha blending 和可微渲染构成理解原始方法的基础。
- 原始 3DGS 论文把这些环节组织成一个可训练系统。
- 主体问题依次是：Gaussian 存什么、怎么投影、怎么合成图像、怎么通过 loss 更新、怎么动态增删。

技术主体包括：3DGS 的显式 Gaussian representation、splatting renderer、training loss 和 density control。

原始论文的方法模块：表示、渲染、密度控制、加速

方法目标

3D Gaussian Splatting for Real-Time Radiance Field Rendering 的目标是：在保持高质量新视角合成的同时，实现实时渲染。



Source: Kerbl et al., 3DGS project page

- 表示与渲染：各向异性 Gaussian 经 projection 和 differentiable tile rasterizer 生成图像。
- 优化与结构调整：图像 loss 反传参数，adaptive density control 同时调整 Gaussian 分布。

原始论文的方法模块可以压成四块：Gaussian 表示、可微 splatting、密度控制和快速 rasterizer。

参考：Kerbl et al., 3D Gaussian Splatting project page

一个 Gaussian 的可学习参数

$$G_i = \{\mu_i, R_i, S_i, \alpha_i, \mathrm{SH}_i\}$$

参数	控制什么	为什么重要
中心 μ	三维位置	决定投影到哪里，是几何对齐的基础
尺度 S	三个方向的大小	贴合表面、边缘和细长结构
旋转 R	椭球方向	让 Gaussian 可以沿真实结构摆放
透明度 α	对像素的贡献强度	控制可见性、遮挡和 pruning
颜色/SH	外观与视角相关颜色	支持不同观察方向下的颜色变化

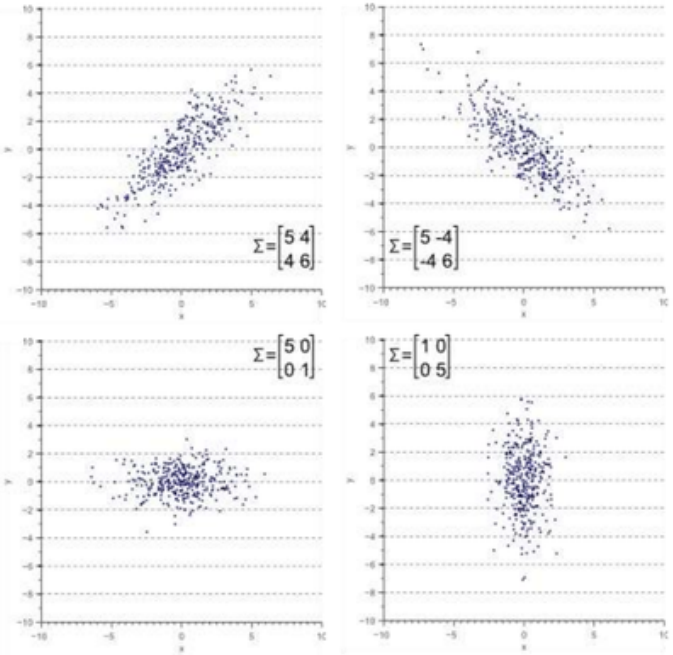
实现中常见字段包括 position、opacity、scale、quaternion 和 SH coefficients。可以把它理解成：点云点被升级成了“有方向、有范围、有视角相关颜色、有透明度”的局部可学习元素。

$G(x)$ 先用直觉理解：空间影响强度

它回答的问题

空间点 x 落在这个 Gaussian 的影响范围里有多深?

- $G(x)$ 描述位置受当前 Gaussian 影响的强度，后续还会参与 opacity、color 和 loss。
- 它只描述“这个位置受当前 Gaussian 影响多强”，在渲染中作为 **unnormalized influence weight** 使用。
- 协方差不同，影响区域的方向和拉伸形状也不同。
- 真正给像素上色时，还要再乘 opacity 和 color，并与其他 Gaussian 做 alpha blending。



Source: Shi Yan, How to Render a Single Gaussian Splat

先把 $G(x)$ 当作一个软权重：中心强、边缘弱，然后再看公式会更容易。

$G(x)$ 的公式：带形状的距离

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

部分	直观含义
$x-\mu$	当前点离 Gaussian 中心的偏移
Σ^{-1}	按 Gaussian 的形状重新衡量这个偏移
$(x-\mu)^T\Sigma^{-1}(x-\mu)$	带方向和尺度的距离：沿长轴偏一点不严重，沿短轴偏一点衰减快
$\exp(-\frac{1}{2}\cdot)$	把距离变成 0 到 1 附近的软影响强度

- 协方差 Σ 决定小云团是圆的、扁的还是长条的。
- 所以同样离中心一段距离，沿不同方向的影响强度可能不同。

公式真正表达的是“带形状的距离越小，Gaussian 的影响越强”。

参考：Shi, How to Render a Single Gaussian Splat?; Scthe, Rendering in 3DGS

从一维高斯到三维椭球

一维高斯	**三维 Gaussian**
$G(x) = \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)$	$G(x) = \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$
μ 是中心 σ 控制曲线宽窄 离中心越远值越小	μ 是三维小云团中心 Σ 控制大小、扁平程度和方向 沿长轴衰减慢，沿短轴衰减快

- 普通距离只问“离中心多远”。
- $(x-\mu)^T\Sigma^{-1}(x-\mu)$ 是**带方向和尺度的距离**。
- 因此 3DGS 的 Gaussian 可以贴住墙面、桌面、边缘和细长结构。

把 σ^2 换成 Σ 后，高斯从钟形曲线变成了可旋转、可拉伸的三维椭球。

协方差构造：$\Sigma = RSS^T R^T$

- 协方差矩阵 Σ 必须是合法的，通常要保持对称和半正定。
- 如果直接优化矩阵元素，梯度更新后可能得到无效 covariance。
- 原始 3DGS 因此用旋转 R 和缩放 S 构造协方差：

$$\Sigma = RSS^T R^T, \qquad S = \mathrm{diag}(\exp(s_x), \exp(s_y), \exp(s_z))$$

- 这里 S 是**正尺度对角矩阵**，可以理解为三个主轴方向上的标准差尺度；因为它是对角矩阵，

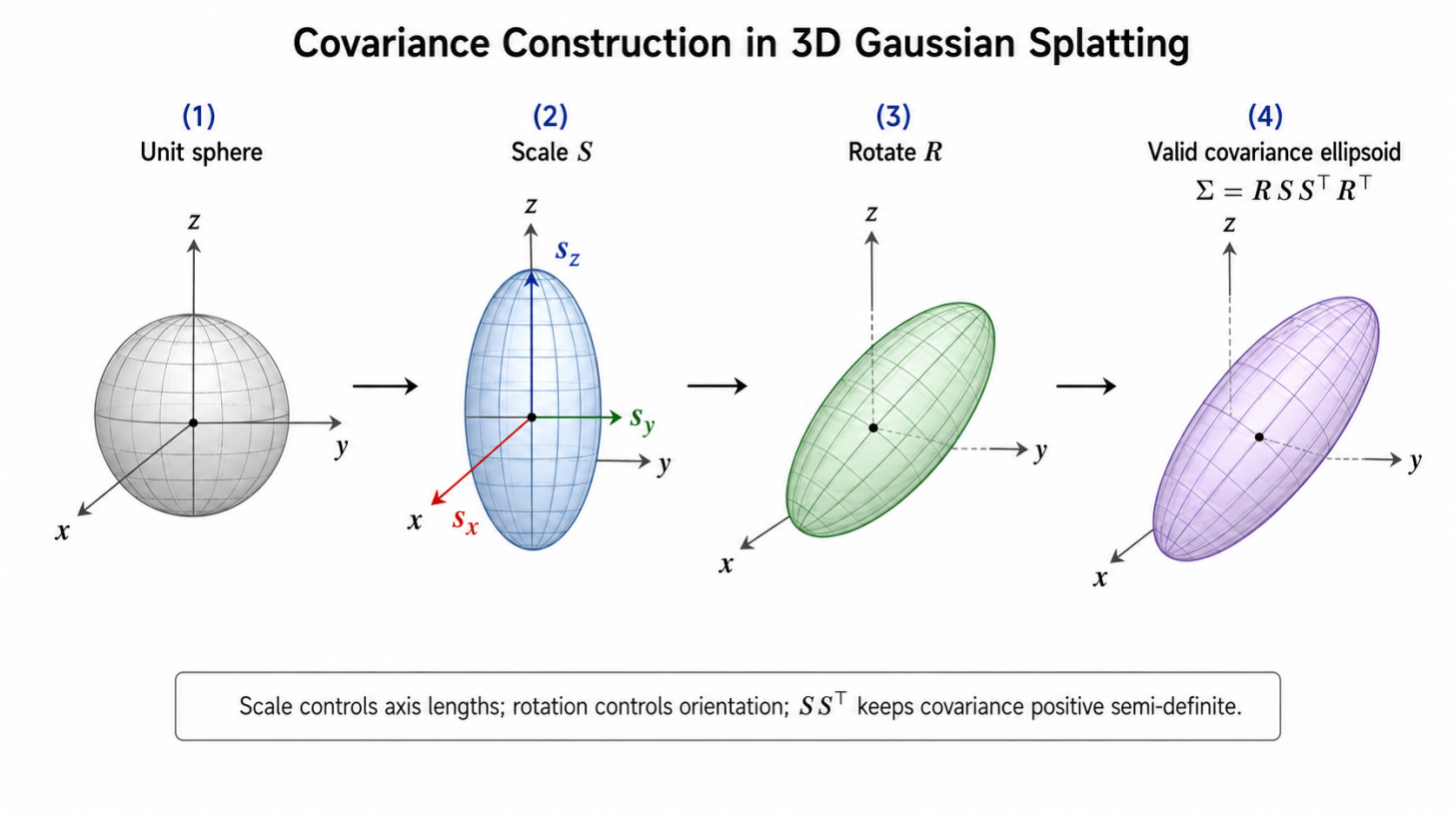
$$SS^T = \mathrm{diag}(\exp(2s_x), \exp(2s_y), \exp(2s_z))$$

  也就是常见写法里的 S^2。
- 这样每次更新后仍然对应一个合理椭球：
  - S 控制三个主轴有多长，并通过正尺度参数化避免负轴长；
  - R 控制椭球朝哪个方向；
  - SS^T 保证尺度部分是合法正半定结构。

用 R 和 S 就像先保证它永远是合法椭球，再调长短轴和朝向。

参考：Scthe, Rendering in 3DGS

协方差构造的几何直觉



Source: Generated teaching diagram

- S 把单位球沿三个主轴拉伸，决定 Gaussian 的长短轴。
- R 再把椭球旋转到局部表面或边缘方向，得到各向异性的 covariance ellipsoid。

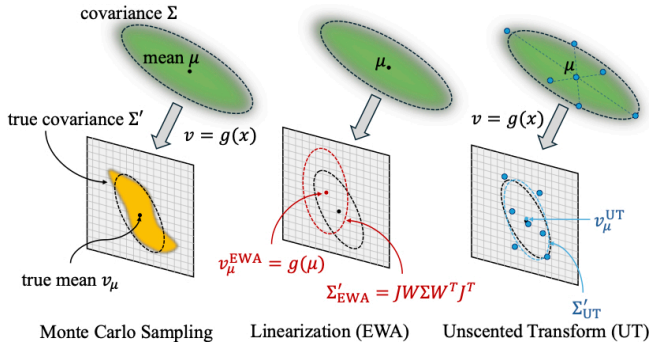
$\Sigma = RSS^T R^T$ 的直觉是：先形成合法尺度，再旋转到场景结构需要的方向。

3D Gaussian 到 2D Gaussian 的投影

约定：μ_w, Σ_w 在世界坐标；$\mu_c = (x, y, z)^T$ 在相机坐标；R_{cw}, t_{cw} 表示 world-to-camera。

中心投影

$$\mu_c = R_{cw}\mu_w + t_{cw}, \quad m = \begin{bmatrix} f_x \frac{x}{z} + c_x \\ f_y \frac{y}{z} + c_y \end{bmatrix}$$



Source: NVIDIA 3DGUT

协方差投影

$$\Sigma_c = R_{cw}\Sigma_w R_{cw}^T, \quad \Sigma_{2D} = J\Sigma_c J^T$$

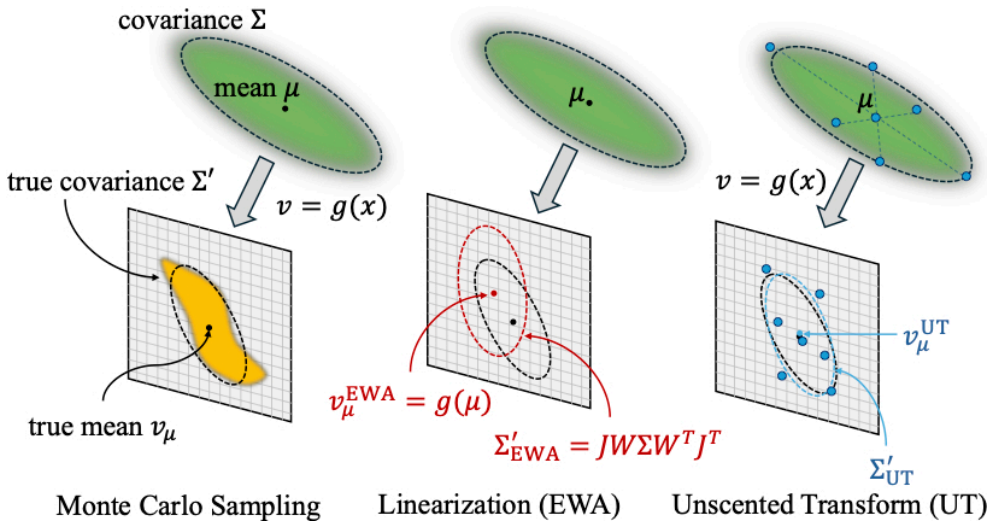
$$J = \begin{bmatrix} \frac{f_x}{z} & 0 & -\frac{f_x x}{z^2} \\ 0 & \frac{f_y}{z} & -\frac{f_y y}{z^2} \end{bmatrix}$$

- J 是投影函数在 μ_c 附近的雅可比。
- 这是局部线性化近似。

中心变成 2D 中心，三维协方差通过投影雅可比变成二维椭圆 footprint。

投影雅可比 J 的局部线性近似

$$u = f_x \frac{x}{z} + c_x, \qquad v = f_y \frac{y}{z} + c_y$$

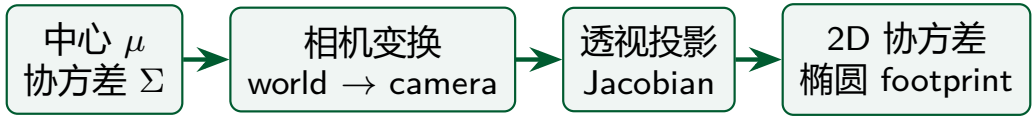


Source: NVIDIA 3DGUT

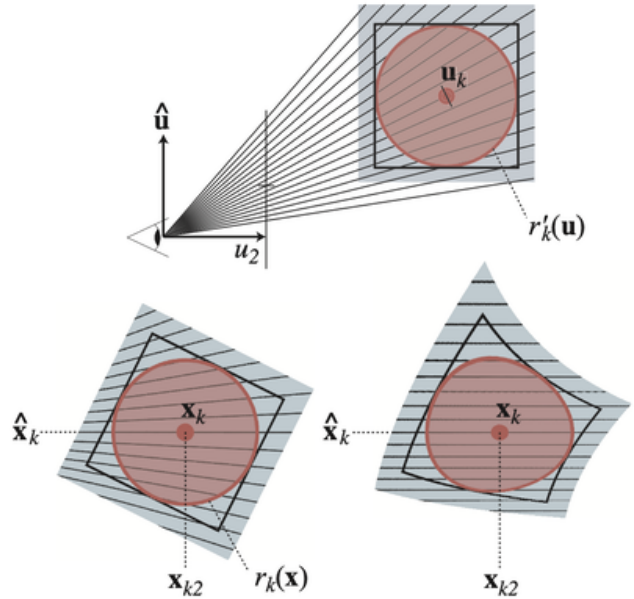
- 透视投影是非线性的；J 是投影函数在 Gaussian 中心附近的一阶近似。
- 它回答三维空间里沿某个方向移动一点，图像上的 (u, v) 会怎么变。
- 深度会强烈影响变化率：z 越小，同样三维偏移带来的像素位移越大。

J 把“三维形状的微小变化”翻译成“图像平面上的椭圆变化”。

从 3D Gaussian 到 2D footprint



- Gaussian 的投影需要同时处理中心点和三维形状，把空间范围变成二维 footprint。
- 离相机近的 Gaussian footprint 更大；形状和视角也会改变椭圆方向。
- 工程实现会由 2D covariance 推出主要影响范围，再转成屏幕 bbox 或 tile overlap。



Source: Shi Yan, How to Render a Single Gaussian Splat

3DGS 渲染关注投影后的椭圆区域，普通点云里的像素点只提供位置对照。

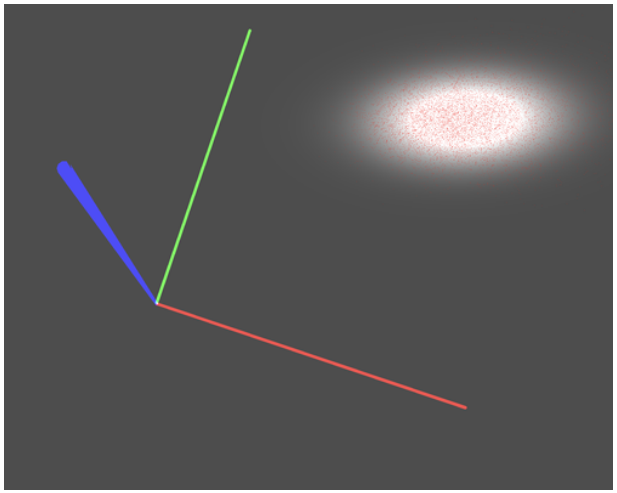
像素级 Gaussian 贡献

投影后，一个 Gaussian 有 2D 中心 m_i、2D 协方差 $\Sigma_{2D,i}$、透明度 α_i 和颜色 c_i。

$$p = \begin{bmatrix} u \\ v \end{bmatrix}$$

$$w_i(p) = \exp\left(-\frac{1}{2}(p - m_i)^T \Sigma_{2D,i}^{-1}(p - m_i)\right)$$

$$\alpha_i(p) = \alpha_i \cdot w_i(p)$$



Source: Shi Yan, How to Render a Single Gaussian Splat

- 像素越靠近 2D Gaussian 中心，$w_i(p)$ 越大。
- 离 footprint 中心越远，贡献越弱。
- $\alpha_i(p)$ 是乘上 footprint 权重后的像素级 alpha。

Gaussian 的颜色贡献集中在二维 footprint 附近，并呈现中心强、边缘弱的软边界。

3DGS 的像素合成公式

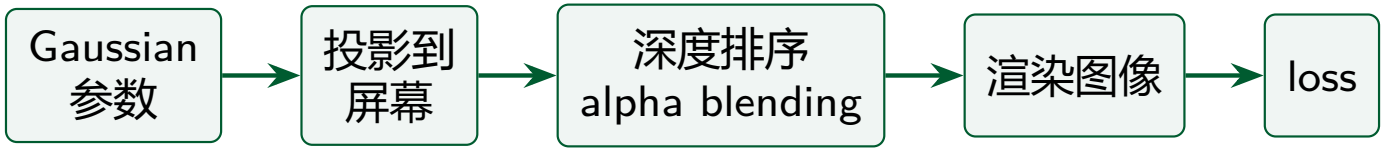
$$C(p) = \sum_{i=1}^{N} T_i(p)\alpha_i(p)c_i, \qquad T_i(p) = \prod_{j=1}^{i-1}(1 - \alpha_j(p))$$

方法	哪些元素影响像素	共同直觉
Ray-based volumetric rendering	沿 ray 采样连续密度场	前面密度/透明度会遮挡后面颜色
3DGS	投影离散 Gaussian 并按深度排序	颜色和透明度从前到后累积

- 先按 3DGS 理解：投影到同一像素的 Gaussian 按深度排序，逐个贡献颜色并削弱后方可见性。
- 和 ray-based volumetric rendering 的共同点是“前面遮挡后面、颜色逐层累积”。
- 两者的计算路径不同：3DGS 的像素影响来自显式投影和 splatting；ray-based 方法来自沿 ray 的 dense sampling。

3DGS 是 splatting/rasterization 组织方式，但像素颜色仍按可见性做透明度累积。

可微 splatting：渲染路径也是训练路径

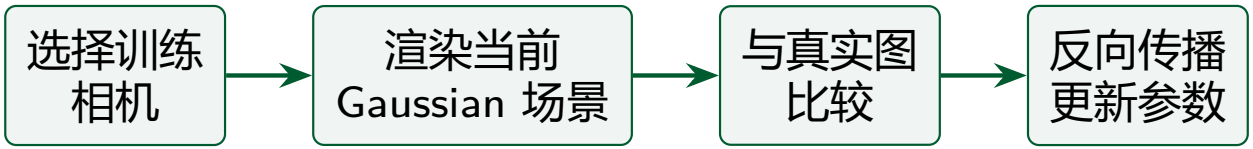


- 前向过程：从 Gaussian 参数生成图像。
- 反向过程：图像误差回传到位置、尺度、旋转、透明度和颜色。
- 连续参数到像素颜色的路径可反传；排序、culling 和 tile assignment 通常作为渲染组织处理，核心 rasterizer 会实现自定义 backward。
- 如果渲染边缘错位，位置和形状会被调整；如果颜色偏暗，透明度和颜色会被调整。

3DGS 的训练沿着同一条渲染路径回传梯度，前向渲染和反向更新在同一系统中完成。

训练目标：让渲染图像接近真实照片

$$\mathcal{L} = (1 - \lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{D-SSIM}}$$



- $\mathcal{L}_1$ 负责像素层面的颜色差异。
- D-SSIM 让结构相似性也参与监督。
- 原始论文中常用 $\lambda = 0.2$，体现像素误差和结构相似性的折中。
- 训练的结果是一组更能解释所有训练视角的 Gaussian。

3DGS 仍然是图像监督，但优化对象从网络函数变成了显式 Gaussian 集合。

梯度回传到 Gaussian 参数

$$\hat{I} = Render(\{\mu_i, R_i, S_i, \alpha_i, c_i\}_{i=1}^{N}), \qquad L = L(\hat{I}, I)$$

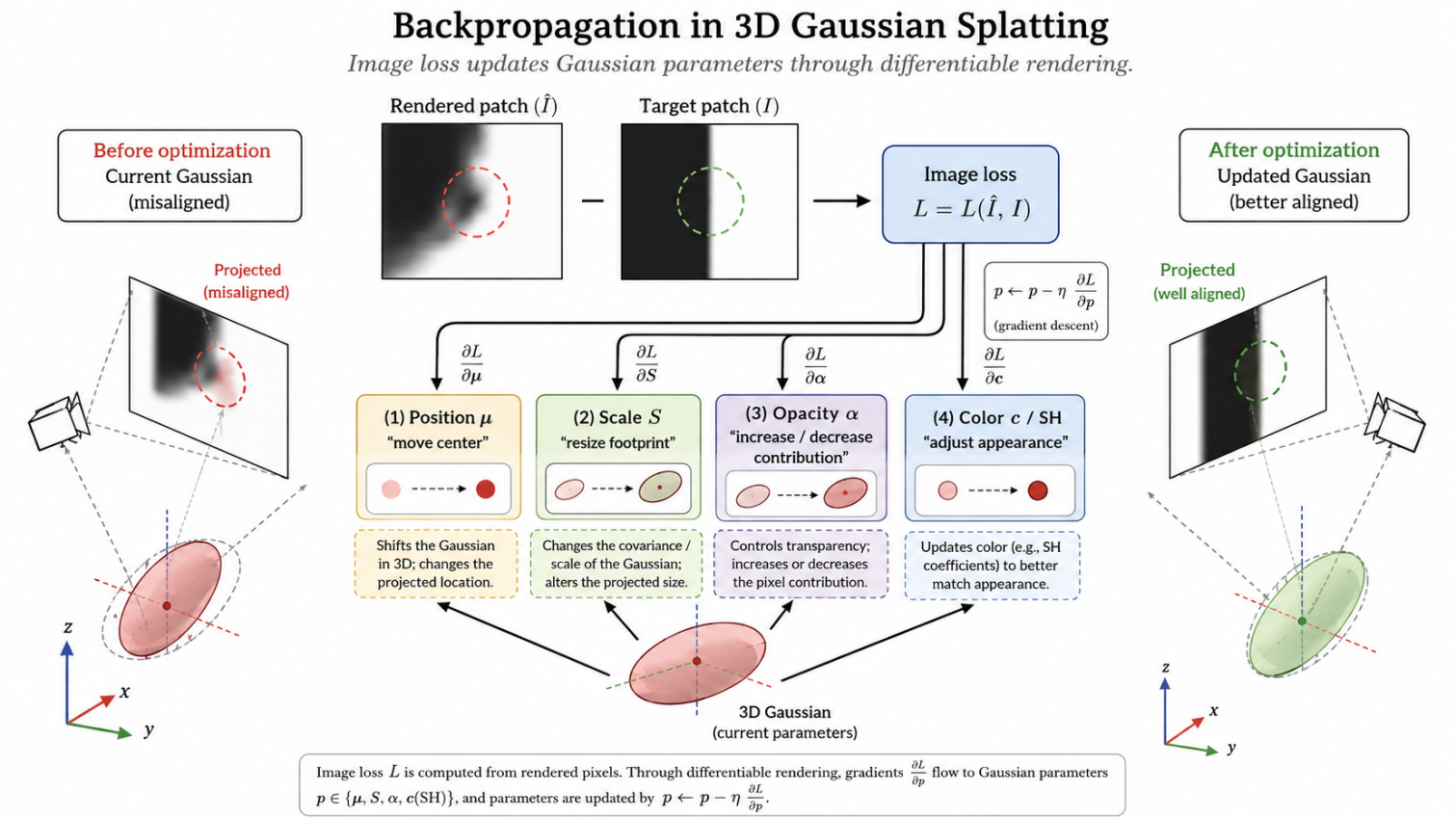
$$\frac{\partial L}{\partial p_i} = \frac{\partial L}{\partial \hat{I}} \frac{\partial \hat{I}}{\partial p_i}, \qquad p_i \leftarrow p_i - \eta \frac{\partial L}{\partial p_i}$$

参数	梯度会让它怎样变
μ_i	边缘或结构错位时，位置沿能降低 loss 的方向移动
S_i	细节太糊可能缩小，覆盖不足可能变大
R_i	椭球朝向会调整到更贴合局部表面或边缘
α_i	有用区域变强；错误或无贡献区域变透明，后续可能被删
c_i/SH	颜色偏暗、偏亮或偏色时调整外观参数

$$\mu_i \to m_i \to w_i(p) \to \alpha_i(p) \to C(p) \to L$$

反向传播通过链式法则计算参数变化对图像误差的影响。

图像误差如何更新 Gaussian



Source: Generated teaching diagram

- 图像 loss 通过可微渲染路径回到 Gaussian 参数。
- 位置、尺度、透明度和颜色分别对应“对齐结构、调整 footprint、改变贡献强度、修正外观”。

教学代码：一个 2D Gaussian 如何被优化

在 $G(x)$、footprint、alpha blending 和 loss/backward 的公式之后，这段代码作为最小教学例子，帮助理解“图像误差如何更新 Gaussian 参数”。

```
import torch
import torch.nn.functional as F

H, W = 32, 32
yy, xx = torch.meshgrid(torch.linspace(-1, 1, H),
                        torch.linspace(-1, 1, W), indexing="ij")
grid = torch.stack([xx, yy], dim=-1)

target = torch.zeros(H, W, 3)
target[..., 0] = torch.exp(-((xx + 0.35) ** 2 + yy ** 2) / 0.03)

mu = torch.nn.Parameter(torch.tensor([0.4, 0.2]))
log_scale = torch.nn.Parameter(torch.tensor([-1.5, -1.5]))
opacity_logit = torch.nn.Parameter(torch.tensor(0.0))
color_logit = torch.nn.Parameter(torch.tensor([0.0, -2.0, -2.0]))
optimizer = torch.optim.Adam([mu, log_scale, opacity_logit, color_logit], lr=0.05)
```

教学代码：loss.backward() 在更新什么？

```
for step in range(300):
    optimizer.zero_grad()

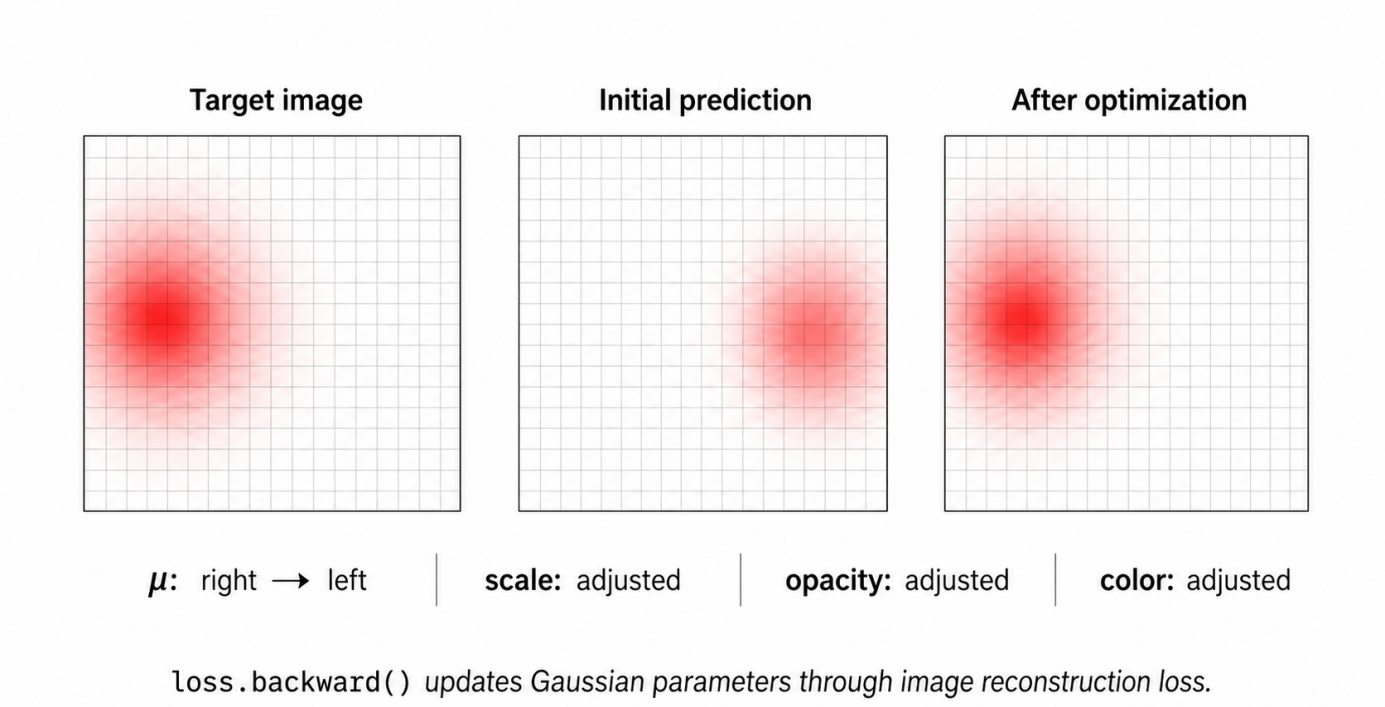
    scale = torch.exp(log_scale) + 1e-6
    diff = (grid - mu) / scale
    G = torch.exp(-0.5 * (diff ** 2).sum(dim=-1))

    alpha = torch.sigmoid(opacity_logit) * G
    color = torch.sigmoid(color_logit)
    pred = alpha[..., None] * color[None, None, :]

    loss = F.mse_loss(pred, target)
    loss.backward()
    optimizer.step()
```

- 初始 Gaussian 在右侧，目标红色软圆点在左侧。
- 反向传播会同时更新位置、尺度、透明度和颜色。
- 完整 3DGS 把这个思想扩展到三维，并用可微 rasterizer 处理大量 Gaussian。

教学代码运行结果：2D Gaussian 被 loss 拉回目标

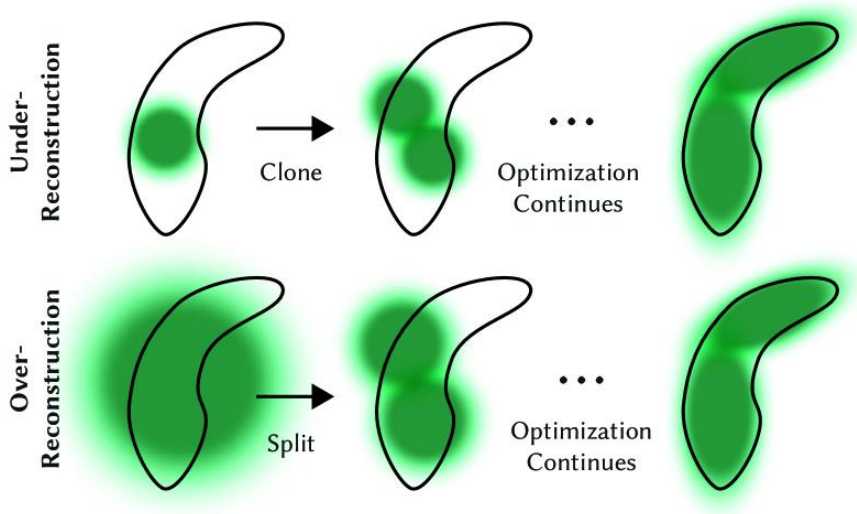


Source: Generated teaching diagram

- 目标 blob 在左侧，初始预测在右侧；优化后中心 μ、scale、opacity 和 color 一起被更新。
- 完整 3DGS 的差别是从一个 2D Gaussian 扩展到三维空间中的大量 Gaussian。

Adaptive density control：Gaussian 数量也在学习

只优化已有 Gaussian 通常不够，因为初始表示可能粗糙、不均匀。原始 3DGS 会交替进行参数优化和密度控制。

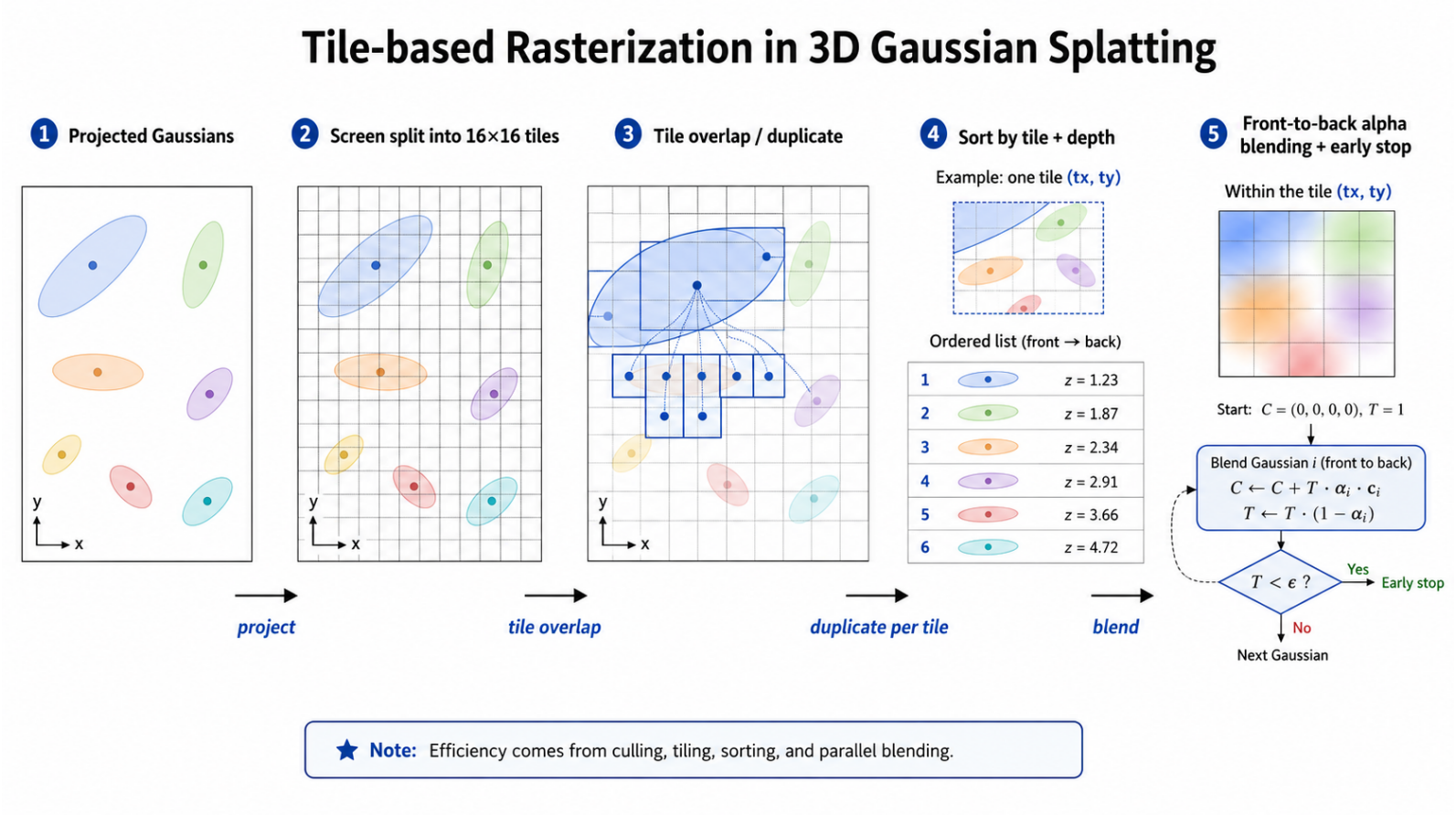


Source: Kerbl et al., 3DGS project page

- **clone：** view-space positional gradient 大且 Gaussian 较小，用于补欠重建细节。
- **split：** gradient 大且 Gaussian 尺度偏大，把过大的元素拆成更小结构。
- **prune / opacity reset：** 删除贡献小或低 opacity 的冗余元素，并避免可见性过早固化。

通过增删和拆分 Gaussian，模型不只是调整参数，也在训练中逐步调整场景结构。

Fast rasterizer：tile-based 并行组织



Source: Generated teaching diagram

- 每个 tile 只处理可能覆盖自己的 Gaussian，并按 tile 与 depth 排序。
- front-to-back blending 可以在透射率足够低时 early stop，从而避免 Gaussian × pixel 的暴力比较。

实时性能来自 culling、tiling、sorting 和并行 alpha blending 的组织方式。

训练结果：带属性的 Gaussian PLY

- 3DGS 训练完得到的是一组 Gaussian 参数，可继续从不同视角渲染，区别于单张图像或传统 mesh。
- 常见保存格式可以是 ‘.ply‘，但里面不只是普通点云坐标。
- 一个 Gaussian 可能保存：
  - position：`x,y,z`;
  - opacity：常见字段名如 `opacity`;
  - scale：常见字段名如 `scale_0, scale_1, scale_2`;
  - rotation / quaternion：常见字段名如 `rot_0,...,rot_3`;
  - spherical harmonics color coefficients：常见字段名如 `f_dc_*` 和 `f_rest_*`;
  - 其他实现相关属性。
- 所以每一行更像一个“可投影、可混合、可学习的小云团参数表”。

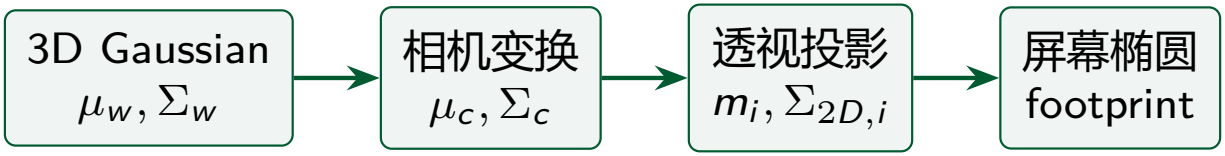
3DGS 的显式性，也体现在训练结果可以被保存、查看、筛选和继续编辑。

参考：Evolve Benchmark 3DGS rendering note

- 前面逐步讲了一个 Gaussian 的表示、投影、像素贡献和反向优化。
- 公式链路可以合到两页总览中，形成答疑时可以快速回看的路径。
- 实验部分主要看原始论文如何证明：质量、速度和训练可行性可以同时成立。

散开的公式可以连成完整路径：3D Gaussian 到 2D image supervision。

3DGS 数学链路总览（上）：先得到 2D footprint



$$\mu_c = R_{cw}\mu_w + t_{cw}, \qquad \Sigma_c = R_{cw}\Sigma_w R_{cw}^T$$

$$m_i = \begin{bmatrix} f_x \frac{x}{z} + c_x \\ f_y \frac{y}{z} + c_y \end{bmatrix}, \qquad \Sigma_{2D,i} = J\Sigma_c J^T$$

- 这半条链路只回答：一个 3D Gaussian 从当前相机看，会落到图像哪里、变成多大的椭圆。
- 中心决定 footprint 的位置，协方差决定 footprint 的方向和范围。

先得到 2D footprint，再根据 footprint 计算它对像素颜色的贡献。

3DGS 数学链路总览（下）：再合成像素并反传

$$w_i(p) = \exp\left(-\frac{1}{2}(p-m_i)^T\Sigma_{2D,i}^{-1}(p-m_i)\right), \qquad \alpha_i(p) = \alpha_i w_i(p)$$

$$C(p) = \sum_i T_i(p)\alpha_i(p)c_i, \qquad T_i(p) = \prod_{j<i}(1-\alpha_j(p))$$

$$L = (1-\lambda)L_1 + \lambda L_{D\text{-}SSIM}, \qquad p_i \leftarrow p_i - \eta\frac{\partial L}{\partial p_i}$$

$$p_i \in \{\mu_i, S_i, R_i, \alpha_i, c_i\}$$

- $w_i(p)$ 说明像素 p 在这个 2D footprint 里的影响强度。
- alpha blending 把多个 Gaussian 的贡献按深度和透明度合成。
- loss 再沿着同一条渲染链路反传，更新位置、尺度、旋转、透明度和颜色。

这样看，3DGS 的训练就是把二维图像误差沿渲染链路传回显式三维参数。

原始论文实验：评价维度

实验维度	看什么指标	原始论文要证明什么
质量	PSNR / SSIM / LPIPS	在 Mip-NeRF360、Tanks&Temples、Deep Blending 等数据上达到接近或超过当时代表方法的视觉质量
速度	FPS / 1080p rendering	论文强调 1080p 下达到实时新视角合成，核心数量级是 ≥ 30 FPS
训练和表示代价	training time / memory / storage	显式 Gaussian 数量可控，训练时间保持在可接受范围，同时改善渲染速度

- InstantNGP、Plenoxels、Mip-NeRF360 等方法主要作为质量和速度基线。
- 实验页真正要看的是三件事是否同时成立：图像质量、实时渲染、可训练的显式表示。

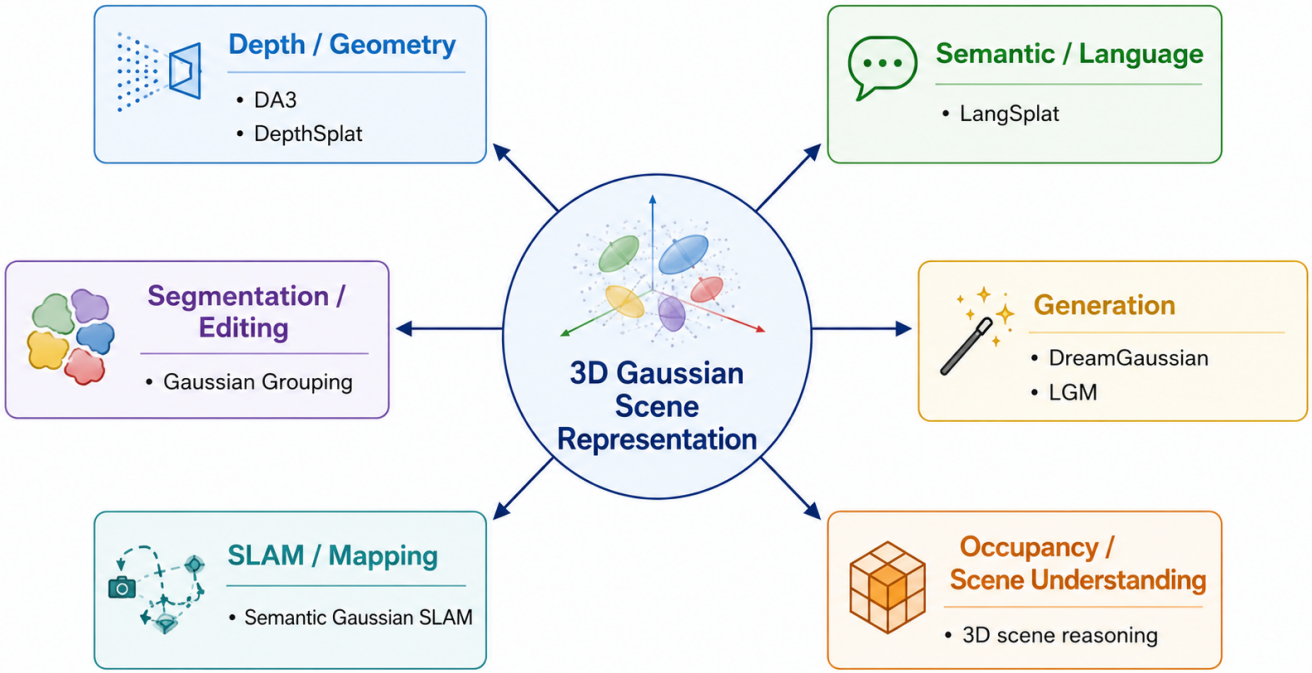
原始论文的说服力在于：显式 Gaussian + 可微 splatting 没有只换速度，而是把质量和实时性同时拉到可用区间。

参考：3D Gaussian Splatting official project page

原始 3DGS 的局限与后续扩展

- 几何不一定干净：新视角图像好看，不代表 depth、normal 或 mesh 表面都准确；例如可能有 floaters、孔洞或表面噪声。
- 对输入和初始化敏感：少视角、弱纹理、反光区域会更难。
- 动态场景更复杂：原始方法主要面向静态场景。
- 语义和对象结构缺失：原始 Gaussian 主要存几何和外观，不知道“这是什么物体”。
- 每场景优化有成本：大规模生成和在线应用希望更快预测 Gaussian。

后续研究沿着这条思路，把 Gaussian 从 rendering representation 推向更强的 space representation。

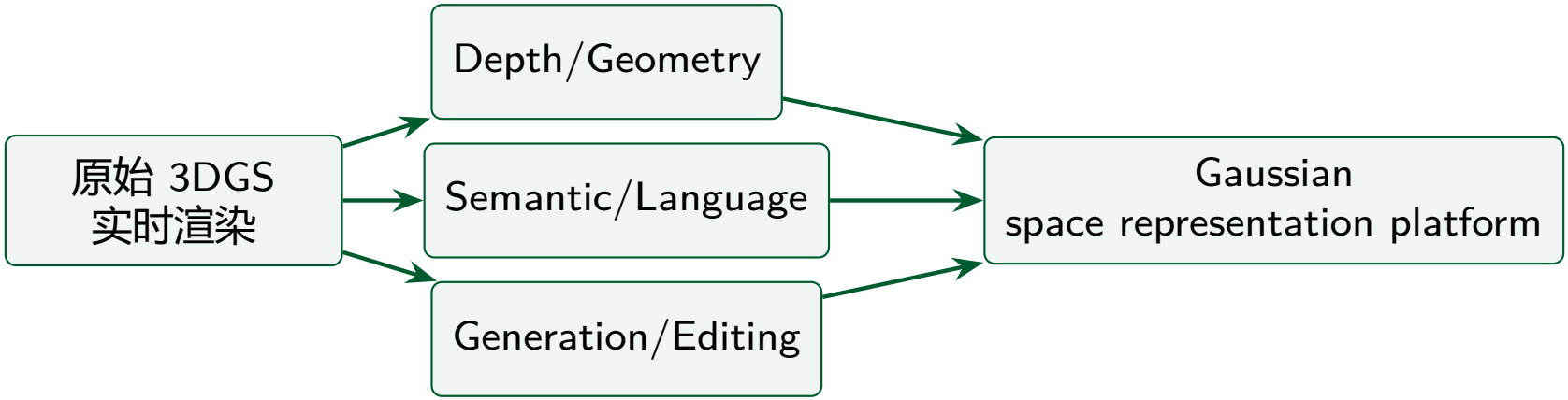


Source: Generated teaching diagram

- 原始 3DGS 证明 Gaussian primitives 可以用于高质量实时渲染。
- 后续工作沿着同一个显式表示继续扩展到几何、语义、语言、编辑、生成和地图。
- 每个方向都可以回到一个问题：Gaussian 上新增了什么信息，或者替换了哪一个训练环节。

应用发展关注 3DGS 如何从 rendering primitive 扩展为空间信息载体。

后续工作：从 rendering primitive 到空间表示



- 第一阶段：把照片重建成能实时渲染的 Gaussian 场景。
- 第二阶段：让 Gaussian 承载更多属性，例如深度、语义、语言、对象 ID、生成先验。
- 第三阶段：从每场景优化走向快速预测、在线建图和空间理解。

3DGS 的学术脉络可以理解为：Gaussian 从渲染 primitive 变成空间信息载体。

从 rendering primitive 到 space representation platform

代表方向	怎么用 3DGS	改造模块	说明的趋势
DA3 / DepthSplat	几何/深度辅助 Gaussian 估计	initialization / predictor	几何基础模型进入 Gaussian 预测与优化流程
LangSplat	给 Gaussian 加语言特征	attributes / feature rendering	从“能看”到“能查询”
Gaussian Grouping	给 Gaussian 加 identity	attributes / grouping loss	显式元素可分割和编辑
DreamGaussian / LGM	用 Gaussian 做生成表示	generator / optimization path	从逐场景优化到快速生成
Semantic Gaussian SLAM	Gaussian map 携带语义特征	map update / tracking / features	可渲染、可理解地图

读应用论文时不要只看任务名，要看它到底给 Gaussian 增加了什么属性，或改了哪段训练/预测路径。

几何方向：DA3 / DepthSplat 如何服务 3DGS

- **DA3：**用强几何基础模型预测 depth、pose 或 Gaussian 相关条件，服务 initialization、几何约束和 feed-forward reconstruction。
- **DepthSplat：**把预训练 monocular depth features 融入 multi-view depth model，再服务 feed-forward 3D Gaussian reconstruction。
- **和原始 3DGS 的关系：**原始 3DGS 主要是每场景优化；这类工作把更强的 depth/geometry prior 接入 Gaussian 生成或优化。
- **说明的趋势：**3DGS 的发展不只追求图像质量，也在补少视角、弱初始化和未观察区域的几何稳定性。

参考：Depth Anything 3 project page; DepthSplat

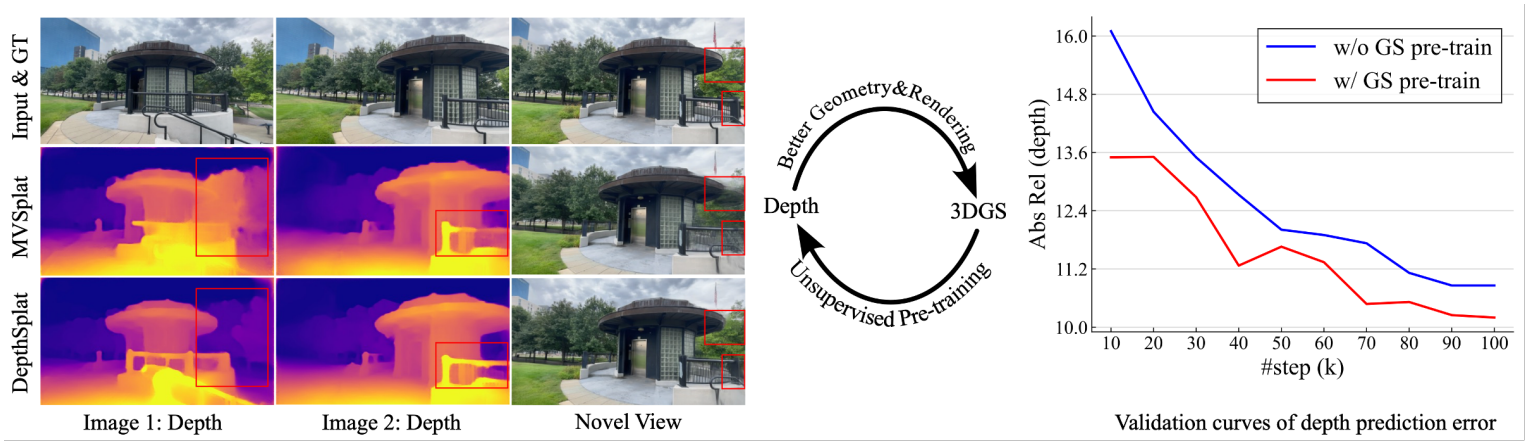
DA3：几何基础模型作为 3DGS 入口



Source: Depth Anything 3 project page

- DA3 的项目图强调 depth、ray maps、pose/geometry accuracy 可以作为 3DGS rendering 和 Gaussian 参数估计的几何入口。
- 这类工作把原始 3DGS 对相机和初始化的依赖，转化为更强的 geometry prior。

DepthSplat：深度与 3DGS 的双向促进



Source: DepthSplat project / paper figure

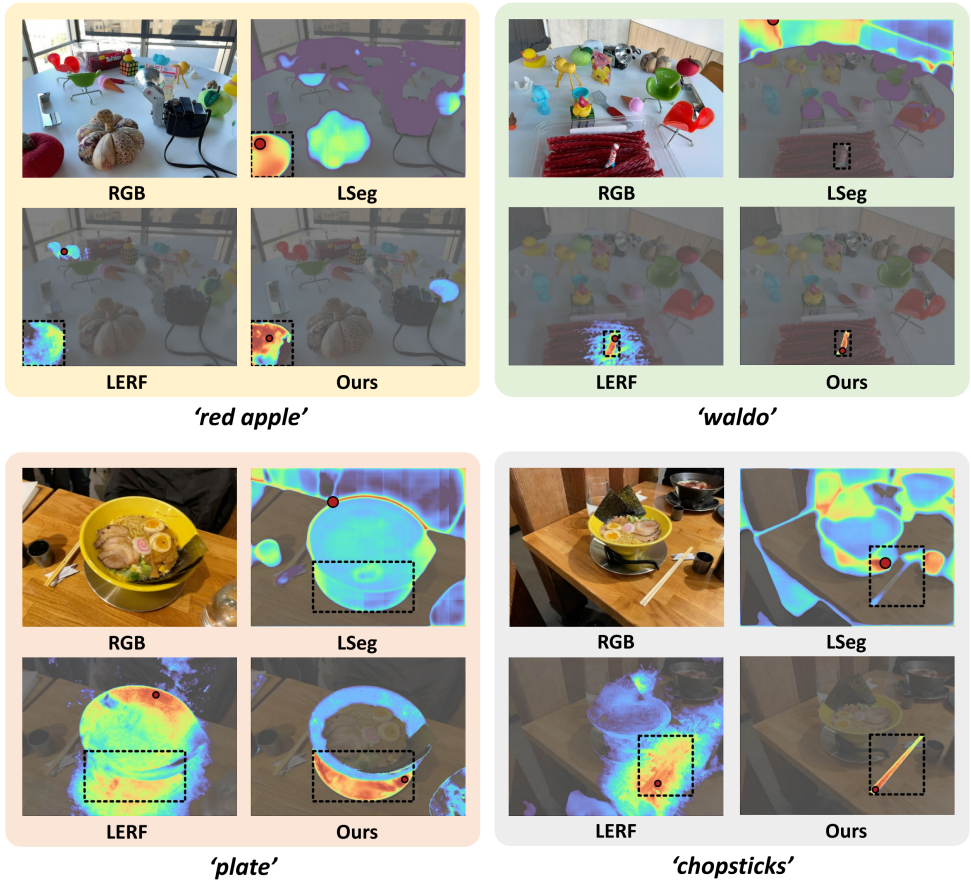
- 图中一侧是 multi-view depth 与 novel view，另一侧展示 depth 与 3DGS pre-training 的互相促进。
- 这对应应用页里的主线：几何先验帮助 Gaussian reconstruction，Gaussian rendering 也能反过来改善几何学习。

语义与编辑：LangSplat / Gaussian Grouping

- **LangSplat：**在 Gaussian attributes 上增加 language / CLIP-like features，让 3D 场景支持 open-vocabulary text querying。
- **Gaussian Grouping：**给每个 Gaussian 增加 Identity Encoding，用 2D mask supervision 和 3D consistency 把分割知识提升到 3D。
- **和原始 3DGS 的关系：**渲染机制基本保留，但 Gaussian 不再只存颜色和透明度，还能存语义、语言或对象 ID。
- **说明的趋势：**显式 Gaussian 元素可以被查询、分组、删除、重组和编辑。

参考：LangSplat; Gaussian Grouping

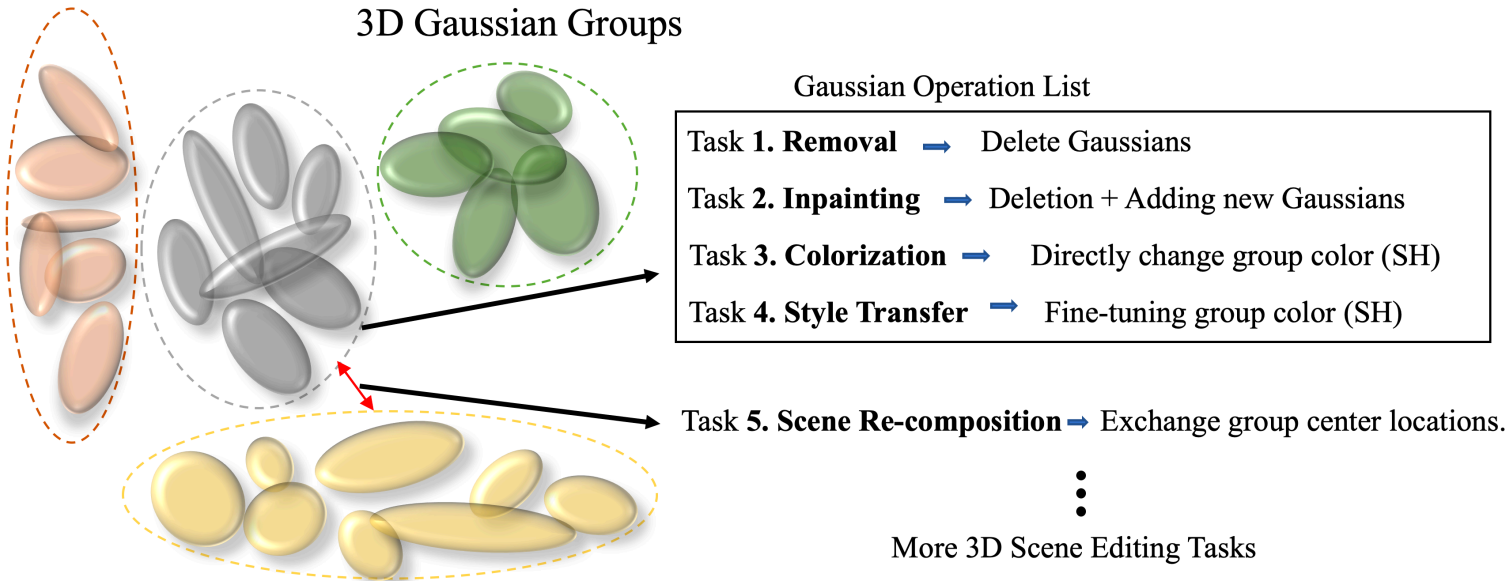
LangSplat：Gaussian 上的开放词汇查询



Source: LangSplat project page

- 图中 text query 例如 “red apple”、“waldo” 被映射到 3D 场景中的语言响应。
- 这说明 Gaussian attributes 可以从颜色/透明度扩展到 language feature field。

Gaussian Grouping：对象级分组与编辑



Source: Gaussian Grouping project page

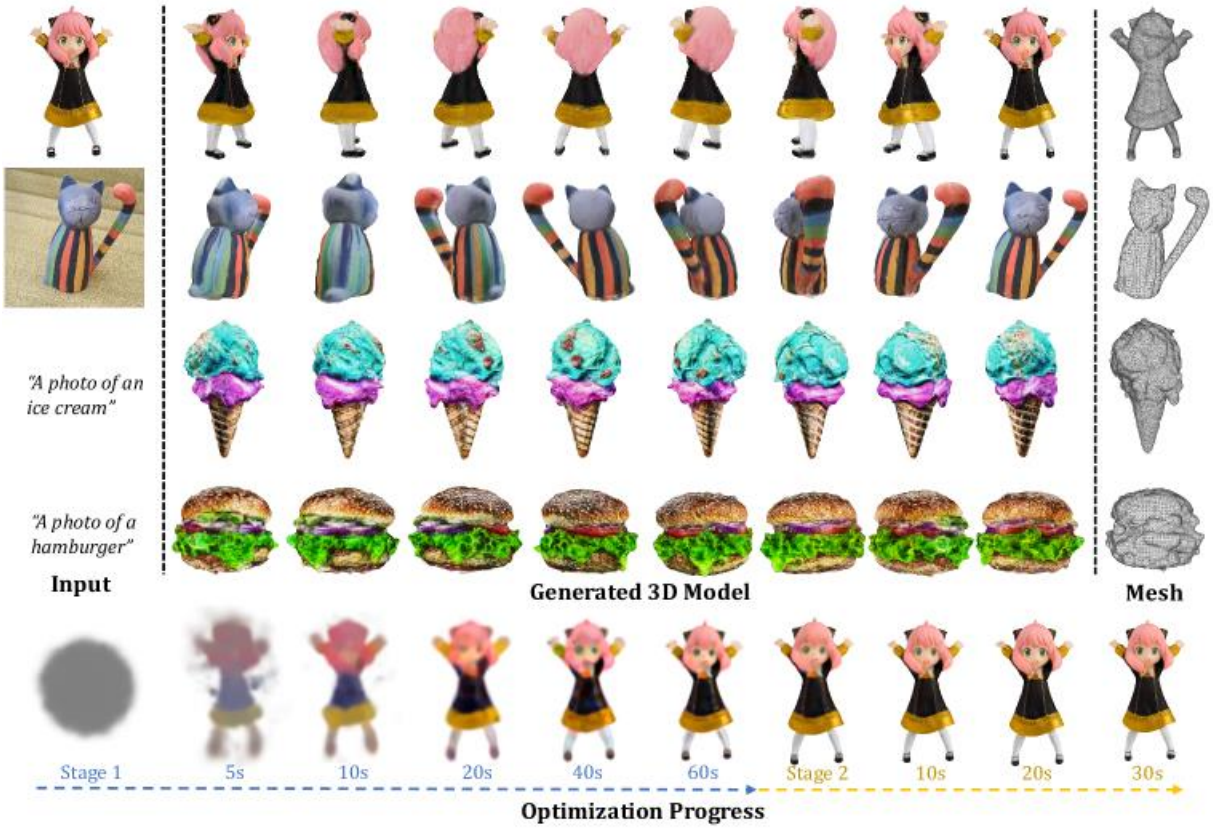
- Gaussian Grouping 给显式 Gaussian 增加 identity / group 信息。
- 一旦 Gaussian 被组织成对象组，就可以支持 removal、inpainting、colorization、style transfer 和 recomposition。

生成方向：DreamGaussian / LGM

- **DreamGaussian：**把 Gaussian 用作 SDS 等生成优化中的快速 3D intermediate representation，先形成粗几何和外观，再转 textured mesh refinement。
- **LGM：**从文本、单图或多视图条件直接预测高分辨率 Gaussian，代表 feed-forward 生成方向。
- **和原始 3DGS 的关系：**原始方法重建真实场景；生成方向把 Gaussian 变成 3D 内容生产中的中间表示或网络输出格式。
- **说明的趋势：**Gaussian 从 per-scene optimization 结果，扩展成可由生成模型直接预测的显式 3D representation。

参考：DreamGaussian; LGM

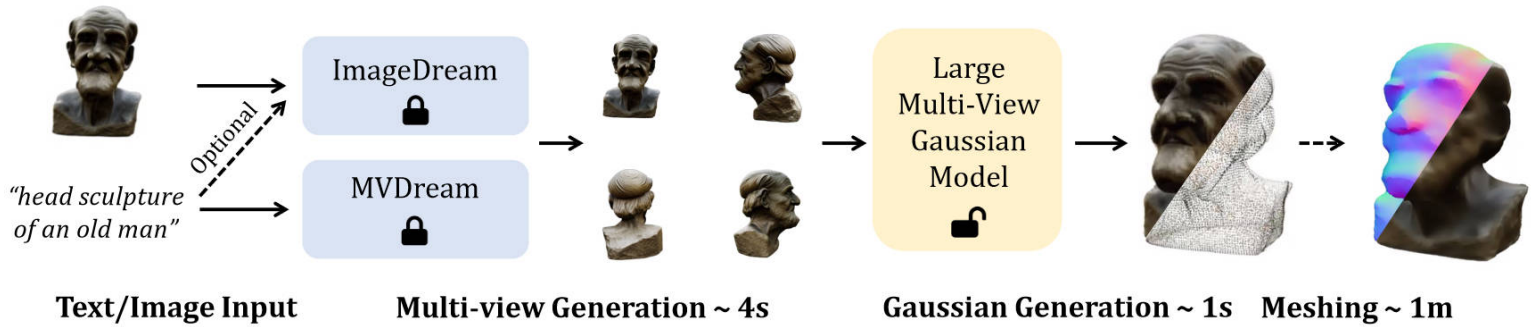
DreamGaussian：Gaussian 作为生成式 3D 中间表示



Source: DreamGaussian paper / project figure

- 图中从输入图像或文本条件出发，先优化出 Gaussian 3D model，再生成 mesh。
- 这里 Gaussian 的角色从“重建真实场景”变成“生成式优化中的快速 3D representation”。

LGM：前馈预测 multi-view Gaussian



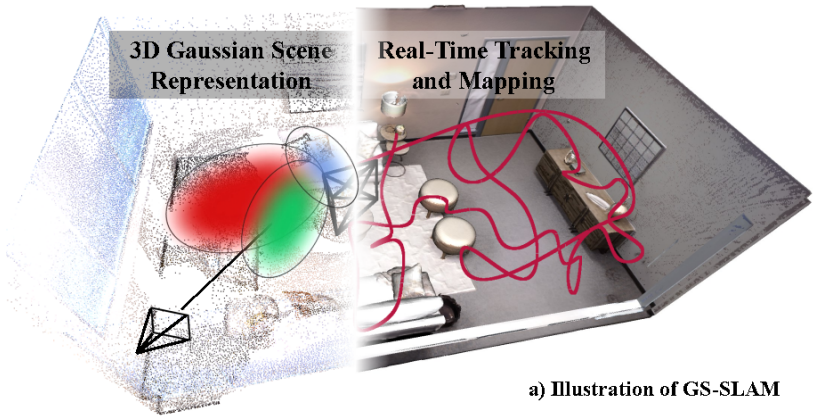
Source: LGM project page

- LGM pipeline 先生成多视角，再由 large multi-view Gaussian model 预测 Gaussian，最后转 mesh。
- 这对应生成方向的另一个分支：Gaussian 可以成为网络直接输出的 3D 表示。

Semantic Gaussian SLAM：Gaussian 作为语义地图

- **解决什么问题：**SLAM 需要在线定位和建图，语义 SLAM 还希望地图知道“哪里是什么”。
- **改造模块：**以 Gaussian map 作为可渲染地图，结合 tracking、map update 和 semantic feature rendering。
- **和原始 3DGS 的关系：**从离线新视角合成扩展到在线可更新地图。
- **说明的趋势：**Gaussian 可以成为可更新、可查询的空间记忆。

参考：GSFF-SLAM; MSGS-SLAM



Source: GS-SLAM project page

3DGS 应用正在形成独立研究生态

- 应用方向已经覆盖：
  - segmentation、editing、generation；
  - robotics、dynamic scenes、avatar；
  - medical/endoscopic scenes；
  - autonomous driving、occupancy 和 scene understanding。
- 读后续论文时，重点问三个问题：
  - 往 Gaussian 上增加了什么属性？
  - 改了哪种 loss、初始化或预测路径？
  - 目标是更好渲染、更准几何、更强语义，还是更快生成/建图？

参考：Awesome 3DGS Applications; A Survey on 3D Gaussian Splatting

对研究的启发：3DGS 作为 space representation

- 如果研究目标是视觉质量：关注投影、rasterizer、抗锯齿、动态场景和压缩。
- 如果研究目标是几何可靠性：关注深度、法线、表面约束、少视角稳定性。
- 如果研究目标是空间理解：关注 Gaussian 上挂载语义、语言、occupancy 或 feature。
- 如果研究目标是生成和编辑：关注 feed-forward Gaussian、对象分组和可控编辑。

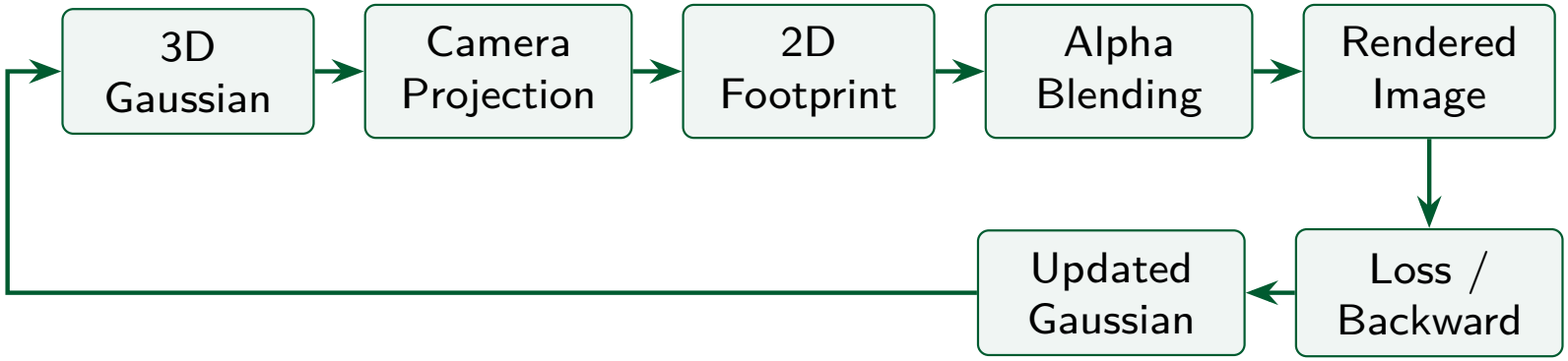
3DGS 的真正潜力，是把渲染、几何、语义和生成放进同一个显式 3D representation 中。

Part 6. 总结与答疑准备

- 总结 3DGS 的核心机制：表示、投影、渲染、优化。
- 概括后续发展：Gaussian primitives 承载更多空间信息。
- 提前准备常见追问：输入依赖、几何准确性、点云差别、实时性和应用扩展。

答疑时可以反复回到技术链路：Gaussian 的参数如何决定图像，图像误差又如何更新 Gaussian。

总结：3DGS 的核心机制



- 3DGS 把三维场景表示为一组显式、可学习的 Gaussian primitives。
- 每个 Gaussian 具有位置、尺度、旋转、透明度和颜色。
- 相机模型把 3D Gaussian 投影成二维 footprint。
- 多个 footprint 通过 alpha blending 合成图像。
- 图像误差沿可微渲染链路反向更新 Gaussian 参数。

这条链路概括了 3DGS 的核心机制：表示、投影、渲染、优化。

总结：从实时渲染到三维表示平台

- **原始论文：**证明显式 Gaussian primitives 可以实现高质量实时渲染。
- **数学机制：**Gaussian 的中心、协方差、opacity 和 SH color 共同决定图像贡献。
- **训练闭环：**rendered image 与真实图像计算 loss，梯度更新位置、尺度、旋转、透明度和颜色。
- **后续发展：**Gaussian 可以继续承载 depth、semantics、language、object grouping、editing 和 generation 信息。

因此，3DGS 的价值来自一套可解释的显式三维表示：它可以被投影、被混合、被优化，也可以继续扩展成面向理解和交互的空间信息载体。

答疑准备（一）：概念与数学

可能问题	回答要点
3DGS 和 neural field 的关系?	都服务于新视角合成；neural field 用函数查询颜色/密度，3DGS 用显式 Gaussian primitives，经 projection 和 splatting 生成图像。
$G(x)$ 是颜色还是概率?	$G(x)$ 是空间影响强度，表示点 x 位于该 Gaussian 影响范围内的程度；真正的像素贡献还要乘 opacity 和 color。
为什么要投影协方差?	Gaussian 有空间范围，中心投影只给出位置；协方差投影决定屏幕上的 footprint 大小、方向和软边界。
为什么用 R, S 构造 Σ?	直接优化协方差矩阵可能破坏合法性；用正尺度 S 和旋转 R 构造，可以保持椭球形状合理。

答疑准备（二）：训练、工程与应用

可能问题	回答要点
3DGS 必须依赖某个前处理工具吗?	核心需求是相机几何和初始表示；来源可以是数据集标注、多视角估计、SLAM、RGB-D/LiDAR、仿真或网络预测。
为什么渲染能反向更新位置?	位置影响投影中心，投影中心影响 footprint 覆盖像素，像素误差通过这条可微链路给出位置梯度。
为什么 fast rasterizer 会更快?	先把 Gaussian 分配到可能覆盖的 tiles，再在 tile 内排序和 alpha blending，避免每个 Gaussian 与所有像素暴力比较。
应用为什么能继续扩展?	Gaussian 是显式元素，可以在原有位置、形状、opacity、color 之外挂载 depth、semantic feature、language feature 或 object ID。

当前局限与未来方向

- **几何可靠性：**视觉效果好不代表表面和深度完全准确。
- **泛化能力：**per-scene optimization 成本高，feed-forward 仍在发展。
- **动态和语义：**真实世界是动态的，还包含对象、关系和语言指令。
- **系统集成：**要进入机器人、AR 和自动驾驶，还需要实时更新、压缩、安全和鲁棒性。

3DGS 的未来会同时追求渲染质量、几何可靠性、可控性和真实系统集成。

附录 A：四个坐标系再对照

坐标系	回答的问题	在 3DGS 中的作用
世界坐标	场景里的点在哪里	存 Gaussian 中心和相机位置
相机坐标	当前相机怎么看世界	判断深度和可见性
归一化坐标	透视投影后在哪里	表达近大远小
像素坐标	落到图像哪个像素	计算 footprint 和 loss 对齐

这条链路也是理解 3DGS 公式、渲染过程和后续扩展时最稳定的主线。

附录 B：Benchmark 阅读方式

- 图像质量指标：
  - PSNR：像素误差；
  - SSIM：结构相似度；
  - LPIPS：感知相似度。
- 效率指标：
  - FPS 或渲染时间；
  - 训练时间；
  - 显存和模型大小。
- 应用型论文还要看几何、语义、跟踪、编辑或生成指标。

不同论文优化目标不同，不能只看一个分数。

附录 C：资料来源

- 原始 3DGS：Kerbl et al., 3D Gaussian Splatting official project page
- 相机/隐式神经场背景：Lei Mao Camera Intrinsics and Extrinsics, PyTorch3D Cameras, PyImageSearch neural field tutorial
- 3DGS 教学解释：Hugging Face 3DGS blog, How to Render a Single Gaussian Splat?, Scthe's 3DGS rendering blog, Chauffier 3DGS
- 公式/实现笔记：kwea123 gaussian_splatting_notes, joeyan MATH.md, Evolve Benchmark
- 几何/深度：Depth Anything 3, DA3-GIANT, DepthSplat
- 语义/语言/编辑：LangSplat, LangSplatV2, Gaussian Grouping
- 生成/前向预测：DreamGaussian, LGM
- SLAM/综述：GSFF-SLAM, MSGS-SLAM, Awesome 3DGS Applications, 3DGS Survey
- 配图来源：Original 3DGS project page, MIT Vision Book, Shi Yan single Gaussian splat, NVIDIA 3DGUT, Depth Anything 3, DepthSplat, LangSplat, Gaussian Grouping, DreamGaussian, LGM, GS-SLAM project page; generated teaching diagrams.

补充讲解路线：从论文贡献回到方法细节

顺序	页	讲解作用
1. 先定贡献	01	从原始论文贡献和整体 pipeline 进入，明确方法由表示、渲染、优化三部分组成。
2. 铺数学底座	02–06	从一维 Gaussian 推到三维 Gaussian，再解释 covariance 为什么控制形状。
3. 相关工作对照	07–08	用 Point-NeRF 和 Plenoxels 对照 3DGS 的显式表示与高效渲染路线。
4. 回到具体方法	09–12	解释点里存什么、$G(x)$ 如何表达形状、3D Gaussian 如何可微 splatting。
5. 训练机制	13–14	说明参数优化、loss、clone/split/prune 和 adaptive density control。
6. 加速与收束	15–16	用 tile-based rasterizer 和 pipeline 回扣解释实时性和完整闭环。

这部分作为最后补充：按视频逻辑重新走一遍原始 3DGS 的“贡献—数学—方法—训练—加速”链路。

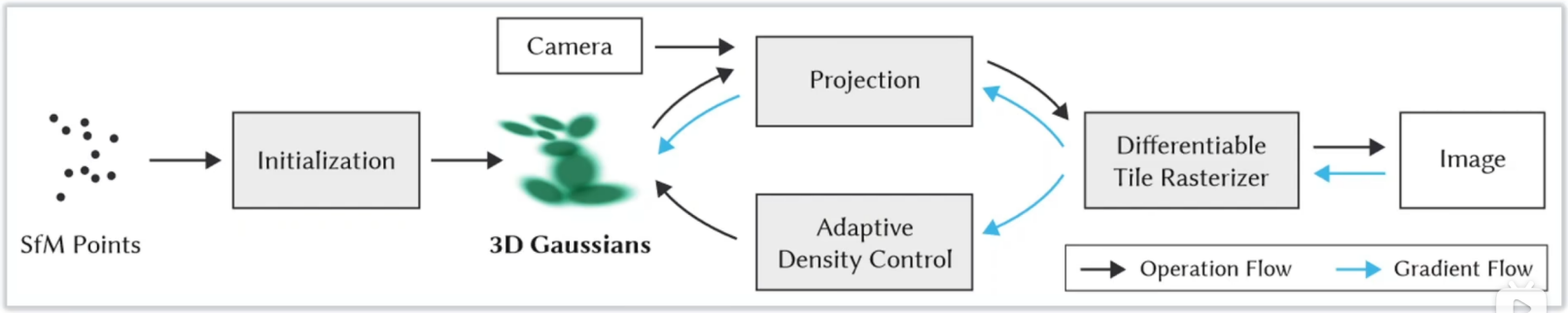
1. 论文贡献：3DGS 贡献与整体流程

Contributions

1 引入了一种各向异性(anisotropic)的3D高斯分布作为高质量、非结构化的辐射场表达；

2 实现了使用GPU进行快速可微的渲染，允许各向异性的抛雪球(Splatting)和快速反向传播。

3 提出了针对3D高斯特性的优化方法，并同时进行自适应密度控制；



Source: Video lecture PPT screenshot provided by user

2. 数学预备：一维 Gaussian 的基本形式

3D Gaussian

- 一维高斯分布的概率密度函数

$$x \sim \mathcal{N}(\mu, \sigma^2)$$

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

均值

方差

- 三维高斯分布的概率密度函数

$$\mathbf{v} = [a, b, c]^T$$

N(0,1)

$$p(\mathbf{v}) = p(a)p(b)p(c)$$

$$= \frac{1}{(2\pi)^{3/2}} exp(-\frac{a^2+b^2+c^2}{2})$$

$$= \frac{1}{(2\pi)^{3/2}} exp(-\frac{1}{2}\mathbf{v}^T\mathbf{v})$$

Source: Video lecture PPT screenshot provided by user

2. 数学预备：三维 Gaussian 概率密度

3D Gaussian

$$p(\mathbf{v}) = \frac{1}{(2\pi)^{3/2}} exp(-\frac{1}{2}\mathbf{v}^T\mathbf{v})$$

变换矩阵

$$\mathbf{v} = \mathbf{A}(\mathbf{x} - \mu)$$

N(0,1): $\mathbf{v} = [a, b, c]^T$

任意分布: $\mathbf{x} = [x, y, z]^T$

$$= \frac{1}{(2\pi)^{3/2}} exp(-\frac{1}{2}(\mathbf{x} - \mu)^T\mathbf{A}^T\mathbf{A}(\mathbf{x} - \mu))$$

均值向量 $\mu = [E(x), E(y), E(z)]^T$

Source: Video lecture PPT screenshot provided by user

2. 数学预备：仿射变换与变量替换

3D Gaussian

$$p(\mathbf{v}) = \frac{1}{(2\pi)^{3/2}} exp(-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{A}^T \mathbf{A}(\mathbf{x}-\mu))$$

两边积分

$$d\mathbf{v} = d(\mathbf{A}(\mathbf{x}-\mu)) = |\mathbf{A}|d\mathbf{x}$$

$$1 = \iiint_{-\infty}^{+\infty} \frac{1}{(2\pi)^{3/2}} exp(-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{A}^T \mathbf{A}(\mathbf{x}-\mu))d\mathbf{v}$$

$$1 = \iiint_{-\infty}^{+\infty} \frac{|\mathbf{A}|}{(2\pi)^{3/2}} exp(-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{A}^T \mathbf{A}(\mathbf{x}-\mu))d\mathbf{x}$$

Source: Video lecture PPT screenshot provided by user

2. 数学预备：仿射变换后的 Gaussian 密度

3D Gaussian

$$1 = \iiint_{-\infty}^{+\infty} \frac{|\mathbf{A}|}{(2\pi)^{3/2}} exp(-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{A}^T \mathbf{A}(\mathbf{x}-\mu)) d\mathbf{x}$$

概率密度函数：

$$p(\mathbf{x}) = \frac{|\mathbf{A}|}{(2\pi)^{3/2}} exp(-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{A}^T \mathbf{A}(\mathbf{x}-\mu))$$

Source: Video lecture PPT screenshot provided by user

2. 数学预备：协方差矩阵与特征分解

3D Gaussian

协方差矩阵：

特征值分解

$$\Sigma = U\Lambda U^T$$

$UU^T = I$

特征值矩阵

$$\Sigma = (U\Lambda^{1/2})(U\Lambda^{1/2})^T$$

$$\Sigma = \mathbf{A}\mathbf{A}^T$$

概率密度函数

$$p(\mathbf{x}) = \frac{|\mathbf{A}|}{(2\pi)^{3/2}} exp(-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{A}^T\mathbf{A}(\mathbf{x}-\mu))$$

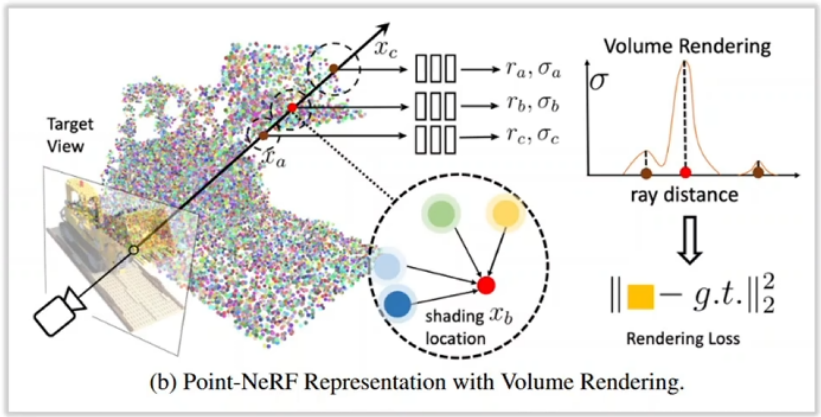
$$= \frac{1}{|\Sigma|^{1/2}(2\pi)^{3/2}} exp(-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu))$$

Source: Video lecture PPT screenshot provided by user

3. 相关工作对照：Point-NeRF

Point-NeRF

1. 为每个点赋予特征向量；
2. 体渲染时取采样点周边一定领域内的点的特征做线性插值，解码得颜色和体密度；
3. 自适应点云生长和剔除。



(b) Point-NeRF Representation with Volume Rendering.

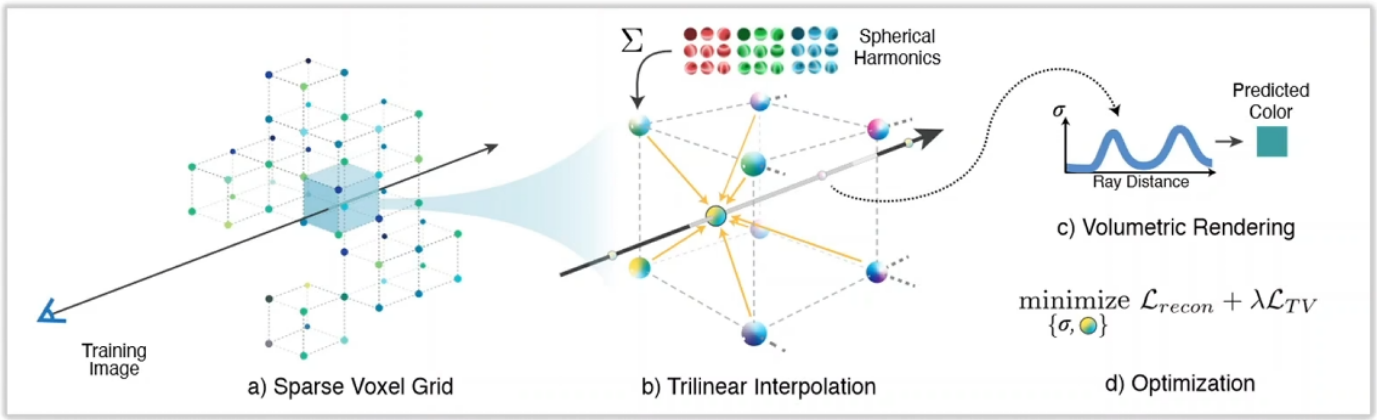
[CVPR 2022] Point-NeRF: Point-Based Neural Radiance Fields

Source: Video lecture PPT screenshot provided by user

3. 相关工作对照：Plenoxels

Plenoxels

- 体素格点储存球谐函数(spherical harmonic)系数，系数做插值得到采样位置的球谐函数。



[CVPR 2022] Plenoxels: Radiance Fields without Neural Networks

Source: Video lecture PPT screenshot provided by user

4. 具体方法入口：三块核心内容

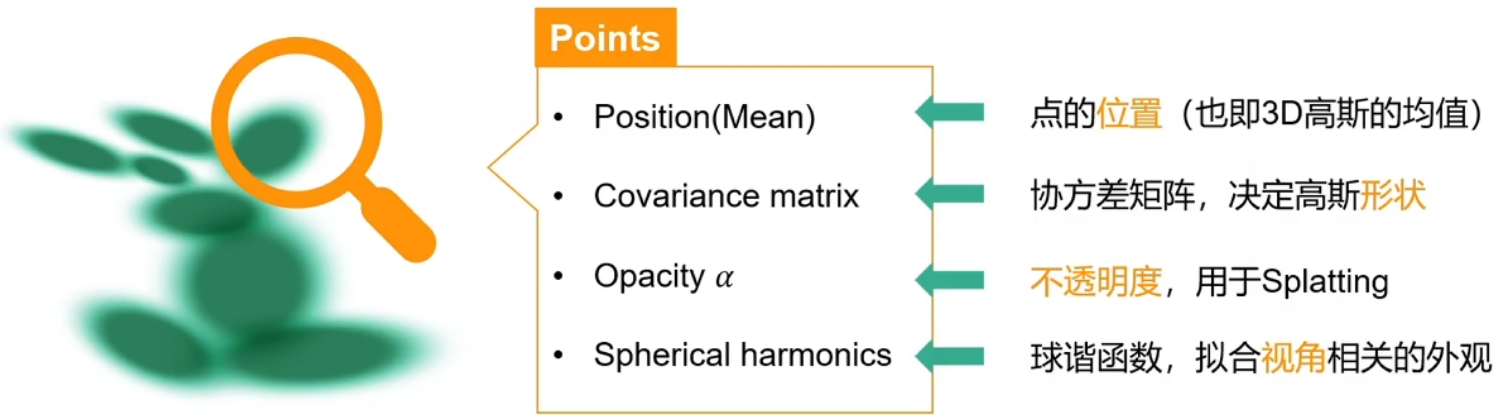


具体方法

1. 可微的3D高斯Splatting
2. 优化与自适应密度控制
3. 快速可微光栅化

Source: Video lecture PPT screenshot provided by user

4. 具体方法：Point 参数与可学习属性



Source: Video lecture PPT screenshot provided by user

4. 具体方法：Covariance Matrix 与 $G(x)$

Covariance Matrix

1. 3D高斯的定义

$$\frac{1}{|\Sigma|^{1/2}(2\pi)^{3/2}} exp(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu))$$

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

Source: Video lecture PPT screenshot provided by user

可微的3D高斯Splatting

Differentiable 3D Gaussian Splatting

Covariance Matrix

2. 协方差矩阵的物理含义

$$\Sigma = \mathbf{A}\mathbf{A}^T$$

旋转 $R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$

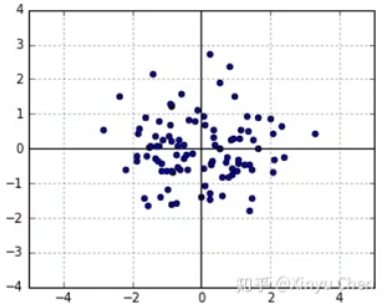
$A = RS$

缩放 $S = \begin{bmatrix} s_y & 0 \\ 0 & s_x \end{bmatrix}$

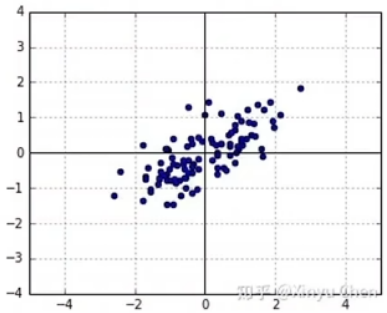
$$\Sigma = RSS^T R^T$$

（旋转矩阵的优化？求导？渲染？）

图源——知乎@Xinyu Chen：如何直观地理解「协方差矩阵」？



Δ 线性变换前



Δ 线性变换后

Source: Video lecture PPT screenshot provided by user

5. 训练机制：优化目标与训练细节

Optimization

- 随机梯度下降（Stochastic Gradient Descent）
- CUDA核心
- 快速光栅化
- Sigmoid 和 Exponential 激活函数
- 损失函数：

$$\mathcal{L} = (1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{D-SSIM}}$$

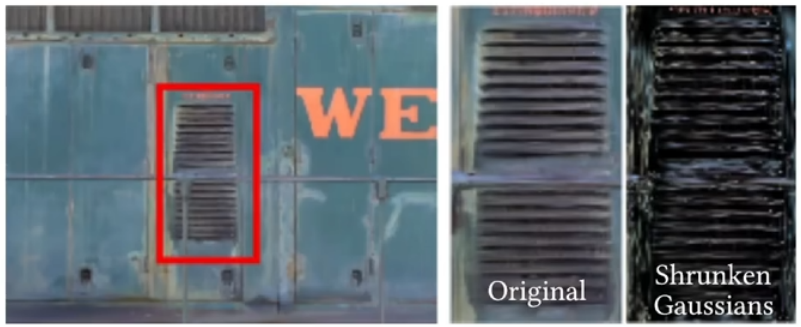


Fig. 3. We visualize the 3D Gaussians after optimization by shrinking them 60% (far right). This clearly shows the anisotropic shapes of the 3D Gaussians that compactly represent complex geometry after optimization. Left the actual rendered image.

Source: Video lecture PPT screenshot provided by user

5. 训练机制：Adaptive Control of Gaussians

Adaptive Control of Gaussians

- 每100次迭代移除不透明度小于阈值的点；
- 重建不充分的区域往往会有较大的梯度
  - Under-reconstruction：clone
  - Over-reconstruction：split
- 周期性将不透明度重置为0用于去除floaters；
- 周期性移除较大的高斯用于避免重叠；

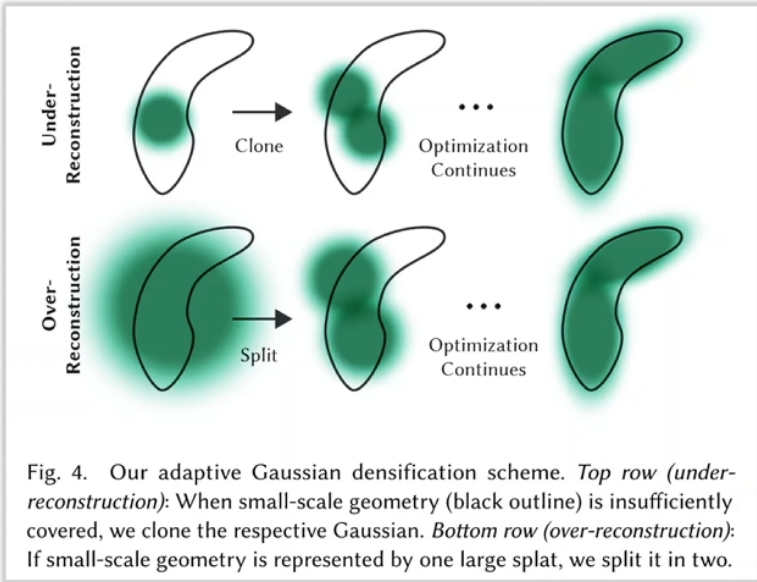


Fig. 4. Our adaptive Gaussian densification scheme. *Top row (under-reconstruction)*: When small-scale geometry (black outline) is insufficiently covered, we clone the respective Gaussian. *Bottom row (over-reconstruction)*: If small-scale geometry is represented by one large splat, we split it in two.

Source: Video lecture PPT screenshot provided by user

6. 工程加速：Tile-based Rasterizer



快速可微光栅化

Fast Differentiable Rasterizer for Gaussians

Tile-based Rasterizer

1. 把整个图像划分为 16*16 个 tiles，每个 tile 视锥内挑选可视的3D Gaussian；
2. 每个视锥内只取执行度大于 99% 的高斯，并按深度排序；
3. 并行地在每个 tile 上 splat；
4. 有像素的不透明度达到饱和就停止对应线程；
5. 反向传播误差时按 tile 对高斯进行索引；

Source: Video lecture PPT screenshot provided by user

6. Pipeline 回扣：原始 3DGS 完整闭环

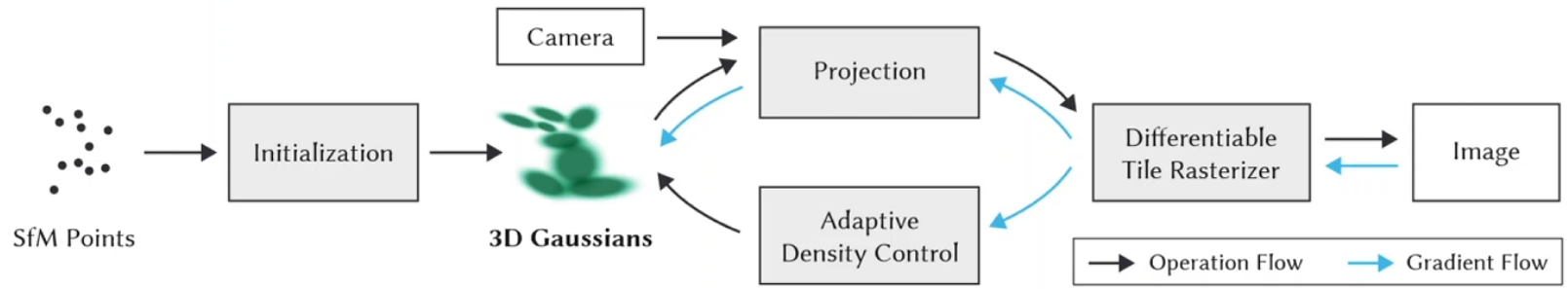


Fig. 2. Optimization starts with the sparse SfM point cloud and creates a set of 3D Gaussians. We then optimize and adaptively control the density of this set of Gaussians. During optimization we use our fast tile-based renderer, allowing competitive training times compared to SOTA fast radiance field methods. Once trained, our renderer allows real-time navigation for a wide variety of scenes.

Source: Video lecture PPT screenshot provided by user

谢谢大家