以GaussianImage为范本的pipline（丢掉z）

$\boldsymbol{\mu} \in \mathbb{R}^2$ 2 实际上应该更新O，也就是offsets

$\boldsymbol{c}'_n \in \mathbb{R}^3$ 3

$\boldsymbol{\Sigma} = (\boldsymbol{RS})(\boldsymbol{RS})^T \quad \boldsymbol{R} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \quad \boldsymbol{S} = \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix}$ 3

or $\boldsymbol{\Sigma} = \boldsymbol{LL}^T \quad \boldsymbol{l} = \{l_1, l_2, l_3\}$ 3

已否决

$$\boldsymbol{C}_i = \sum_{n \in \mathcal{N}} \boldsymbol{c}_n \cdot \alpha_n \cdot T_n, \quad T_n = \prod_{m=1}^{n-1} (1 - \alpha_m)$$

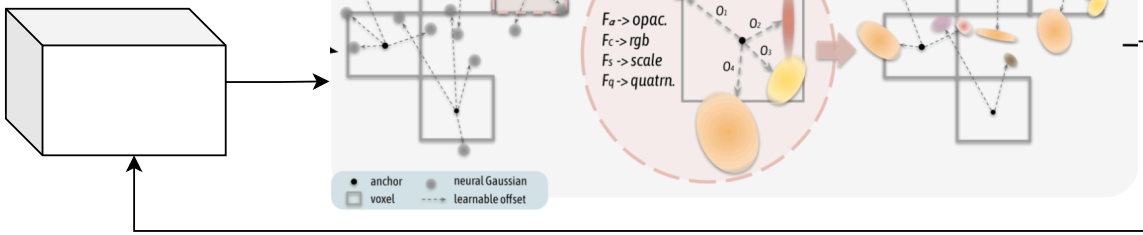
$$\alpha_n = o_n \cdot \exp(-\sigma_n), \quad \sigma_n = \frac{1}{2} \boldsymbol{d}_n^T \boldsymbol{\Sigma}^{-1} \boldsymbol{d}_n$$

where d ∈ R2 is the displacement between the pixel center and the projected 2D Gaussian center

$$\boldsymbol{C}_i = \sum_{n \in \mathcal{N}} \boldsymbol{c}_n \cdot \alpha_n = \sum_{n \in \mathcal{N}} \boldsymbol{c}_n \cdot o_n \cdot \exp(-\sigma_n).$$

$$\boldsymbol{C}_i = \sum_{n \in \mathcal{N}} \boldsymbol{c}'_n \cdot \exp(-\sigma_n)$$

8x8xKx(C+2+3+3)

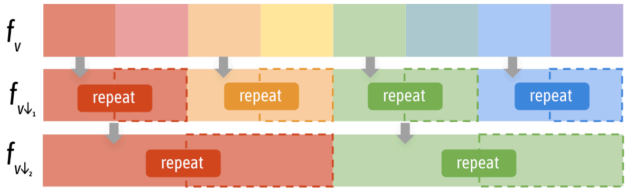


Gradient

Loss

$\mathbf{V} = \left\{ \left\lfloor \frac{\mathbf{P}}{\epsilon} \right\rceil \right\} \cdot \epsilon$ $\mathbf{V} \in \mathbb{R}^{N \times 3}$ denotes voxel centers 事实上，如果按照它GaussianImage的Method，这里应该是Nx2

$\delta_{vc} = \|\mathbf{x}_v - \mathbf{x}_c\|_2, \vec{\mathbf{d}}_{vc} = \frac{\mathbf{x}_v - \mathbf{x}_c}{\|\mathbf{x}_v - \mathbf{x}_c\|_2}$ 事实上，这里的d可以直接删掉

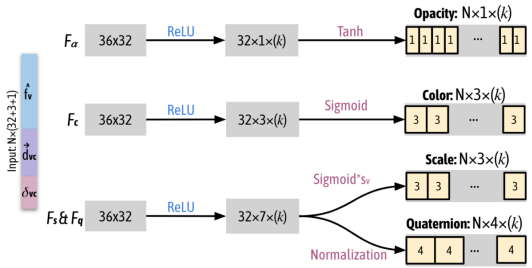


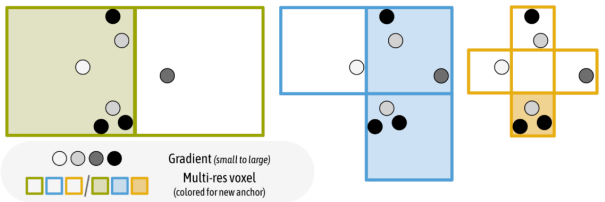
$$\{w, w_1, w_2\} = \text{Softmax}(F_w(\delta_{vc}, \vec{\mathbf{d}}_{vc})),$$

$$\hat{f}_v = w \cdot f_v + w_1 \cdot f_{v_{\downarrow_1}} + w_2 \cdot f_{v_{\downarrow_2}},$$

同理，这里的d可以直接删掉

$\{\mu_0, ..., \mu_{k-1}\} = \mathbf{x}_v + \{\mathcal{O}_0, \ldots, \mathcal{O}_{k-1}\} \cdot l_v \{\mathcal{O}_0, \mathcal{O}_1, ..., \mathcal{O}_{k-1}\} \in \mathbb{R}^{k \times 3}$ O应该是R^(k*2)





For each voxel, we compute the averaged gradients of the included neural Gaussians over N training iterations, denoted as ∇_g

$\nabla_g > \tau_g$ is deemed as *significant*

$$\epsilon_g^{(m)} = \epsilon_g / 4^{m-1}, \quad \tau_g^{(m)} = \tau_g * 2^{m-1}$$

Figure 3. **Growing operation.** We develop an anchor growing policy guided by the gradients of the neural Gaussians. From left to right, we spatially quantize neural Gaussians into multi-resolution voxels ($m \in \{1, 2, 3\}$) of size $\{\epsilon_g^{(m)}\}$. New anchors are added to voxels with aggregated gradients larger than $\{\tau_g^{(m)}\}$.

$$\mathcal{L} = \mathcal{L}_1 + \lambda_{\text{SSIM}} \mathcal{L}_{\text{SSIM}} + \lambda_{\text{vol}} \mathcal{L}_{\text{vol}},$$

$$\mathcal{L}_{\text{vol}} = \sum_{i=1}^{N_{\text{ng}}} \text{Prod}(s_i)$$

以2DGS为范本的pipline

已通过

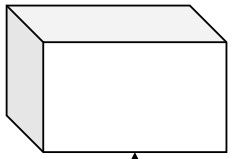
$\{\mathcal{O}_0, \mathcal{O}_1, ..., \mathcal{O}_{k-1}\} \in \mathbb{R}^{k\times 3}$ 3 实际上按照有z轴我们应该是有3维

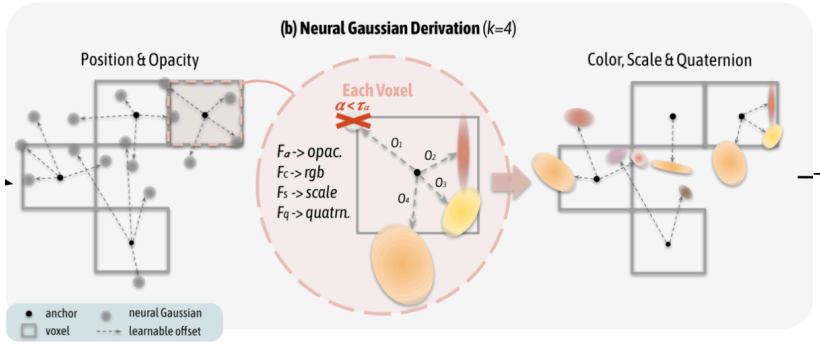
$c'_n \in \mathbb{R}^3$ 3 R:实际上三个自由度 S:实际两个参数 5个参数

$\Sigma = (RS)(RS)^T$ $\mathbf{R} = [\mathbf{t}_u, \mathbf{t}_v, \mathbf{t}_w]$ $\mathbf{S} = (s_u, s_v)$ $\mathbf{t}_w = \mathbf{t}_u \times \mathbf{t}_v$

$$\mathbf{c}(\mathbf{x}) = \sum_{i=1} \mathbf{c}_i \, \alpha_i \, \hat{\mathcal{G}}_i(\mathbf{u}(\mathbf{x})) \prod_{j=1}^{i-1} (1 - \alpha_j \, \hat{\mathcal{G}}_j(\mathbf{u}(\mathbf{x})))$$

8x8xKx(C+3+3+5)





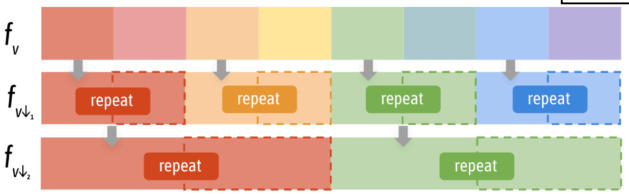
Gradient

Loss

$\mathbf{V} = \left\{ \left\lfloor \frac{\mathbf{P}}{\epsilon} \right\rfloor \right\} \cdot \epsilon$ $\mathbf{V} \in \mathbb{R}^{N\times 3}$ denotes voxel centers

By modeling the scene as dynamic 3D Gaussians in the camera coordinate space, we intertwine camera motion with object motion and treat them as the same type of motion, eliminating the need for camera calibration. During the rendering process, the 3D Gaussians in the camera coordinate space are rasterized into images from an identity pose camera. This approach simplifies the representation of dynamics and avoids the challenges of estimating camera pose from monocular casual videos.

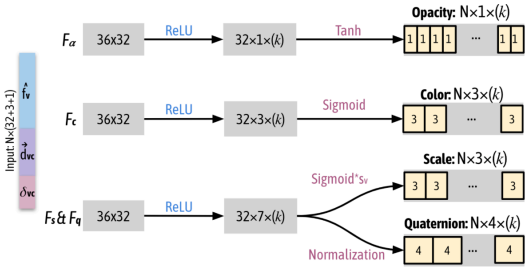
$$\delta_{vc} = \|\mathbf{x}_v - \mathbf{x}_c\|_2, \vec{\mathbf{d}}_{vc} = \frac{\mathbf{x}_v - \mathbf{x}_c}{\|\mathbf{x}_v - \mathbf{x}_c\|_2}$$



$$\{w, w_1, w_2\} = \text{Softmax}(F_w(\delta_{vc}, \vec{\mathbf{d}}_{vc})),$$

$$\hat{f}_v = w \cdot f_v + w_1 \cdot f_{v_{\downarrow_1}} + w_2 \cdot f_{v_{\downarrow_2}}, \quad \{\mathcal{O}_0, \mathcal{O}_1, ..., \mathcal{O}_{k-1}\} \in \mathbb{R}^{k\times 3}$$

$$\{\mu_0, ..., \mu_{k-1}\} = \mathbf{x}_v + \{\mathcal{O}_0, \ldots, \mathcal{O}_{k-1}\} \cdot l_v$$

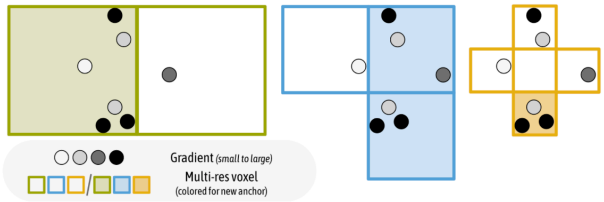


$\mathcal{L}_n = \sum_i \omega_i (1 - \mathbf{n}_i^{\mathrm{T}} \mathbf{N})$可以先用单目估计来预测法向量

$\omega_i = \alpha_i \, \mathcal{G}_i(\mathbf{u}(\mathbf{x})) \prod_{j=1}^{i-1} (1 - \alpha_j \, \hat{\mathcal{G}}_j(\mathbf{u}(\mathbf{x})))$

$\mathcal{L}_{\text{vol}} = \sum_{i=1}^{N_{\text{ng}}} \text{Prod}(s_i)$ $\mathcal{L}_d = \sum_{i,j} \omega_i \omega_j |z_i - z_j|$

$$\mathcal{L} = \mathcal{L}_c + \alpha \mathcal{L}_d + \beta \mathcal{L}_n + \lambda_{\text{vol}} \mathcal{L}_{\text{vol}}$$



For each voxel, we compute the averaged gradients of the included neural Gaussians over N training iterations, denoted as ∇_g

$\nabla_g > \tau_g$ is deemed as *significant*

$$\epsilon_g^{(m)} = \epsilon_g / 4^{m-1}, \quad \tau_g^{(m)} = \tau_g * 2^{m-1}$$

Figure 3. **Growing operation.** We develop an anchor growing policy guided by the gradients of the neural Gaussians. From left to right, we spatially quantize neural Gaussians into multi-resolution voxels ($m \in \{1, 2, 3\}$) of size $\{\epsilon_g^{(m)}\}$. New anchors are added to voxels with aggregated gradients larger than $\{\tau_g^{(m)}\}$.

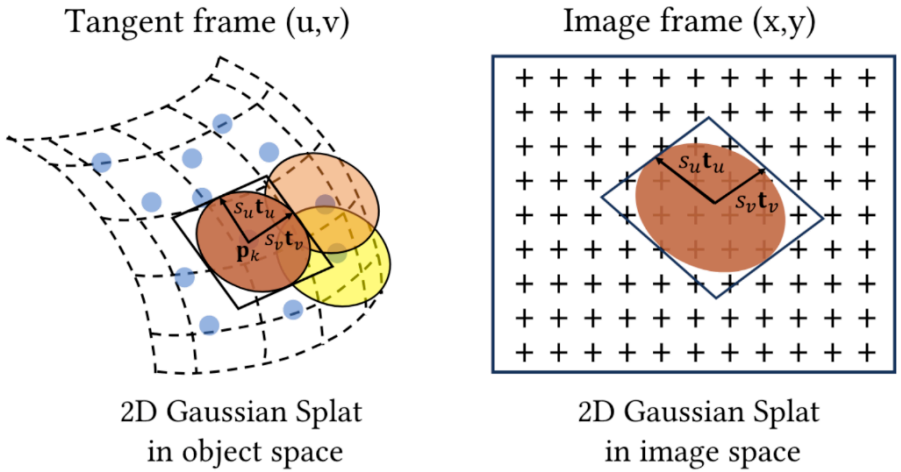
As illustrated in Figure 3, our 2D splat is characterized by its central point $\mathbf{p}_k$, two principal tangential vectors $\mathbf{t}_u$ and $\mathbf{t}_v$, and a scaling vector $\mathbf{S} = (s_u, s_v)$ that controls the variances of the 2D Gaussian. Notice that the primitive normal is defined by two orthogonal tangential vectors $\mathbf{t}_w = \mathbf{t}_u \times \mathbf{t}_v$. We can arrange the orientation into a 3×3 rotation matrix $\mathbf{R} = [\mathbf{t}_u, \mathbf{t}_v, \mathbf{t}_w]$ and the scaling factors into a 3×3 diagonal matrix $\mathbf{S}$ whose last entry is zero.

A 2D Gaussian is therefore defined in a local tangent plane in world space, which is parameterized:

$$P(u,v) = \mathbf{p}_k + s_u \mathbf{t}_u u + s_v \mathbf{t}_v v = \mathbf{H}(u, v, 1, 1)^{\mathrm{T}} \tag{4}$$

$$\text{where } \mathbf{H} = \begin{bmatrix} s_u \mathbf{t}_u & s_v \mathbf{t}_v & \mathbf{0} & \mathbf{p}_k \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{RS} & \mathbf{p}_k \\ \mathbf{0} & 1 \end{bmatrix} \tag{5}$$

The center $\mathbf{p}_k$, scaling (s_u, s_v), and the rotation $(\mathbf{t}_u, \mathbf{t}_v)$ are learnable parameters. Following 3DGS [Kerbl et al. 2023], each 2D Gaussian primitive has opacity α and view-dependent appearance c parameterized with spherical harmonics.



2DGS Pipeline

$$\mathbf{h}_x = (-1, 0, 0, x)^{\mathrm{T}} \qquad \mathbf{h}_y = (0, -1, 0, y)^{\mathrm{T}}$$

$$\mathbf{h}_u = (\mathbf{WH})^{\mathrm{T}} \mathbf{h}_x \qquad \mathbf{h}_v = (\mathbf{WH})^{\mathrm{T}} \mathbf{h}_y$$

W：4x4为世界坐标到屏幕坐标投影矩阵

$$u(\mathbf{x}) = \frac{\mathbf{h}_u^2 \mathbf{h}_v^4 - \mathbf{h}_u^4 \mathbf{h}_v^2}{\mathbf{h}_u^1 \mathbf{h}_v^2 - \mathbf{h}_u^2 \mathbf{h}_v^1} \qquad v(\mathbf{x}) = \frac{\mathbf{h}_u^4 \mathbf{h}_v^1 - \mathbf{h}_u^1 \mathbf{h}_v^4}{\mathbf{h}_u^1 \mathbf{h}_v^2 - \mathbf{h}_u^2 \mathbf{h}_v^1}$$

$$\mathcal{G}(\mathbf{u}) = \exp\left(-\frac{u^2 + v^2}{2}\right)$$

$$\hat{\mathcal{G}}(\mathbf{x}) = \max\left\{\mathcal{G}(\mathbf{u}(\mathbf{x})), \mathcal{G}\left(\frac{\mathbf{x} - \mathbf{c}}{\sigma}\right)\right\}$$

$$\mathbf{c}(\mathbf{x}) = \sum_{i=1} \mathbf{c}_i \, \alpha_i \, \hat{\mathcal{G}}_i(\mathbf{u}(\mathbf{x})) \prod_{j=1}^{i-1} (1 - \alpha_j \, \hat{\mathcal{G}}_j(\mathbf{u}(\mathbf{x})))$$

Loss

$$\mathcal{L}_d = \sum_{i,j} \omega_i \omega_j |z_i - z_j| \qquad \mathcal{L}_n = \sum_i \omega_i (1 - \mathbf{n}_i^{\mathrm{T}} \mathbf{N}) \qquad \mathbf{N}(x,y) = \frac{\nabla_x \mathbf{p}_s \times \nabla_y \mathbf{p}_s}{|\nabla_x \mathbf{p}_s \times \nabla_y \mathbf{p}_s|}$$

$$\omega_i = \alpha_i \, \hat{\mathcal{G}}_i(\mathbf{u}(\mathbf{x})) \prod_{j=1}^{i-1} (1 - \alpha_j \, \hat{\mathcal{G}}_j(\mathbf{u}(\mathbf{x})))$$

$$\mathcal{L} = \mathcal{L}_c + \alpha \mathcal{L}_d + \beta \mathcal{L}_n$$

Splatter A Video参考

2D 单目先验 (2D Monocular Priors)

单目 2D 视频通常会丢失深度信息且容易造成运动与外观的耦合。为此，作者利用现有的 2D 视觉理解模型提供的先验信息，来正则化 3D 高斯体的学习过程：

1. **光流约束 (Flow Distillation)**：
   - 光流描述了视频帧间像素的 2D 投影运动。
   - 使用 RAFT 等光流估计模型从输入视频中提取光流，作为正则化信号。
   - 将高斯体的 3D 运动在 XY 平面投影后，与光流估计值对齐，公式如下：

$$L_{\text{flow}} = \mathbb{E}(t_1, t_2) \| R(\mu(t_1), q, s, \alpha, \pi(\mu(t_2) - \mu(t_1))) - \text{flow} t_1 \to t_2 \|_1$$

其中，π 是从 3D 到 2D 的投影函数，$\mu(t)$ 是高斯体的位置，确保 3D 运动的 2D 投影与光流一致。

2. **深度约束 (Depth Distillation)**：
   - 深度估计提供了每帧的场景几何信息。
   - 使用 MiDaS 提出的尺度平移调整损失函数，确保生成的高斯体深度与单目深度估计一致，公式为：

$$L_{\text{depth}} = \mathbb{E}_t \left(\| \tau(D_t) - \tau(\widehat{D}_t) \|^2 \right)$$

其中，D_t 是高斯体生成的深度图，$\widehat{D}_t$ 是深度估计值，τ 是深度归一化函数 (移除尺度与平移偏差)。

3D 动态正则化 (3D Motion Regularization)

为了避免模型在学习过程中产生非真实的 3D 动态 (如过度弯曲或不规则运动)，引入以下正则化方法：

1. **局部刚性约束 (Local Rigidity Regularization)**：
   - 限制每个高斯体相对于其邻域内其他高斯体的运动为刚性变换。
   - 对某一高斯体 G_i，找到其在时间 t_1 时的邻域 $G_k (k \in N_i)$，并约束时间 t_2 时其相对位置变化为刚性，公式为：

$$L_{\text{rigid}} = \mathbb{E} i, t_1, t_2 \left(\sum k \in N_i \| (\mu_i(t_1) - \mu_k(t_1)) - R(\mu_i(t_2) - \mu_k(t_2)) \|^2 \right)$$

其中，R 是刚性旋转矩阵。

2. **3D 动态一致性 (3D Motion Consistency)**：
   - 结合光流 (XY 方向) 和深度 (Z 方向) 约束，确保高斯体在 3D 空间中的运动轨迹符合真实世界的物理动态。

$$\mathcal{L}_{\text{arap}} = \mathbb{E}_{(i,t_1,t_2)} \left(\sum_{k \in \mathcal{N}_i} ||(\mu_i(t_1) - \mu_k(t_1)) - \hat{R}_i(\mu_i(t_2) - \mu_k(t_2))||^2 \right),$$

where R is the estimated rigid rotation transformation given by

$$\hat{R}_i = \underset{R \in \mathbf{SO}(3)}{\arg\min} \sum_{k \in \mathcal{N}_i} ||\mu_i(t_1) - \mu_k(t_1)) - R(\mu_i(t_2) - \mu_k(t_2))||^2.$$

$$\mathcal{L}_{\text{label}} = \mathbb{E}_t \left(||\mathcal{R}(\mu(t), q, s, \alpha, m) - \mathcal{M}^t||_2^2 \right)$$

$$\mathcal{L} = \lambda_{\text{render}} \mathcal{L}_{\text{render}} + \lambda_{\text{depth}} \mathcal{L}_{\text{depth}} + \lambda_{\text{flow}} \mathcal{L}_{\text{flow}} + \lambda_{\text{arap}} \mathcal{L}_{\text{arap}} + \lambda_{\text{label}} \mathcal{L}_{\text{label}}$$

$$\mu(t) = \mu_0 + \sum_{n=0}^{N} p_p^n t^n + \sum_{l=0}^{L} (p_{\sin}^l \cos(lt) + p_{\cos}^l \sin(lt))$$

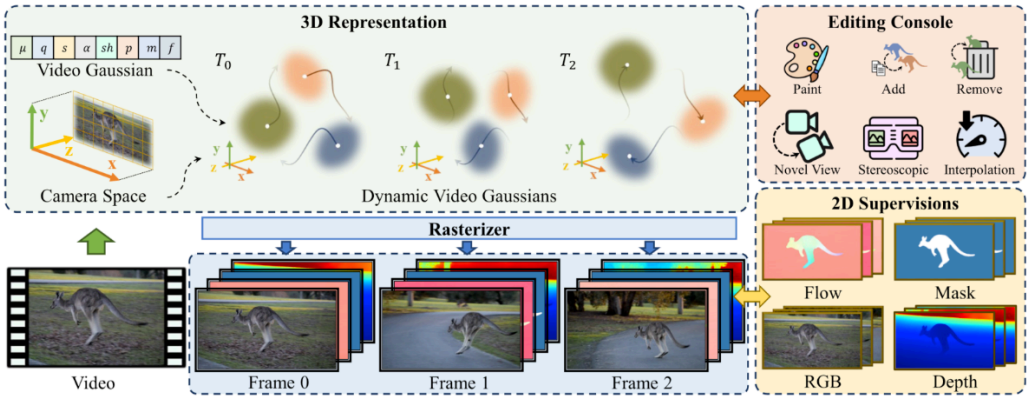


Figure 2: **Pipeline of our approach.** Given a video, we represent its intricate 3D content using video Gaussians in the camera coordinate space. By associating them with motion parameters, we enable video Gaussians to capture the video dynamics. These video Gaussians are supervised by RGB image frames and 2D priors such as optical flow, depth, and label masks. This representation makes it convenient for users to perform various editing tasks on the video.