Fast Differentiable Rendering with 3D-GS

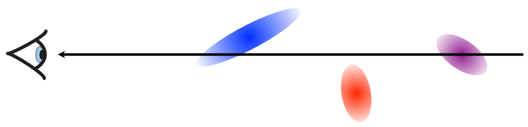
Based on slides from [Takikawa et al, 2023] and [Tulsiani, 2024]

Outline

- Differentiable Primitive Rendering
- Gaussian Splatting

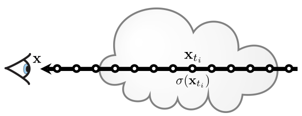
3D Gaussian Splatting

3DGS: Differentiable Primitive Rendering



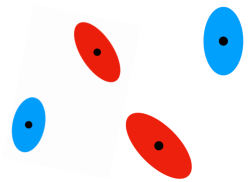
Slides from S. Tulsiani and V. Sitzmann

Volumes: Rendering and Representation



$$\sigma_{t_i} \equiv \sigma(\mathbf{x}_{t_i}) \longrightarrow L(\mathbf{x}, \omega)$$
$$L_e(\mathbf{x}_{t_i}, \omega)$$

Rendering Algorithm

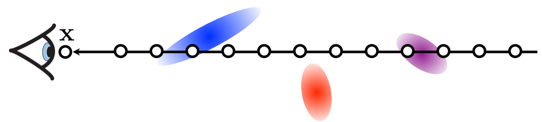


Option 3: 3D Gaussian Splats

(Tulsiani)

Rendering Primitives (e.g. Gaussians)

Ray Marching

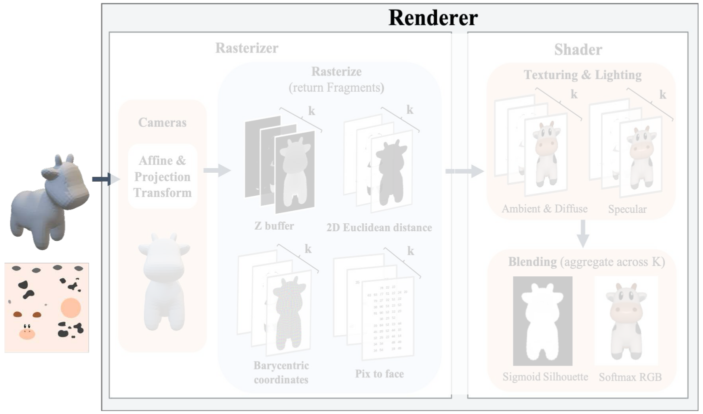


1. Draw samples along the ray

2. Aggregate their contributions to render

(S. Tulsiani, 2024)

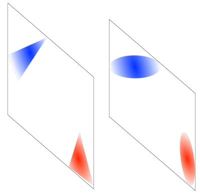
From Ray tracing to Rasterization

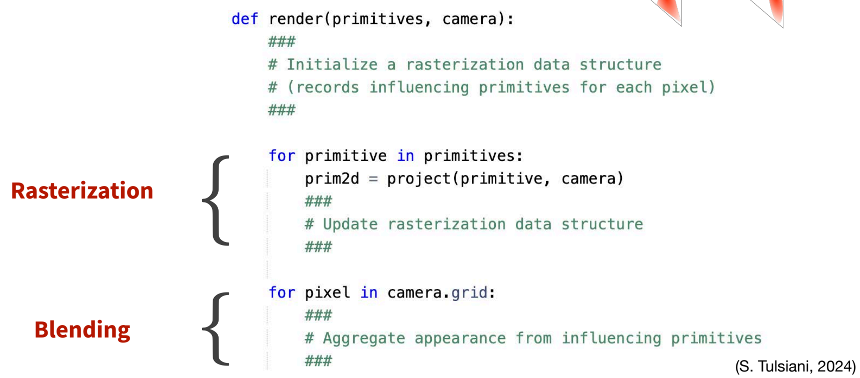


Accelerating 3D Deep Learning with PyTorch3D. *Ravi et. al.*

(S. Tulsiani, 2024)

Differentiable Primitive Rendering






```
def render(primitives, camera):
    ###
    # Initialize a rasterization data structure
    # (records influencing primitives for each pixel)
    ###

    for primitive in primitives:
        prim2d = project(primitive, camera)
        ###
        # Update rasterization data structure
        ###

    for pixel in camera.grid:
        ###
        # Aggregate appearance from influencing primitives
        ###
```

(S. Tulsiani, 2024)

Differentiable Gaussian Rendering

What is the representation of a 3D gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?

Rasterization

Blending

```
def render(gaussians, camera):
    ###
    # Initialize a rasterization data structure
    # (records influencing primitives for each pixel)
    ###

    for gaussian in gaussians:
        gauss2d = project(gaussian, camera)
        ###
        # Update rasterization data structure
        ###

    for pixel in camera.grid:
        ###
        # Aggregate appearance from influencing gaussians
        ###
```

(S. Tulsiani, 2024)

Differentiable Gaussian Rendering

What is the representation of a 3D gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?

(S. Tulsiani, 2024)

Differentiable Gaussian Rendering

What is the representation of a 3D gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?

$$\mathcal{G}_{\mathbf{V}}(\mathbf{x}-\mathbf{p}) = \frac{1}{2\pi|\mathbf{V}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{p})^T \mathbf{V}^{-1}(\mathbf{x}-\mathbf{p})}$$

Factorize as scale and rotation: $\mathbf{V} = RSS^T R^T$

Each gaussian also has an opacity and view-dependent color (via SH coefficients): $\alpha, \mathbf{c}$

Slide adapted from Vincent Sitzmann.

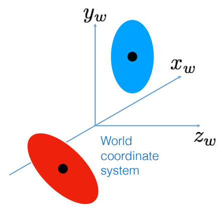
(S. Tulsiani, 2024)

Differentiable Gaussian Rendering

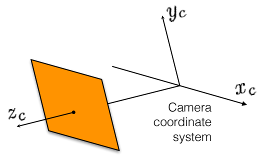
What is the representation of a 3D gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?



$\mathbf{p}, R, S$



$\mathbf{p}', R', S$

We can use the camera extrinsics to transform each 3D gaussian to the camera frame

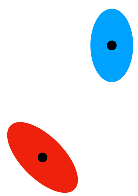
(S. Tulsiani, 2024)

Differentiable Gaussian Rendering

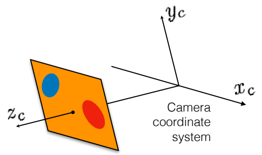
What is the representation of a 3D gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?



$\mathbf{p}', R', S$



Q: What is the image-space projection of a 3D gaussian?

A: Can approximate as a 2D gaussian!

(EWA Volume Splatting, *Zwicker et. al., 2001*)

$$\pi(\mathbf{x}) = \mathbf{u}$$

π: Projection function for mapping 3D points to pixels

$$z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

2D mean: $\mu_{2D} = \pi(\mu_{3D})$

2D covariance:

$$J = \frac{\partial \pi}{\partial \mathbf{x}}(\mu_{3D})$$

$$\Sigma_{2D} = J \Sigma_{3D} J^T$$

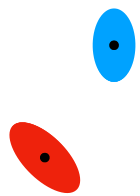
(S. Tulsiani, 2024)

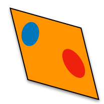
Differentiable Gaussian Rendering

What is the representation of a 3D gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?





$$\mu_{2D} = \pi(\mu_{3D})$$

$$\Sigma_{2D} = J \Sigma_{3D} J^T$$

1. Sort gaussians from closest to furthest from the camera

2. For each pixel **u**, compute opacity for each gaussian $\mathcal{G}_k$:

$$\bar{\alpha}_k = \alpha_k \frac{e^{-(\mathbf{u}-\mu_{2D}^k)^T (\Sigma_{2D}^k)^{-1} (\mathbf{u}-\mu_{2D}^k)}}{2\pi |\Sigma_{2D}^k|^{0.5}}$$

(In practice, can rasterize 'blocks' instead of entire image as not all gaussians influence all blocks)

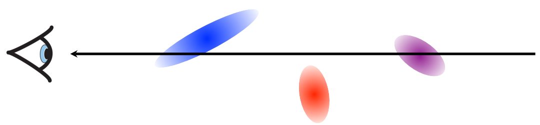
(S. Tulsiani, 2024)

Differentiable Gaussian Rendering

What is the representation of a 3D gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?



Compute per-gaussian weights based on opacities of current and previous gaussians:

$$w_k = \bar{\alpha}_k \, \Pi_{j=1}^{k-1}(1 - \bar{\alpha}_j)$$

Use per-gaussian SH coefficients and ray direction to get view-dependent color $\mathbf{c}_k$

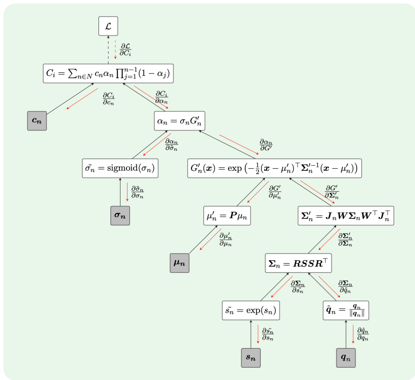
Aggregate to obtain pixel color:

$$\mathbf{c} = \sum_k w_k \mathbf{c}_k$$

(S. Tulsiani, 2024)

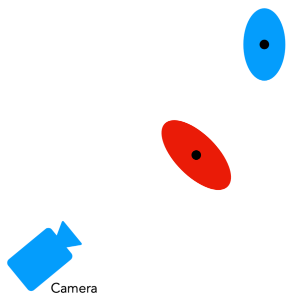
Computational Graph (gsplat)

- Forward (↑) and Backward (↓) Gaussian Splatting Rendering Function



Properties of Gaussians for Rendering

Gaussians are closed under affine transforms, integration



$$\mathcal{G}_{\mathbf{V}}(\mathbf{x}-\mathbf{p}) = \frac{1}{2\pi|\mathbf{V}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{p})^T \underset{\substack{\uparrow \\ \text{3D Covariance!}}}{\mathbf{V}^{-1}} (\mathbf{x}-\mathbf{p})}$$

Affine mapping $\Phi = \mathbf{M}\mathbf{x} + \mathbf{p}$ of coordinates (such as cam2world matrix!):

$$\mathcal{G}_{\mathbf{V}}(\Phi^{-1}(\mathbf{u}) - \mathbf{p}) = \frac{1}{|\mathbf{M}^{-1}|}\mathcal{G}_{\mathbf{M}\mathbf{V}\mathbf{M}^T}(\mathbf{u} - \Phi(\mathbf{p}))$$

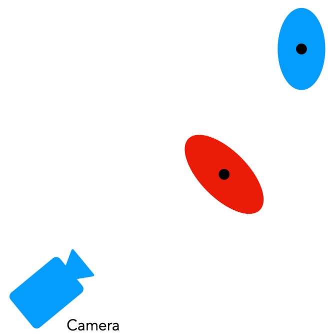
Integrate along axis:

$$\int_{\mathbb{R}} \mathcal{G}^3_{\mathbf{V}}(\mathbf{x}-\mathbf{p})\, dx_2 = \mathcal{G}^2_{\hat{\mathbf{V}}}(\hat{\mathbf{x}} - \hat{\mathbf{p}})$$

$$\mathbf{V} = \begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix} \Leftrightarrow \begin{pmatrix} a & b \\ b & d \end{pmatrix} = \hat{\mathbf{V}}$$

(V. Sitzmann, 2024)

Transform Gaussians into Camera Coordinates



$\boxed{\text{Cam2world}}$ is affine mapping $\phi(x) = \mathbf{W}\mathbf{x} + \mathbf{p}$:

$$\mathcal{G}_{\mathbf{V}''_k}(\varphi^{-1}(\mathbf{u}) - \mathbf{t}_k) = \frac{1}{|\mathbf{W}^{-1}|}\mathcal{G}_{\mathbf{V}'_k}(\mathbf{u} - \mathbf{u}_k) = r'_k(\mathbf{u})$$

$\boxed{\text{Projection}}$ $\mathbf{m}(u)$ is *not* an affine mapping :/

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \mathbf{m}(\mathbf{u}) = \begin{pmatrix} u_0/u_2 \\ u_1/u_2 \\ \|(u_0, u_1, u_2)^T\| \end{pmatrix}$$

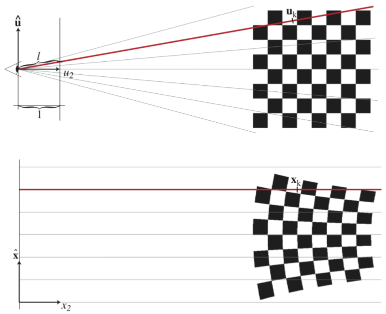
$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} = \mathbf{m}^{-1}(\mathbf{x}) = \begin{pmatrix} x_0/l \cdot x_2 \\ x_1/l \cdot x_2 \\ 1/l \cdot x_2 \end{pmatrix},$$

But can approximate with first-order Taylor Expansion as:

$$\mathbf{m}_{\mathbf{u}_k}(\mathbf{u}) = \mathbf{x}_k + \mathbf{J}_{\mathbf{u}_k} \cdot (\mathbf{u} - \mathbf{u}_k) \qquad \mathbf{J}_{\mathbf{u}_k} = \frac{\partial \mathbf{m}}{\partial \mathbf{u}}(\mathbf{u}_k)$$

(V. Sitzmann, 2024)

Transform Gaussians into Camera Coordinates



But can approximate with first-order Taylor Expansion as:

$$\mathbf{m}_{\mathbf{u}_k}(\mathbf{u}) = \mathbf{x}_k + \mathbf{J}_{\mathbf{u}_k} \cdot (\mathbf{u} - \mathbf{u}_k) \qquad \mathbf{J}_{\mathbf{u}_k} = \frac{\partial \mathbf{m}}{\partial \mathbf{u}}(\mathbf{u}_k)$$

Projected, 2D Gaussians are then:

$$\frac{1}{|\mathbf{W}^{-1}||\mathbf{J}^{-1}|}\mathcal{G}_{\mathbf{V}_k}(\mathbf{x} - \mathbf{x}_k)$$

$$\begin{aligned} \mathbf{V}_k &= \mathbf{J}\mathbf{V}'_k\mathbf{J}^T \\ &= \mathbf{J}\mathbf{W}\mathbf{V}''_k\mathbf{W}^T\mathbf{J}^T. \end{aligned}$$

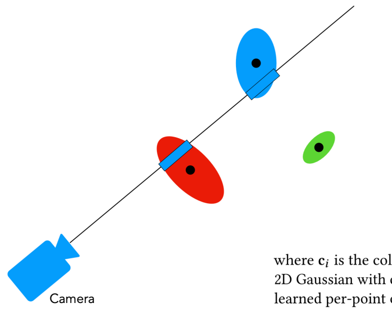
Finally, can integrate along rays:

$$\begin{aligned} q_k(\hat{\mathbf{x}}) &= \int_{\mathbb{R}} \frac{1}{|\mathbf{J}^{-1}||\mathbf{W}^{-1}|}\mathcal{G}_{\mathbf{V}_k}(\hat{\mathbf{x}} - \hat{\mathbf{x}}_k, x_2 - x_{k2})\, dx_2 \\ &= \frac{1}{|\mathbf{J}^{-1}||\mathbf{W}^{-1}|}\mathcal{G}_{\hat{\mathbf{V}}_k}(\hat{\mathbf{x}} - \hat{\mathbf{x}}_k) \end{aligned}$$

(V. Sitzmann, 2024)

Advantage of Rasterization

Can compute volume rendering integral without ever sampling a single 3D point in space!

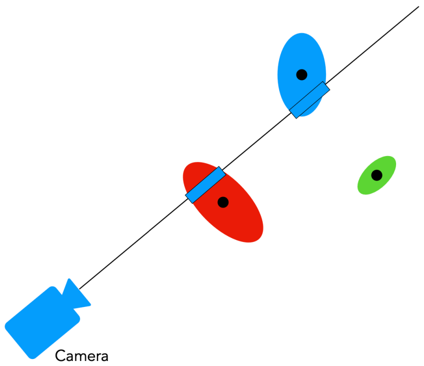


$$C = \sum_{i \in N} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \qquad (3)$$

where $\mathbf{c}_i$ is the color of each point and α_i is given by evaluating a 2D Gaussian with covariance Σ [Yifan et al. 2019] multiplied with a learned per-point opacity.

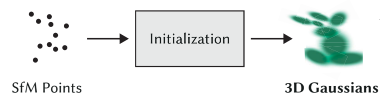
(V. Sitzmann, 2024)

Problem: Local minima…

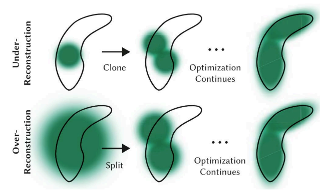


(V. Sitzmann, 2024)

Gaussian Splatting: Bells and Whistles



Fix 1: Initialize with sparse point cloud from SfM



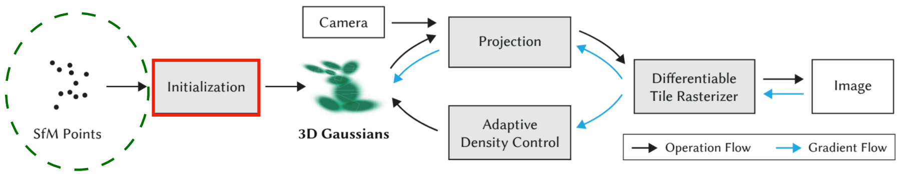
Fix 2: Split/clone gaussians based on heuristics

3D Gaussian Splatting for Real-Time Radiance Field Rendering. *Kerbl et. al.*

(S. Tulsiani, 2024)

Fix 1: Start from SFM point cloud.

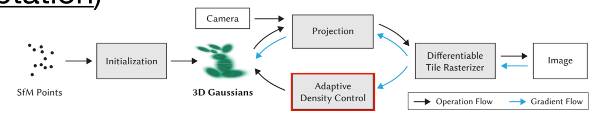
(Initialization)

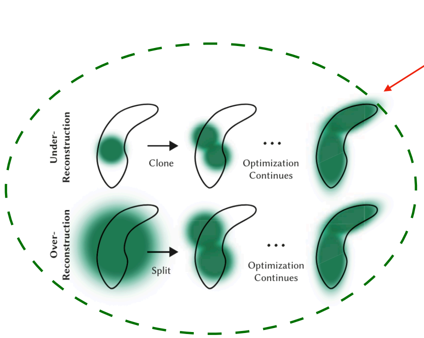


(V. Sitzmann, 2024)

Fix 2: Heuristic *pruning* and *spawning* operations

(Adaptation)





Our adaptive control of the Gaussians needs to populate empty areas. It focuses on regions with missing geometric features ("under-reconstruction"), but also in regions where Gaussians cover large areas in the scene (which often correspond to "over-reconstruction"). We observe that both have *large* view-space positional gradients. Intuitively, this is likely because they correspond to regions that are not yet well reconstructed, and the optimization tries to move the Gaussians to correct this.

Since both cases are good candidates for densification, we densify Gaussians with an average magnitude of view-space position gradients above a threshold τ_{pos}, which we set to 0.0002 in our tests.

(V. Sitzmann, 2024)

“… And Many More Details !”

– LV