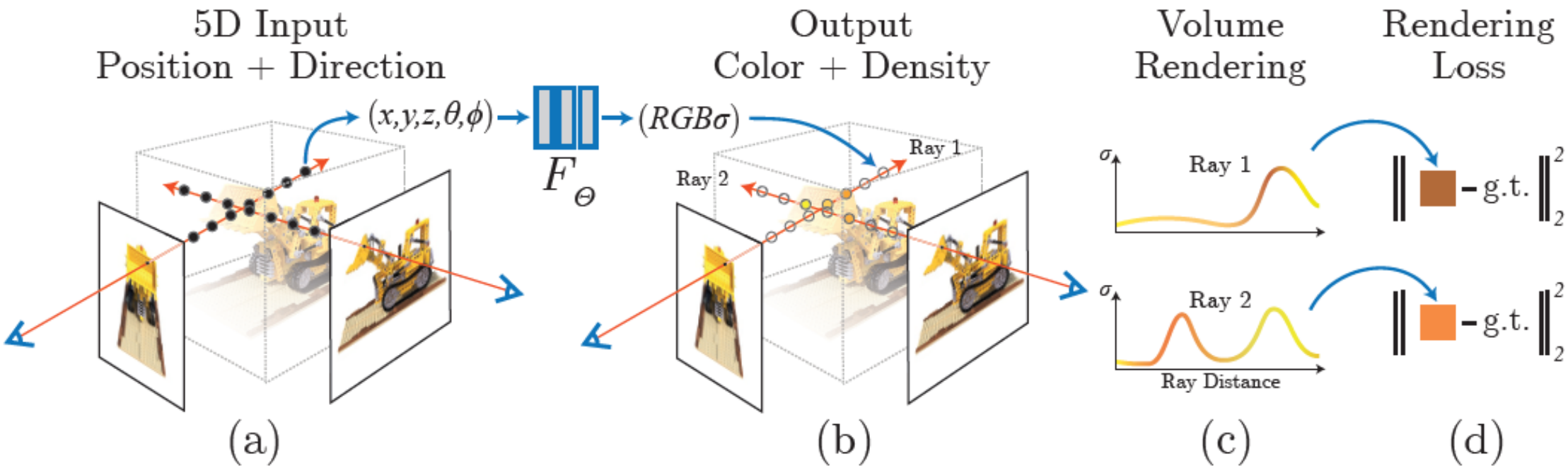
3D Gaussian Splatting for Real-Time Radiance Field Rendering

Xinze LI

Mar.20, 2024

Shor Recap for NeRF

Overview


5D Input
Position + Direction
(x,y,z,θ,ϕ)
F_Θ
Output
Color + Density
$(RGB\sigma)$
Ray 1
Ray 2
Volume
Rendering
Rendering
Loss
σ
Ray 1
Ray 2
Ray Distance
g.t.
g.t.
(a)
(b)
(c)
(d)

Key Points

- An **Overfitting** Neural Network (Singular)
- Volume rendering that is backpropagated.

- Sparse **voxel representation** of a 3D scene
- Learns mapping from 3D voxel position and camera direction to the color & opacity of voxels.

Drawbacks

- Significant cost of sampling & result in noise
- Lots of uninformative data: storing points for an empty space
- Marching Step limitations: detail quality
- Inefficient volume rendering: difficult to back-propagate.

Background

Challenges

- **Costly** to train and render neural networks achieving **high visual quality**.
  - Or low speed.
- Real time rendering
  - Unbounded and complete scenes (rather than isolated objects)
  - 1080p, $\geq$ 30 fps.

Advocation

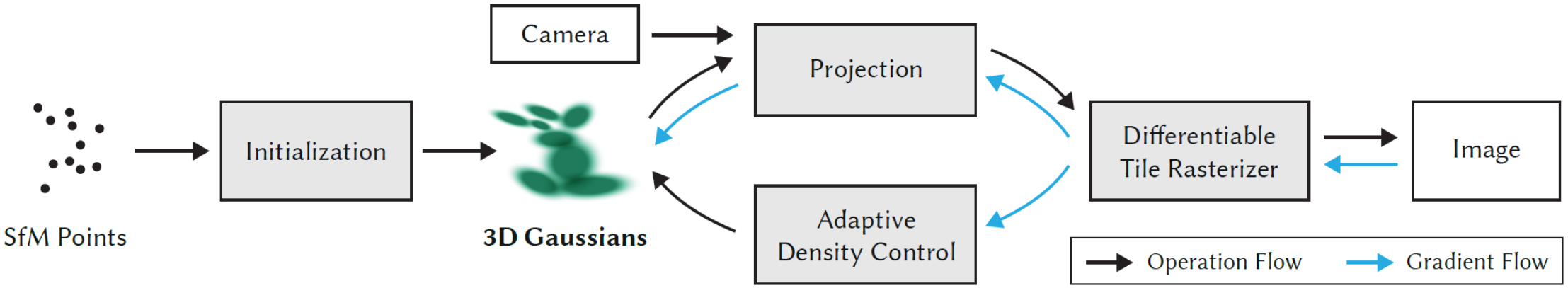
- **3D Gaussians representation method**
- **Fast visibility-aware rendering algorithm**

Method

Solution

1. **anisotropic 3D Gaussians** – High-quality, unstructured representation of radiance fields
2. An optimization method of 3D Gaussian properties, interleaved with **adaptive density control** that creates high-quality representations for captured scenes.
3. A **fast, differentiable rendering approach** for the GPU, which is **visibility-aware**, allows anisotropic splatting and fast back-propagation to achieve high-quality novel view synthesis.

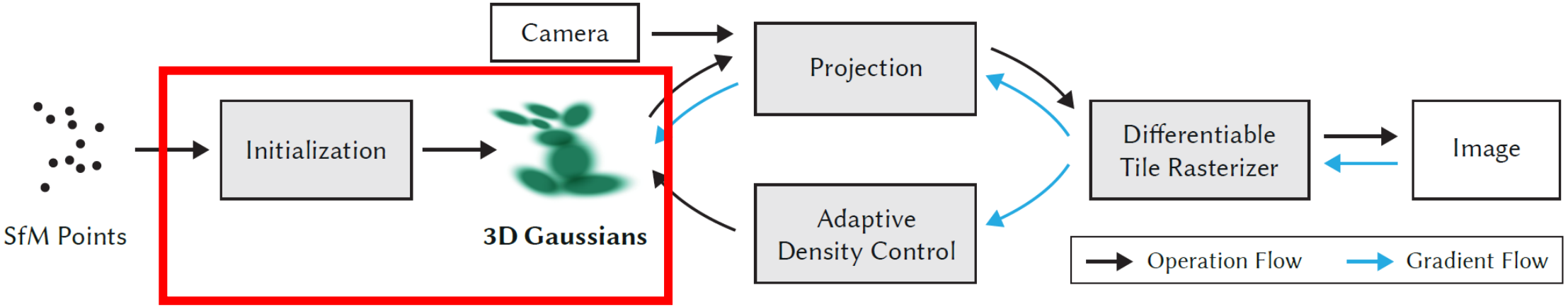
Overview


Camera
Projection
Initialization
Differentiable
Tile Rasterizer
Image
SfM Points
3D Gaussians
Adaptive
Density Control
Operation Flow
Gradient Flow

Overview

1. Step1: sparse SfM point cloud + Create a set of 3D Gaussians
   - Position (mean value)
   - Covariance matrix
   - Opacity α
2. Step2: Create the radiance field representation via a sequence of optimization steps of 3D Gaussian parameters, interleaved with operations for adaptive control of the Gaussian density

3D Gaussians: Initialization



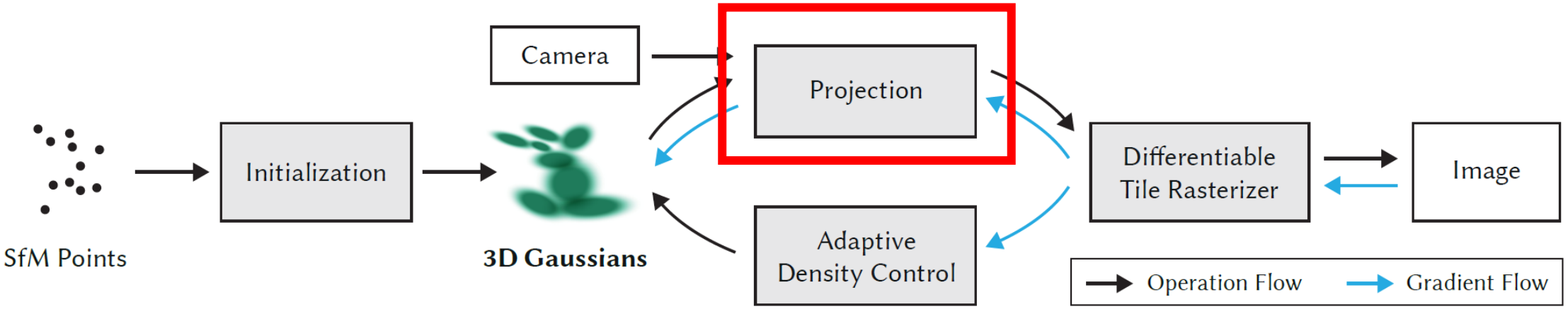
Method: Differentiable 3D Gaussian Splatting

- 3D Gaussians are defined by a full 3D covariance matrix Σ defined in world space centered at point (mean) μ :

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

- The relative density of that point in a Gaussian distribution, which is the probability of finding a data point at that point.

3D Gaussians: Projection



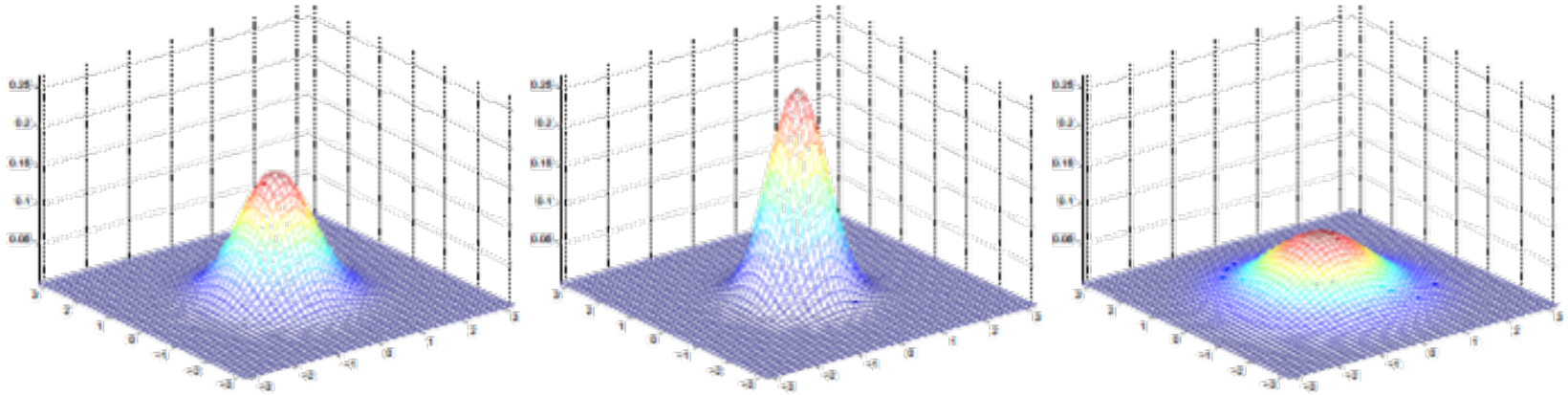
Method: Differentiable 3D Gaussian Splatting

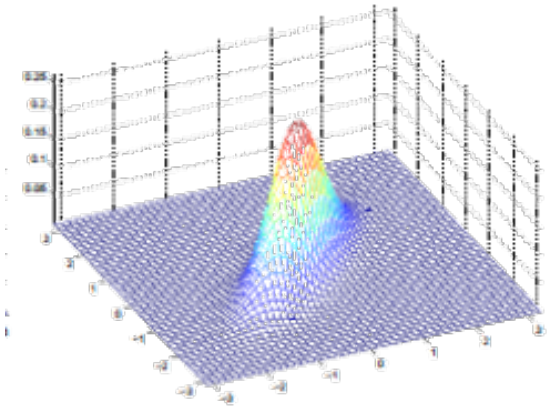
- Project 3D Gaussians to 2D for rendering: Given a viewing transformation W the covariance matrix Σ' in camera coordinates is given as follows:

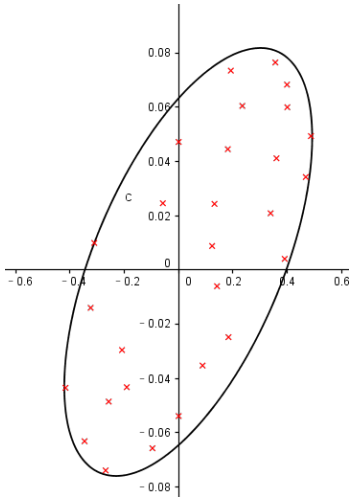
$$\Sigma' = JW\Sigma W^T J^T$$

- where J is the Jacobian of the affine approximation of the projective transformation.
- *Challenge: Covariance matrices have physical meaning only if they are semi-definite.* $\Leftrightarrow \forall v \neq \mathbf{0}, s.t. v^T \Sigma v \geq 0$

Method: Differentiable 3D Gaussian Splatting







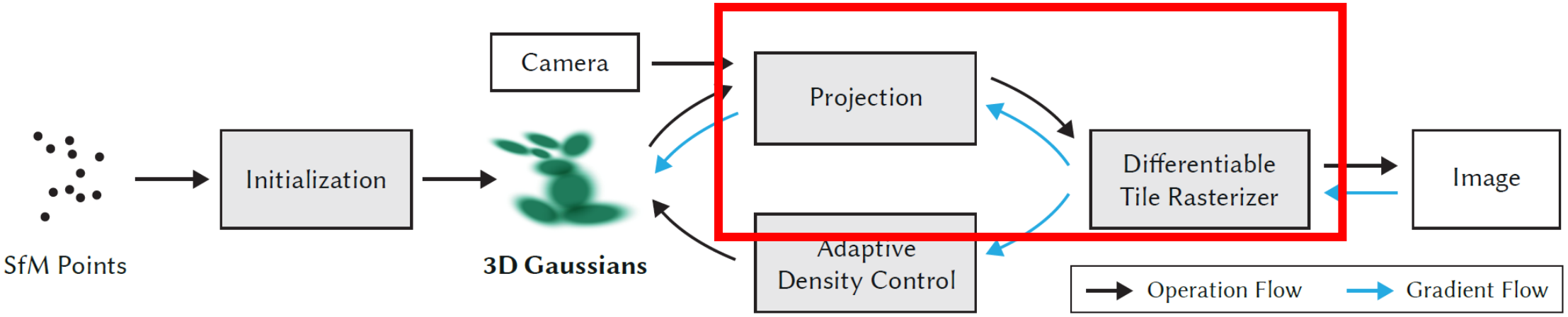
Method: Differentiable 3D Gaussian Splatting

- 3D Gaussian is analogous to describing the configuration of an **ellipsoid**
- Given a scaling matrix S and rotation matrix R, we can find the corresponding Σ:

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

- Hint: S is a diagonal matrix whose diagonal element is provided by 3D vector s $\rightarrow$ Semi-definite.
- R is represented by quaternion q , describing the rotation between the main axis of a three-dimensional Gaussian distribution and the standard coordinate axis, which is $\perp$

3D Gaussians: Projection



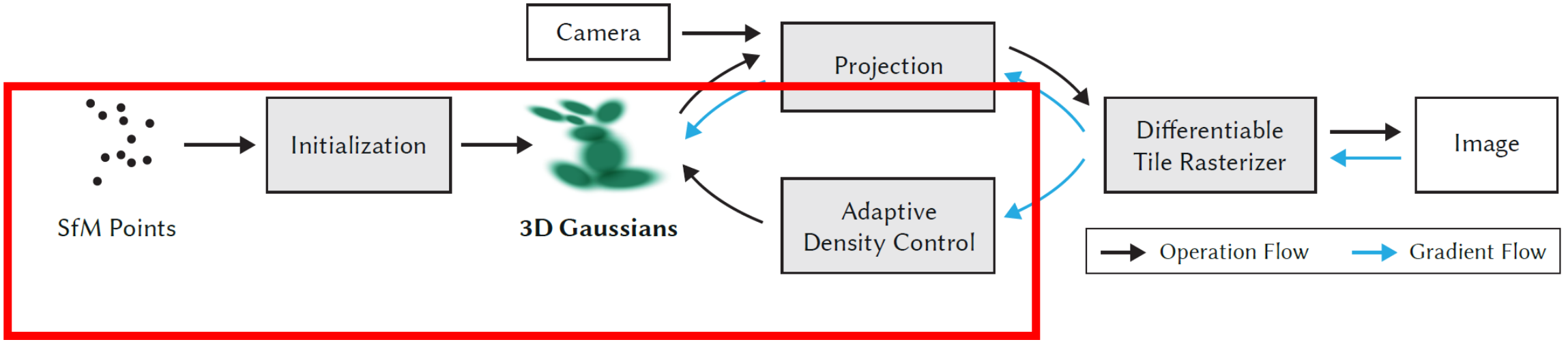
Method: Differentiable Tile Rasterizer

Algorithm 2 GPU software rasterization of 3D Gaussians
w, h: width and height of the image to rasterize
M, S: Gaussian means and covariances in world space
C, A: Gaussian colors and opacities
V: view configuration of current camera

```
function RASTERIZE(w, h, M, S, C, A, V)
    CullGaussian(p, V)                          ▷ Frustum Culling      1
    M', S' ← ScreenspaceGaussians(M, S, V)      ▷ Transform
    T ← CreateTiles(w, h)                                              2
    L, K ← DuplicateWithKeys(M', T)             ▷ Indices and Keys
    SortByKeys(K, L)                            ▷ Globally Sort
    R ← IdentifyTileRanges(T, K)
    I ← 0                                       ▷ Init Canvas          3
    for all Tiles t in I do
        for all Pixels i in t do
            r ← GetTileRange(R, t)
            I[i] ← BlendInOrder(i, L, r, K, M', S', C, A)
        end for                                                        4
    end for
    return I
end function
```

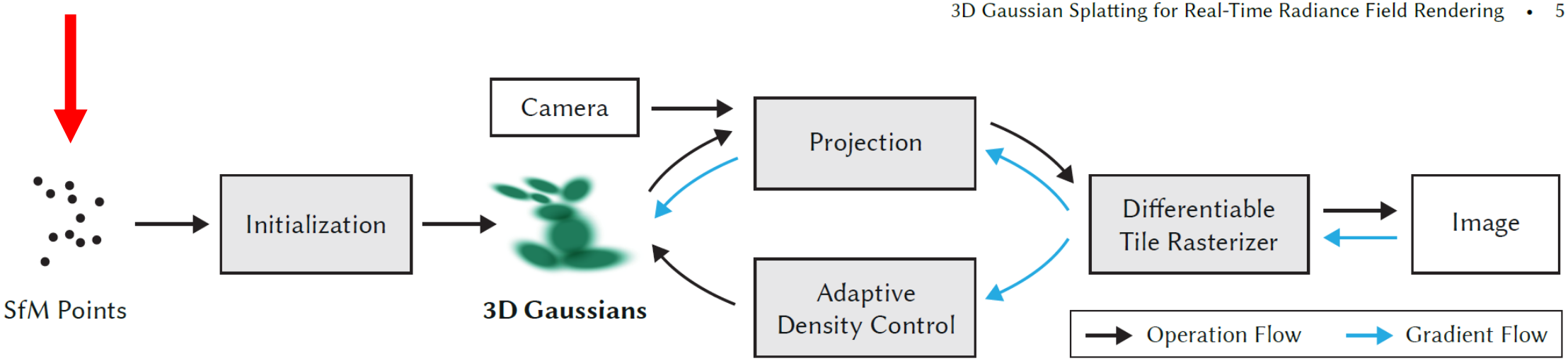
1. Collect splats that are visible (confidence $\geq 99\%$) through view frustrum.
2. Project splats and split the screen into 16×16 tiles
3. Associate splats with tiles by tile-ID and sort them using a single fast GPU Radix sort. Stop when all pixels have sutured (i.e. $a \rightarrow 1$)
4. Process all pixels in a tile perform a-blending, by traversing ordered list of splats.

3D Gaussians: Adaptive Density Control



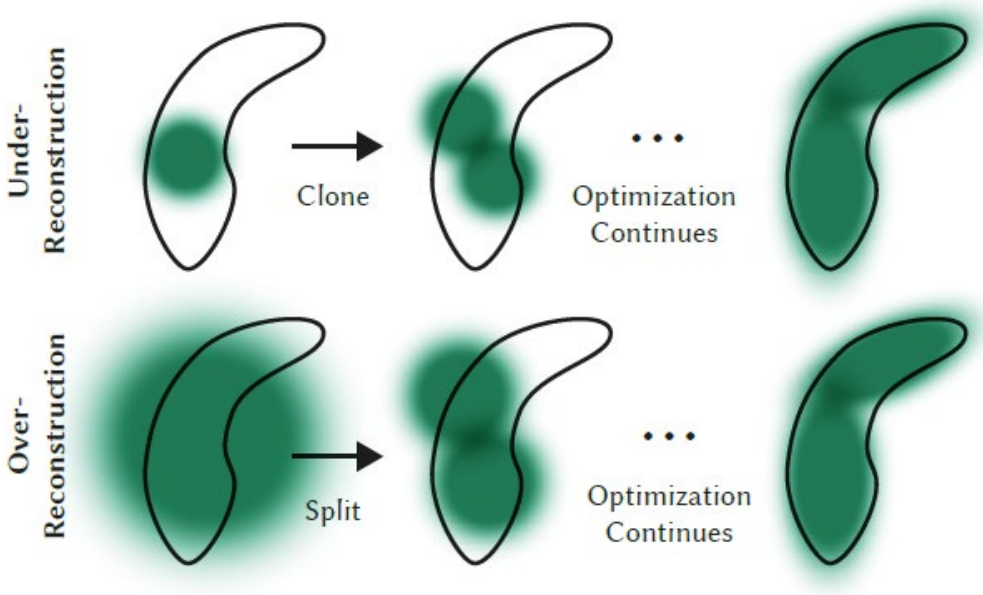
Method: Adaptive Density Control

- Sparse points from Structure from Motion (SfM).
- The need for adaptive control to achieve a denser and more accurate representation.



Method: Adaptive Density Control

- Regions with missing geometric features
  - **Under-Reconstruction**: when small-scale geometry (black outline) is insufficiently covered, they clone the respective Gaussian
  - **Over-Reconstruction:** if small-scale geometry is represented by one large splat, they split it in two.



Method: Adaptive Density Control

- Targeting Gaussians in areas with large view-space positional gradients, indicating regions needing better reconstruction.
- **Criteria for Densification**: Gaussians with an average magnitude of view-space position gradients above a threshold $\tau_{pos} = 2 \times 10^{-4}$ are candidates for densification.

Method: Adaptive Density Control

- Other Tricks:
  - Densifying every 100 iterations and remove any Gaussians that are essentially transparent. (with α threshold)
  - Removing Gaussians that are very large in worldspace and those that have a big footprint in viewspace

Flow overview

- The loss function is L1 combined with a D-SSIM term ($\lambda = 0.2$ was used)

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

Optimizes

- 3D Gaussian parameters:
  - Mean $\mu \in \mathbb{R}^{3\times1}$
  - full 3D covariance matrix $\Sigma \in \mathbb{R}^{3\times3}$
  - Scaling $s \in \mathbb{R}^{3\times1}$
  - Quaternion $q \in \mathbb{R}^{4\times1}$
  - Spherical harmonics (color) $SH \in \mathbb{R}^{3\times9}$
  - Opacity $\alpha \in \mathbb{R}^{1}$

Experiments

Implementation Details

- custom CUDA kernels for rasterization
- NVIDIA CUB sorting routines for the fast Radix sort
- an A6000 GPU

Quantitative evaluation

Table 1. Quantitative evaluation of our method compared to previous work, computed over three datasets. Results marked with dagger † have been directly adopted from the original paper, all others were obtained in our own experiments.

Dataset	Mip-NeRF360						Tanks&Temples						Deep Blending					
Method|Metric	*SSIM*↑	*PSNR*↑	*LPIPS*↓	Train	FPS	Mem	*SSIM*↑	*PSNR*↑	*LPIPS*↓	Train	FPS	Mem	*SSIM*↑	*PSNR*↑	*LPIPS*↓	Train	FPS	Mem
Plenoxels	0.626	23.08	0.463	25m49s	6.79	2.1GB	0.719	21.08	0.379	25m5s	13.0	2.3GB	0.795	23.06	0.510	27m49s	11.2	2.7GB
INGP-Base	0.671	25.30	0.371	5m37s	11.7	13MB	0.723	21.72	0.330	5m26s	17.1	13MB	0.797	23.62	0.423	6m31s	3.26	13MB
INGP-Big	0.699	25.59	0.331	7m30s	9.43	48MB	0.745	21.92	0.305	6m59s	14.4	48MB	0.817	24.96	0.390	8m	2.79	48MB
M-NeRF360	0.792†	27.69†	0.237†	48h	0.06	8.6MB	0.759	22.22	0.257	48h	0.14	8.6MB	0.901	29.40	0.245	48h	0.09	8.6MB
Ours-7K	0.770	25.60	0.279	6m25s	160	523MB	0.767	21.20	0.280	6m55s	197	270MB	0.875	27.78	0.317	4m35s	172	386MB
Ours-30K	0.815	27.21	0.214	41m33s	134	734MB	0.841	23.14	0.183	26m54s	154	411MB	0.903	29.41	0.243	36m2s	137	676MB

More memory used!!!

Ablation study

Table 3. PSNR Score for ablation runs. For this experiment, we manually downsampled high-resolution versions of each scene's input images to the established rendering resolution of our other experiments. Doing so reduces random artifacts (e.g., due to JPEG compression in the pre-downscaled Mip-NeRF360 inputs).

	Truck-5K	Garden-5K	Bicycle-5K	Truck-30K	Garden-30K	Bicycle-30K	Average-5K	Average-30K
Limited-BW	14.66	22.07	20.77	13.84	22.88	20.87	19.16	19.19
Random Init	16.75	20.90	19.86	18.02	22.19	21.05	19.17	20.42
No-Split	18.31	23.98	22.21	20.59	26.11	25.02	21.50	23.90
No-SH	22.36	25.22	22.88	24.39	26.59	25.08	23.48	25.35
No-Clone	22.29	25.61	22.15	24.82	27.47	25.46	23.35	25.91
Isotropic	22.40	25.49	22.81	23.89	27.00	24.81	23.56	25.23
Full	22.71	25.82	23.18	24.81	27.70	25.65	23.90	26.05

Ablation study

- Above: Initialization with a random point cloud.
- Below: Initialization using SfM points.

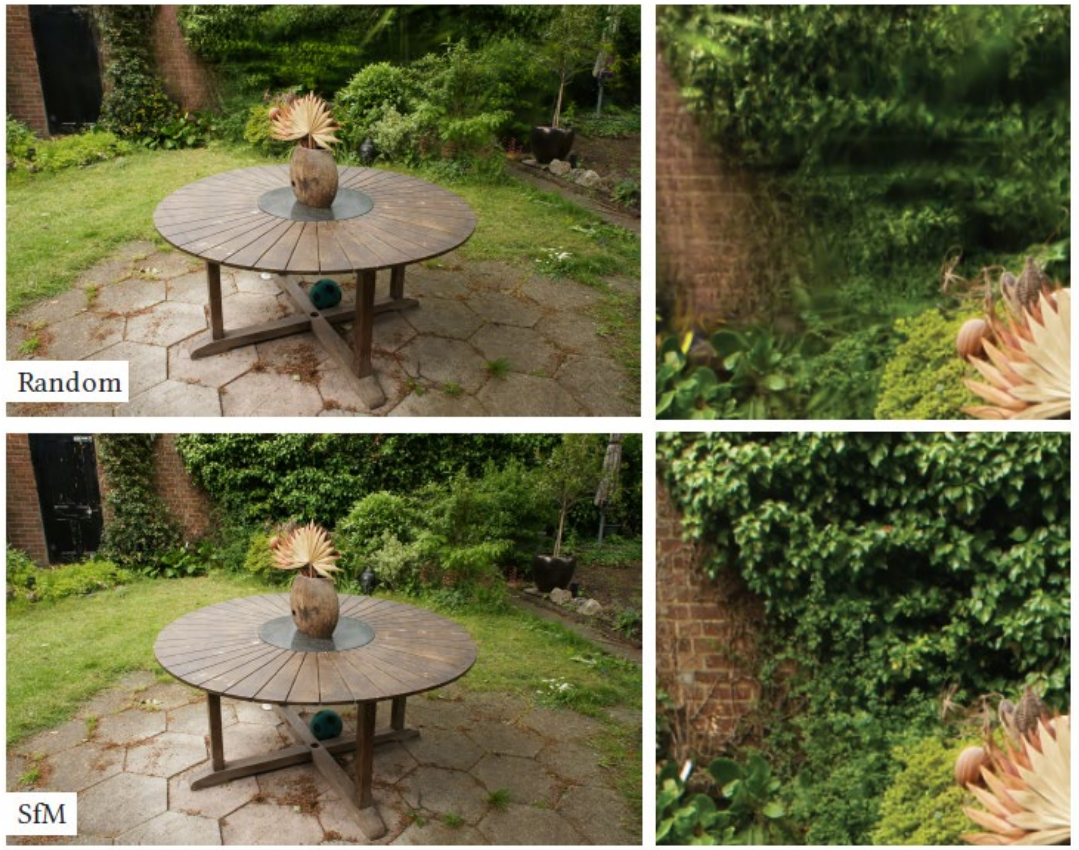
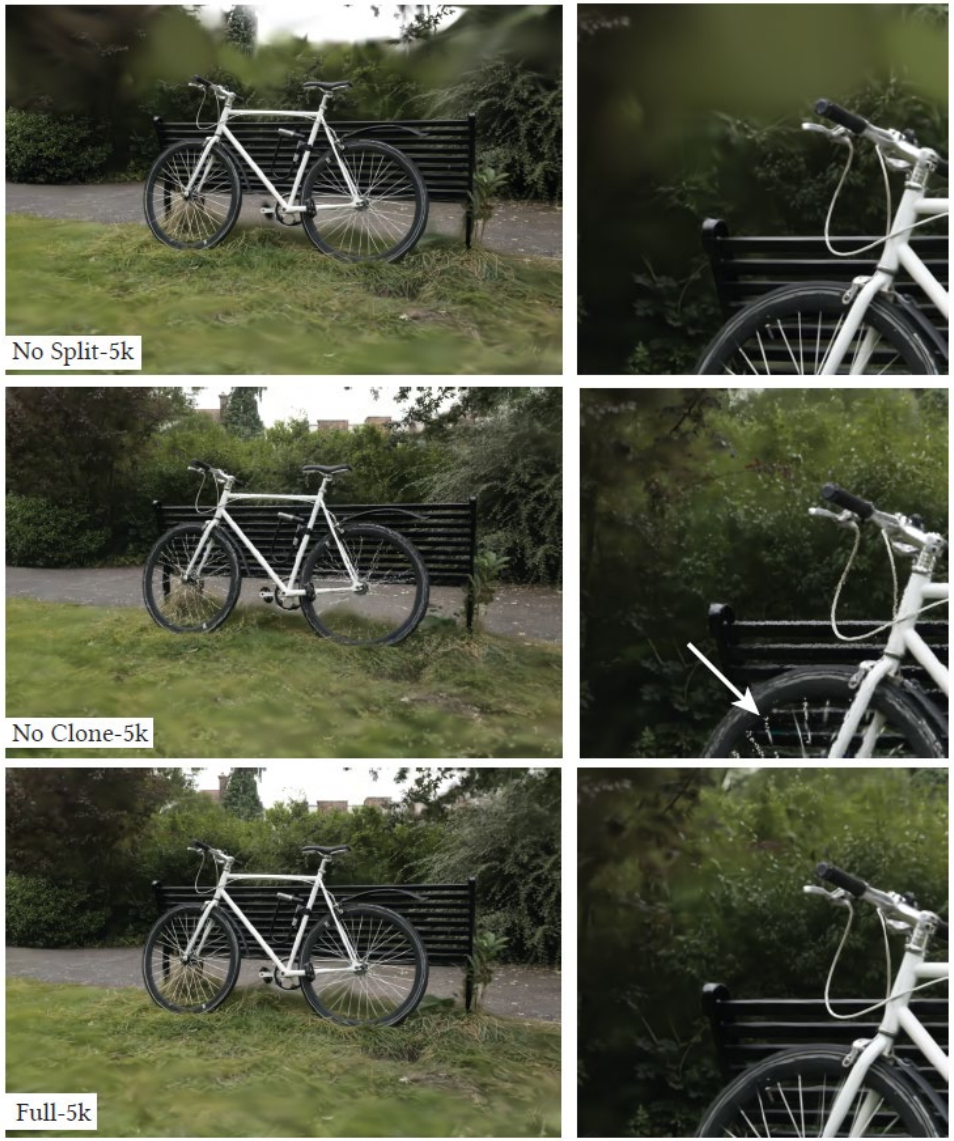


Fig. 7. Initialization with SfM points helps. Above: initialization with a random point cloud. Below: initialization using SfM points.

Ablation study

- No Split
- No Clone
- Full



Ablation study

Fig. 9. If we limit the number of points that receive gradients, the effect on visual quality is significant. Left: limit of 10 Gaussians that receive gradients. Right: our full method.

Ablation study

- Full covariance matrix → anisotropy (directional variance)
- Comparing a version that only uses a single scalar to control the radius uniformly across all three axes (removing anisotropy) with the full model

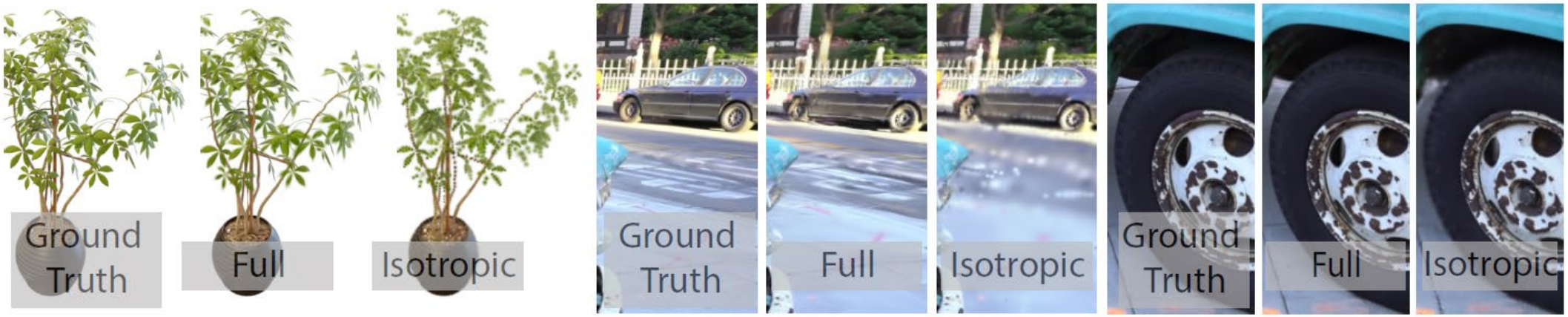


Fig. 10. We train scenes with Gaussian anisotropy disabled and enabled. The use of anisotropic volumetric splats enables modelling of fine structures and has a significant impact on visual quality. Note that for illustrative purposes, we restricted Ficus to use no more than 5k Gaussians in both configurations.

Limitations

Limitations

- Artifacts in regions not well observed, common to other methods as well.
  （观察不足的区域的伪影）
- Creation of elongated artifacts or "splotchy" Gaussians in some cases.
  （各向异性高斯的伪影）
- Occasional popping artifacts when large Gaussians are created, especially in areas with view-dependent appearance.
  （弹出伪影）
- Simple visibility algorithm leading to sudden changes in depth/blending order of Gaussians.
  （简单的可见性算法）

Limitations

- No application of regularization in the optimization process, which could mitigate unseen region and popping artifacts.
  （未应用正则化技术）
- Significant memory consumption compared to NeRF-based solutions, with peak GPU memory usage exceeding 20 GB during training of large scenes in the unoptimized prototype.
  （内存消耗）
- Need for hyperparameter adjustment for large scene convergence, as observed in early experiments.
  （减少内存消耗的机会）

Conclusion

Conclusions

- 3D Gaussians is a excellent method for scene representation due to:
  - A differentiable volumetric representation
  - Can be easily projected to 2D splats
  - Allowing fast α-blending for rendering
- Utilizing Gaussians for a scene representation allows to avoid unnecessary computation in empty space and achieve an accurate representation of the scene.
- The differentiable tile rasterization approach implemented as optimized CUDA kernels is a key factor for the performance of both training and real-time rendering.

Conclusions

- It has a higher memory cost compared to other approaches.
- This is **the first approach that truly allows both real-time and high-quality rendering**, while requiring training times competitive with the fastest existing solutions.

Q&A