INTRO
PRINCIPLES
APPLICATIONS

# 3D GAUSSIAN SPLATTING
## SURVEY

# OUTLINE

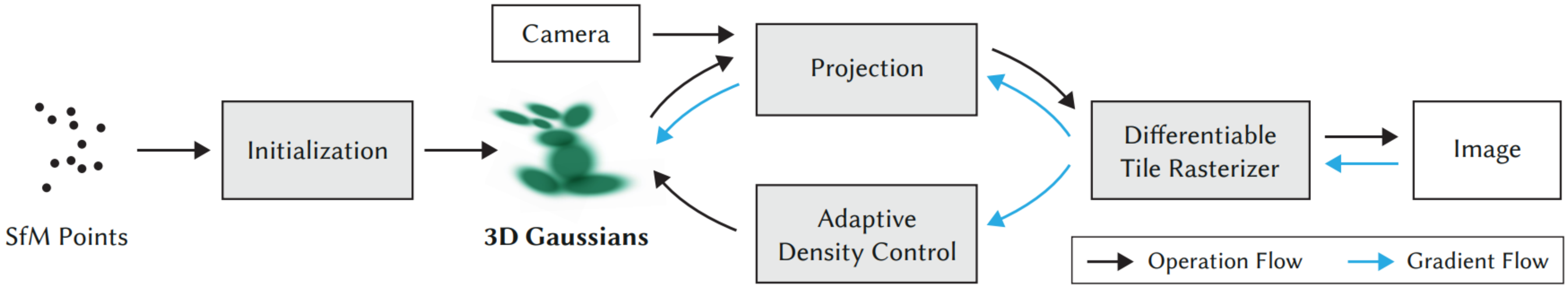
1. Preview
2. 3D Gaussian Splatting Model
   1. Pipeline
   2. Initialization
   3. Properties
   4. Rendering
   5. Training
3. Comparison with NeRF
4. Applications and Possibilities


2025


File
Edit
Window
Help
Import...
Start Training
Create Camera
Render
Default Camera
Scene
Preferences
Log
Scene
Default Camera
Image Set
Rdnc Field
I ♥ HTCV
2025
Hugo
Parameters
Rdnc Field
Train
Info
Edit
Render
Model Profile
Splat3
Downsample Images
848
[px]
Max Splat Count
3000
[kSplats]
Anti-Aliasing
Treat Zero Alpha as Mask
Max Sph. Hrm. Degree
3
Create Sky Model
Region of Interest
Translate
1.11992
Rotate
14.0786
Scale
5.24078
Stop Training After
64
[kSteps]
Store Training Context
Reset Radiance Field
0
239
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230

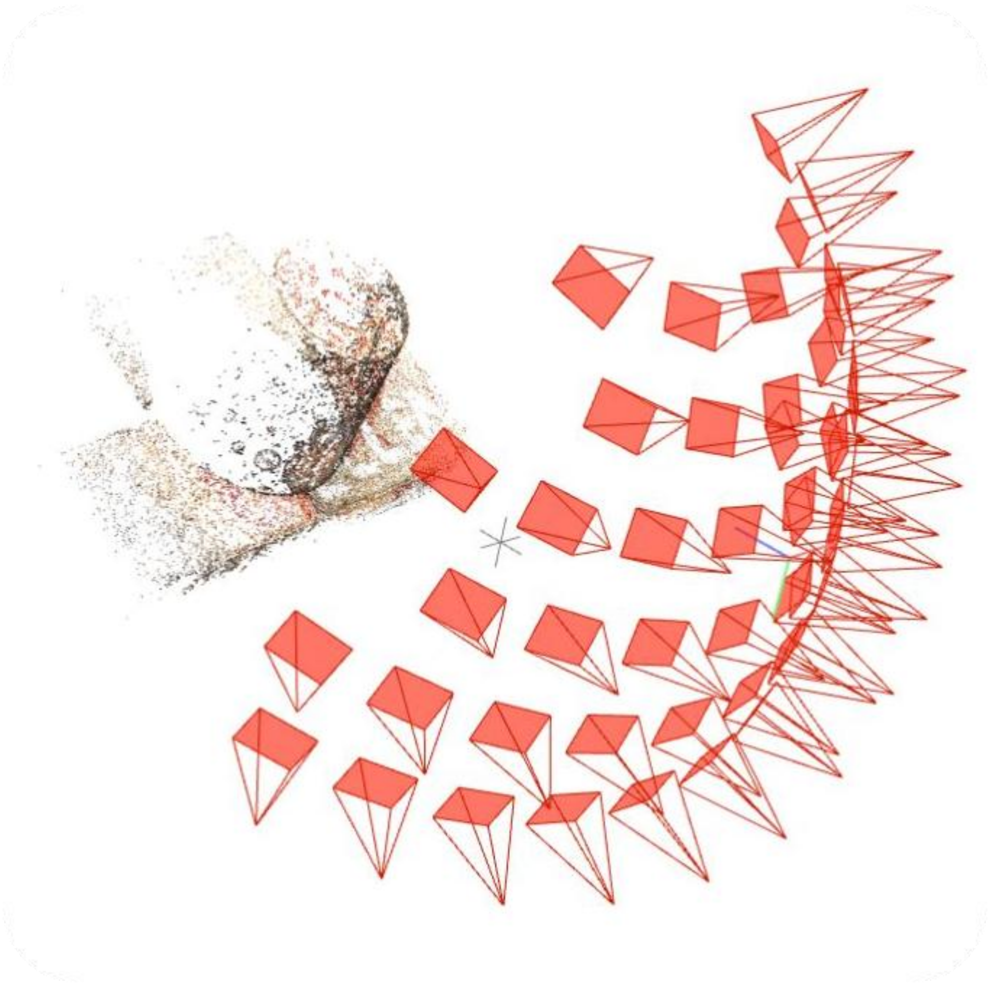
# 3D GAUSSIAN SPLATTING MODEL

## 1. PIPELINE



# 3D GAUSSIAN SPLATTING MODEL
# 2. INITIALIZATION

- Apply Structure from Motion on input images to initialize model parameters
- A widely used SfM pipeline is COLMAP
- COLMAP yields necessary camera configurations
- Additionally, detected key-points by make great initial positions for 3D Gaussians



# 3D GAUSSIAN SPLATTING MODEL
# 2. INITIALIZATION

- Apply Structure from Motion on input images to initialize model parameters

- A widely used SfM pipeline is COLMAP

- COLMAP yields necessary camera configurations

- Additionally, detected key-points by make great initial positions for 3D Gaussians


# 3D GAUSSIAN SPLATTING MODEL
# 2. INITIALIZATION
# KEYPOINTS
# CAMERA CONFIGURATIONS

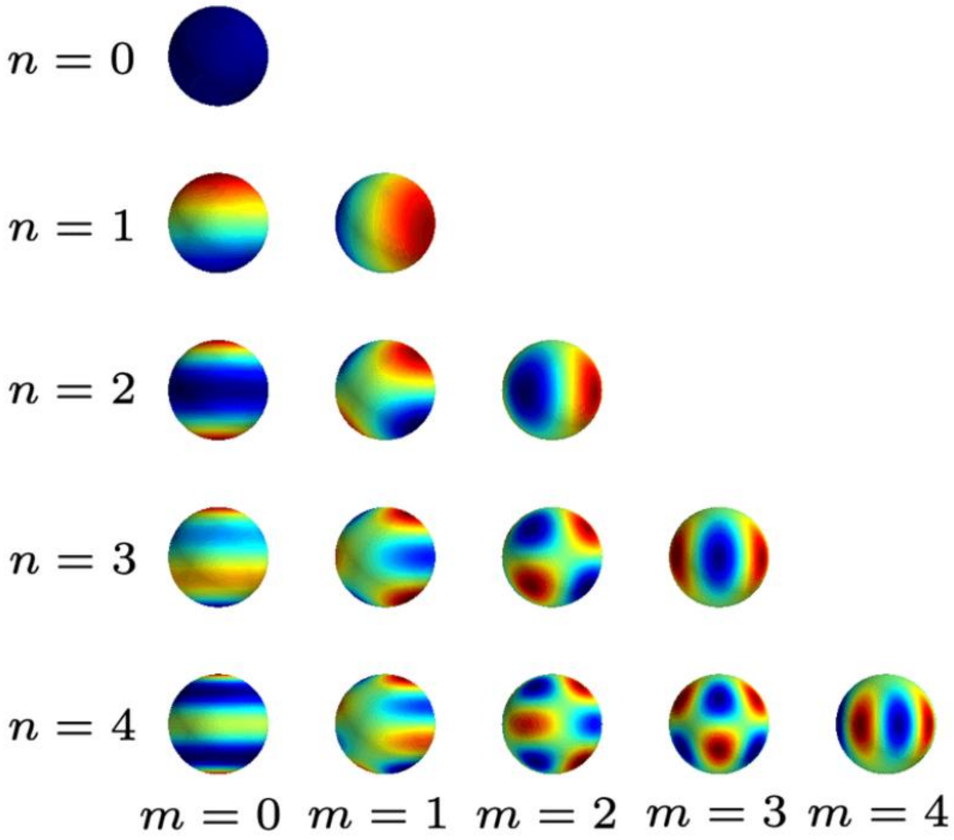
# 3D GAUSSIAN SPLATTING MODEL
## 3. PROPERTIES

3D Gaussians has following learnable parameters:

- Position $\mu$
- Covariance $\Sigma$
- Opacity $\alpha$
- Color $c$
    - Color represented via Spherical Harmonics

# 3D GAUSSIAN SPLATTING MODEL

## 3. PROPERTIES (SPHERICAL HARMONICS)




$$c(v) = \sum_{n=0}^{N} \sum_{m=0}^{n} c_{n,m} \cdot \mathrm{SH}(v)$$

- Color is dependent on view angle
- Calculated from coefficient and Spherical Harmonic

# 3D GAUSSIAN SPLATTING MODEL

## 4. RENDERING

Key components of 3D Gaussian Splatting Rendering:

- Frustum Culling: Given a camera pose, this determine Gaussians that are inside the camera's frustum
- Splatting: Project 3D Gaussians into 2D space
- Calculating color: Pixel color calculation through $\alpha$-blending
- Tiles: Dividing the image into non-overlapping 16x16 pixel tiles
- Parallelism: Process tiles with sorted list of Gaussians

# 3D GAUSSIAN SPLATTING MODEL

## 4. RENDERING

How to project a 3D Gaussian to 2D rendering?

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

- $\sum$ is the covariance of the Gaussian
- W is the view transformation
- J is the Jacobian of the affine approximation of the projective transformaion

Covariance loses pyhsical meaning if optimized blindly, therefore:

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

- Expressed in form of scale $s$ and quaternion $q$

# 3D GAUSSIAN SPLATTING MODEL

## 4. RENDERING

So, which color do my pixels have?

$$C = \sum_{n=1}^{|\mathcal{N}|} c_n \alpha'_n \prod_{j=1}^{n-1} (1 - \alpha'_j)$$

- $c_n$ is the learned color
- $\alpha'_n$ is the final opacity

# 3D GAUSSIAN SPLATTING MODEL

## 4. RENDERING

So, which color do my pixels have?

$$C = \sum_{n=1}^{|\mathcal{N}|} c_n \alpha'_n \prod_{j=1}^{n-1} (1 - \alpha'_j)$$

- $c_n$ is the learned color
- $\alpha'_n$ is the final opacity

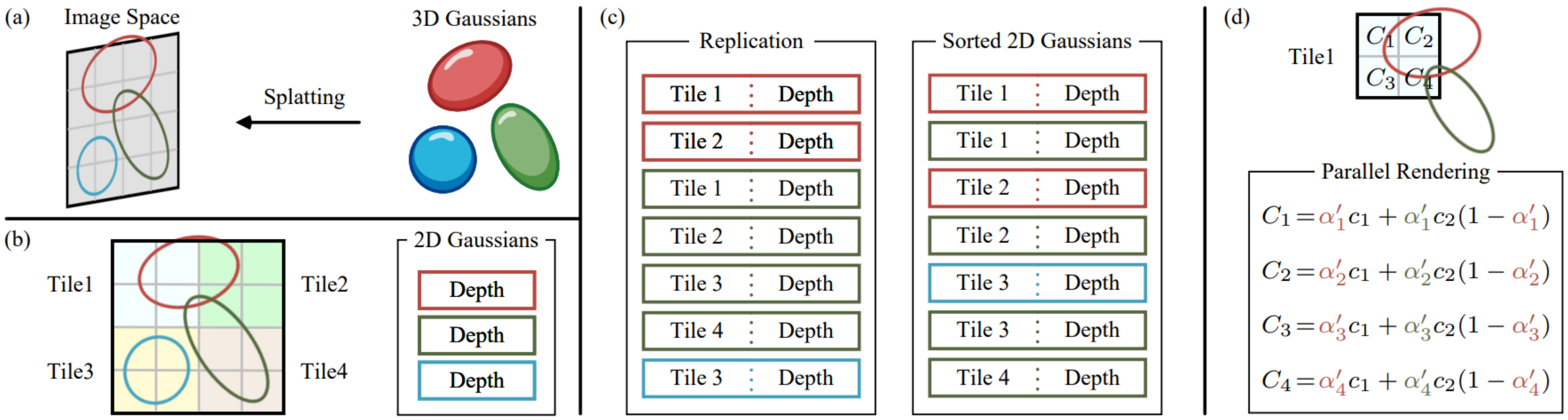
The final opacity is the multiplication result of learned opacity and gaussian.

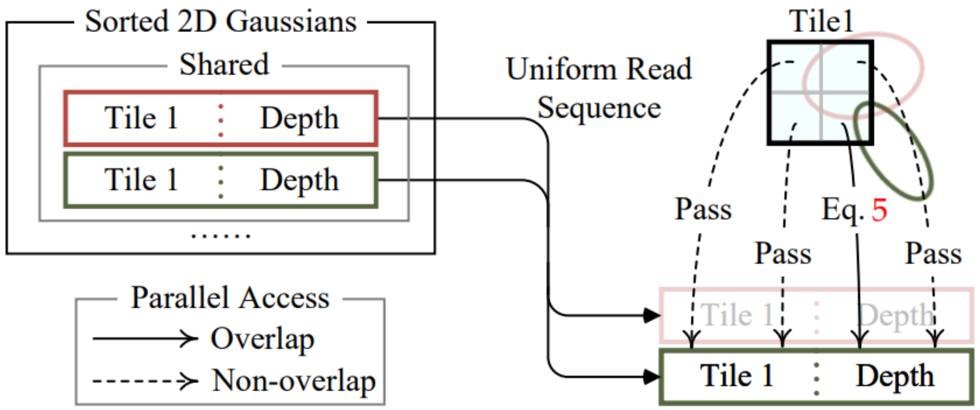
$$\alpha'_n = \alpha_n \times \exp\left(-\frac{1}{2}(\boldsymbol{x}' - \boldsymbol{\mu}'_n)^\top \boldsymbol{\Sigma}'^{-1}_n (\boldsymbol{x}' - \boldsymbol{\mu}'_n)\right)$$

- $\alpha_n$ is the learned opacity

# 3D GAUSSIAN SPLATTING MODEL

## 4. RENDERING



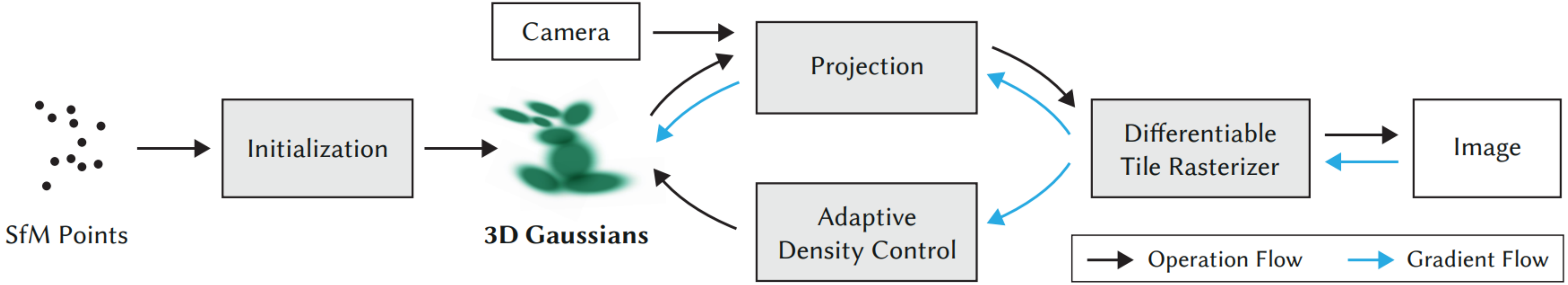




Equation 5:

$$C = \sum_{n=1}^{|\mathcal{N}|} c_n \alpha'_n \prod_{j=1}^{n-1} (1 - \alpha'_j)$$

# 3D GAUSSIAN SPLATTING MODEL

## 5. TRAINING

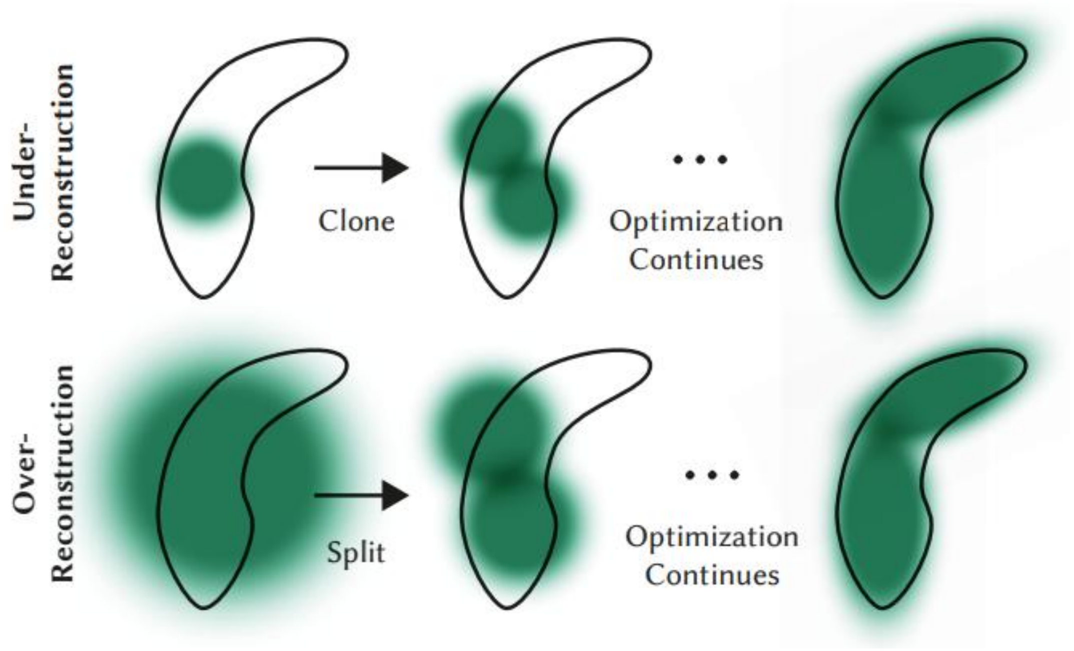



Loss function:

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

# 3D GAUSSIAN SPLATTING MODEL

## 5. TRAINING

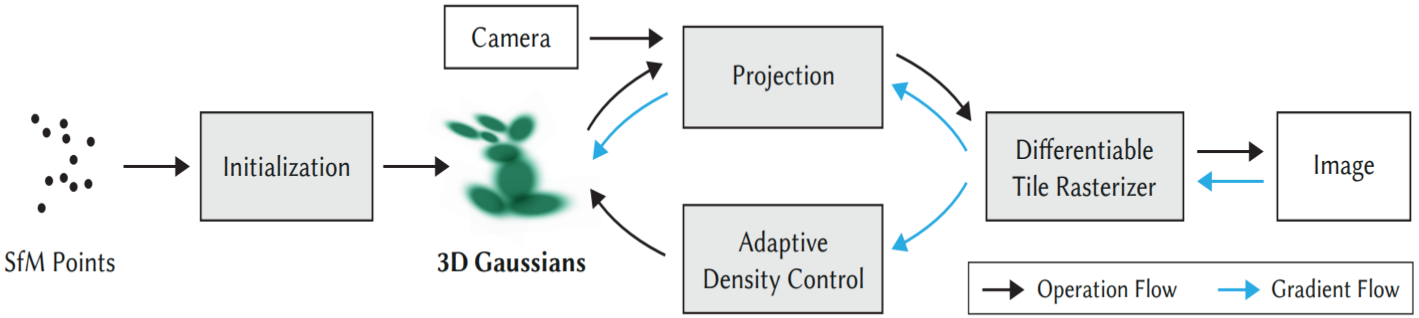



$M, S, C, A \leftarrow \text{Adam}(\nabla L)$ ▷ Backprop & Step
**if** IsRefinementIteration($i$) **then**
    **for all** Gaussians $(\mu, \Sigma, c, \alpha)$ **in** $(M, S, C, A)$ **do**
        **if** $\alpha < \epsilon$ or IsTooLarge$(\mu, \Sigma)$ **then** ▷ Pruning
            RemoveGaussian()
        **end if**
        **if** $\nabla_p L > \tau_p$ **then** ▷ Densification
            **if** $\|S\| > \tau_S$ **then** ▷ Over-reconstruction
                SplitGaussian$(\mu, \Sigma, c, \alpha)$
            **else** ▷ Under-reconstruction
                CloneGaussian$(\mu, \Sigma, c, \alpha)$
            **end if**
        **end if**
    **end for**
**end if**

# COMBINING EVERYTHING TOGETHER

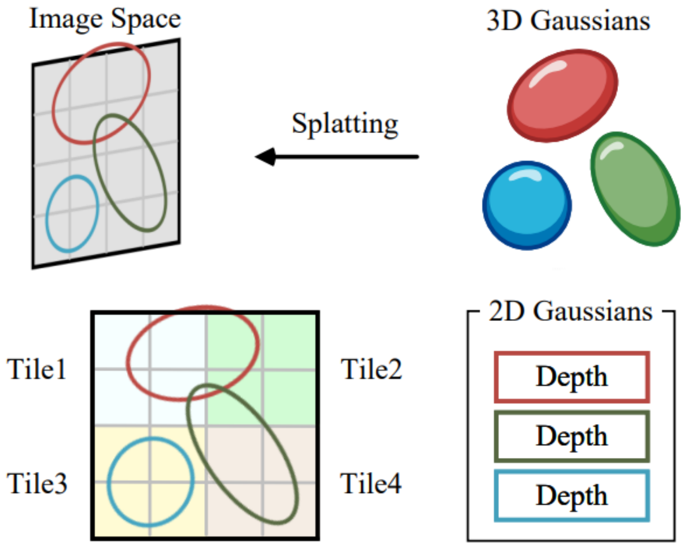
**Algorithm 1** Optimization and Densification
$w, h$: width and height of the training images

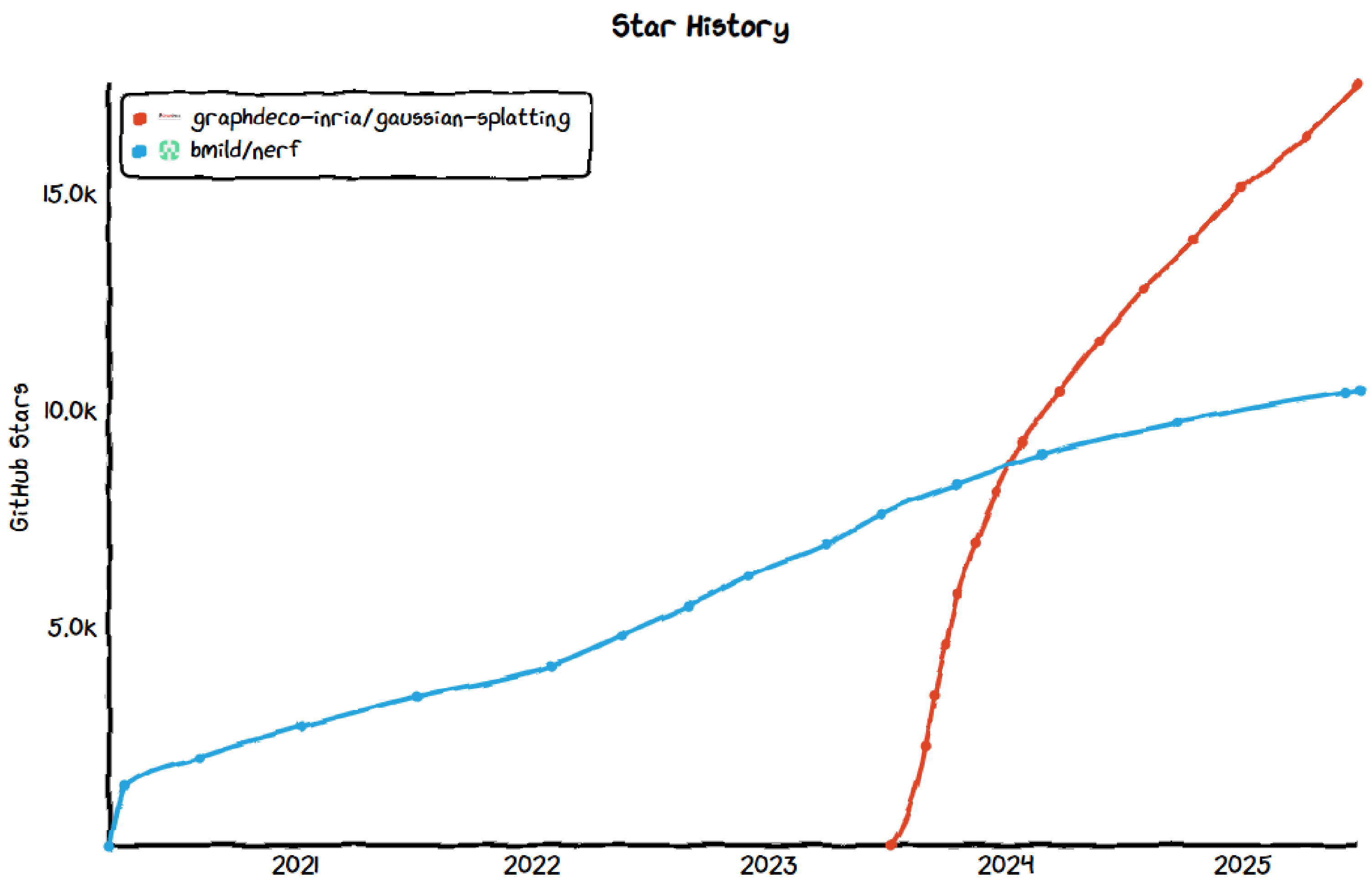
$M \leftarrow$ SfM Points ▷ Positions
$S, C, A \leftarrow$ InitAttributes() ▷ Covariances, Colors, Opacities
$i \leftarrow 0$ ▷ Iteration Count
**while** not converged **do**
  $V, \hat{I} \leftarrow$ SampleTrainingView() ▷ Camera $V$ and Image
  $I \leftarrow$ Rasterize($M, S, C, A, V$) ▷ Alg. 2
  $L \leftarrow Loss(I, \hat{I})$ ▷ Loss
  $M, S, C, A \leftarrow$ Adam($\nabla L$) ▷ Backprop & Step
  **if** IsRefinementIteration($i$) **then**
    **for all** Gaussians $(\mu, \Sigma, c, \alpha)$ **in** $(M, S, C, A)$ **do**
      **if** $\alpha < \epsilon$ or IsTooLarge($\mu, \Sigma$) **then** ▷ Pruning
        RemoveGaussian()
      **end if**
      **if** $\nabla_p L > \tau_p$ **then** ▷ Densification
        **if** $\|S\| > \tau_S$ **then** ▷ Over-reconstruction
          SplitGaussian($\mu, \Sigma, c, \alpha$)
        **else** ▷ Under-reconstruction
          CloneGaussian($\mu, \Sigma, c, \alpha$)
        **end if**
      **end if**
    **end for**
  **end if**
  $i \leftarrow i + 1$
**end while**




**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
  $M', S' \leftarrow$ ScreenspaceGaussians($M, S, V$) ▷ Transform
  $T \leftarrow$ CreateTiles($w, h$)
  $L, K \leftarrow$ DuplicateWithKeys($M', T$) ▷ Indices and Keys
  SortByKeys($K, L$) ▷ Globally Sort
  $R \leftarrow$ IdentifyTileRanges($T, K$)
  $I \leftarrow \mathbf{0}$ ▷ Init Canvas
  **for all** Tiles $t$ **in** $I$ **do**
    **for all** Pixels $i$ **in** $t$ **do**
      $r \leftarrow$ GetTileRange($R, t$)
      $I[i] \leftarrow$ BlendInOrder($i, L, r, K, M', S', C, A$)
    **end for**
  **end for**
  **return** $I$
**end function**






# COMPARISON WITH NERF

# COMPARISON WITH NERF

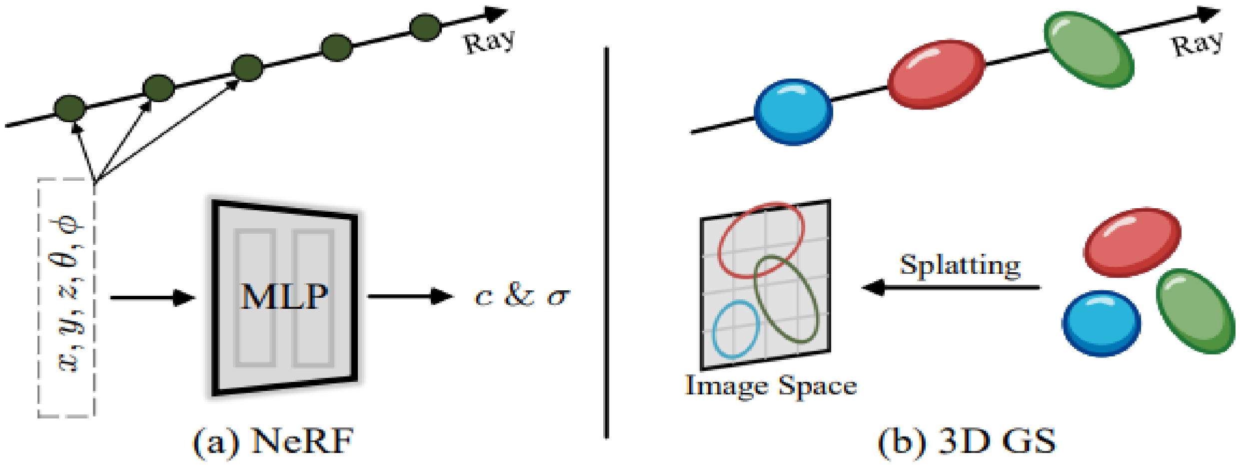
| Dataset | Mip-NeRF360 | | | | | | Tanks&Temples | | | | | | Deep Blending | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method\|Metric | *SSIM*↑ | *PSNR*↑ | *LPIPS*↓ | Train | FPS | Mem | *SSIM*↑ | *PSNR*↑ | *LPIPS*↓ | Train | FPS | Mem | *SSIM*↑ | *PSNR*↑ | *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 |

TABLE I: Comparison of compression on MipNeRF360 [20].

| Method | PSNR↑ | SSIM↑ | LPIPS↓ | Size (MB)↓ |
|---|---|---|---|---|
| 3DGS | 27.49 | **0.813** | **0.222** | 744.7 |
| Scaffold-GS [16] | 27.50 | 0.806 | 0.252 | 253.9 |
| HAC [19] | **27.53** | 0.807 | 0.238 | **15.26** |
| Compact-3DGS [21] | 27.08 | 0.798 | 0.247 | 48.80 |
| EAGLES [10] | 27.15 | 0.808 | 0.238 | 68.89 |
| LightGaussian [15] | 27.00 | 0.799 | 0.249 | 44.54 |
| Gaussian-SLAM [18] | 26.01 | 0.772 | 0.259 | 23.90 |
| Compact3d [11] | 27.16 | 0.808 | 0.228 | 50.30 |

- Storage cost is one of the biggest disadvantages of 3D Gaussian Splatting
- Researhers tried out various stategies to reduce this cost
  - Anchors
  - Codebooks
  - Hash Grids
  - Efficient pruning strategies
  - Efficient Gaussian representations or attributes

# COMPARISON WITH NERF




- NeRF samples along the ray and then queries the MLP to obtain corresponding colors and densities, which can be seen as a ***backward*** mapping (ray tracing)
- 3D GS projects all 3D Gaussians into the image space (splatting) and then performs parallel rendering, which can be viewed as a ***forward*** mapping (rasterization)

# COMPARISON WITH NERF

Is NeRF obsolete?

- Both methods have fundamental differences in their approaches. 3D GS is explicit, NeRF is implicit
- At first glance 3D GS may seem superior, but NeRF enthusiasts claim advantages over 3D GS:
  - NeRF generalizes scenes better and more compact
  - Areas with a very low sample data have better results with NeRF
  - Transparent areas have better results through NeRF

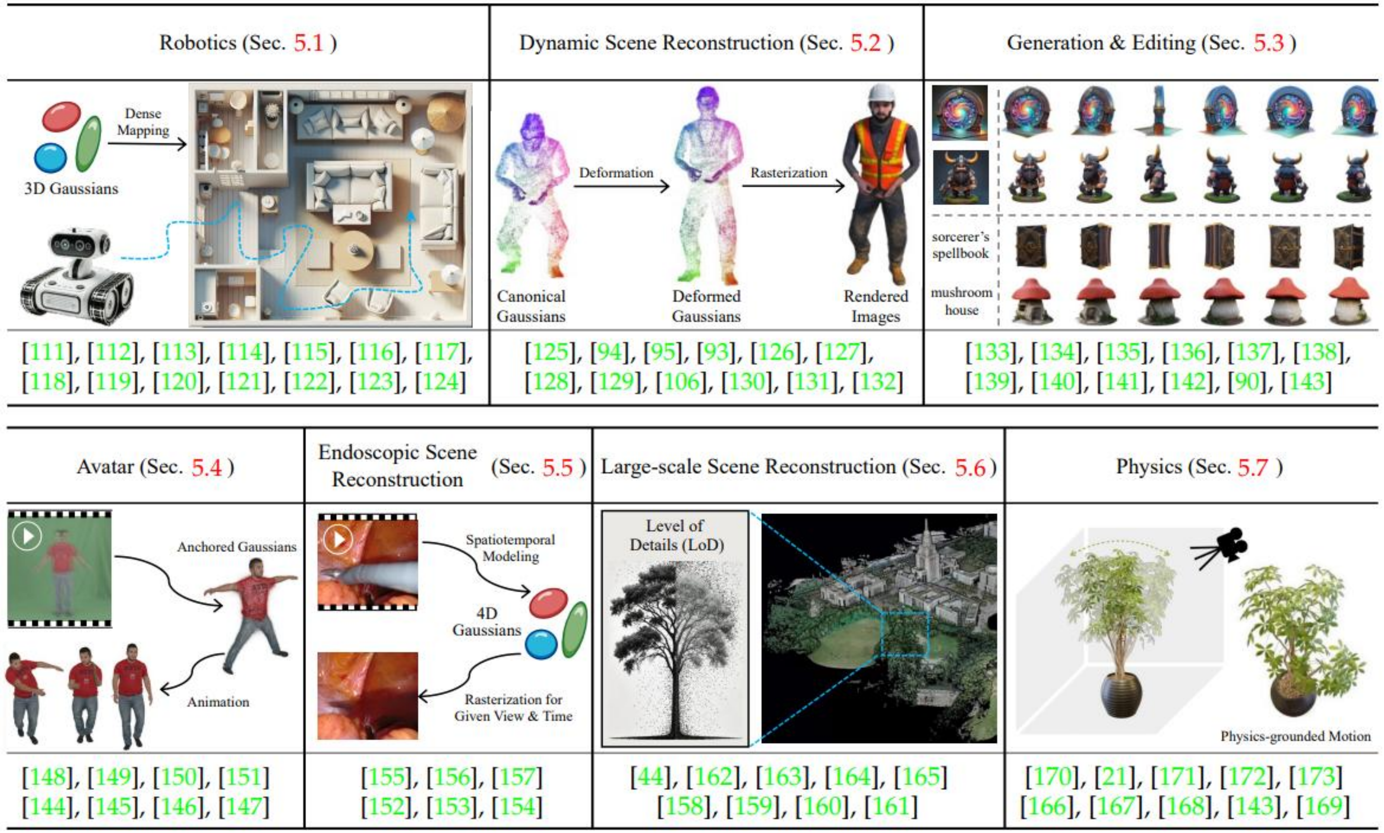
Hard to judge these claims, maybe we can test some of these for the final paper!

- The biggest strengths of 3D GS are:
  - Training and rendering speed (but InstantNPG (5 sec) and FastNeRF (200+ FPS) has competitive performance)
  - Gaussians are easy to understand and edit due to it‘s explicit nature
  - Various use cases by enhancing Gaussians with e.g. language features

# APPLICATIONS AND POSSIBILITIES

## Appilications:

- Robotics
- Dynamic Scene Reconstruction
- Generation and Editing
- Avatar
- Endoscopic Scene Reconstruction
- Large-scale Scene Reconstruction
- Physics



# APPLICATIONS AND POSSIBILITIES

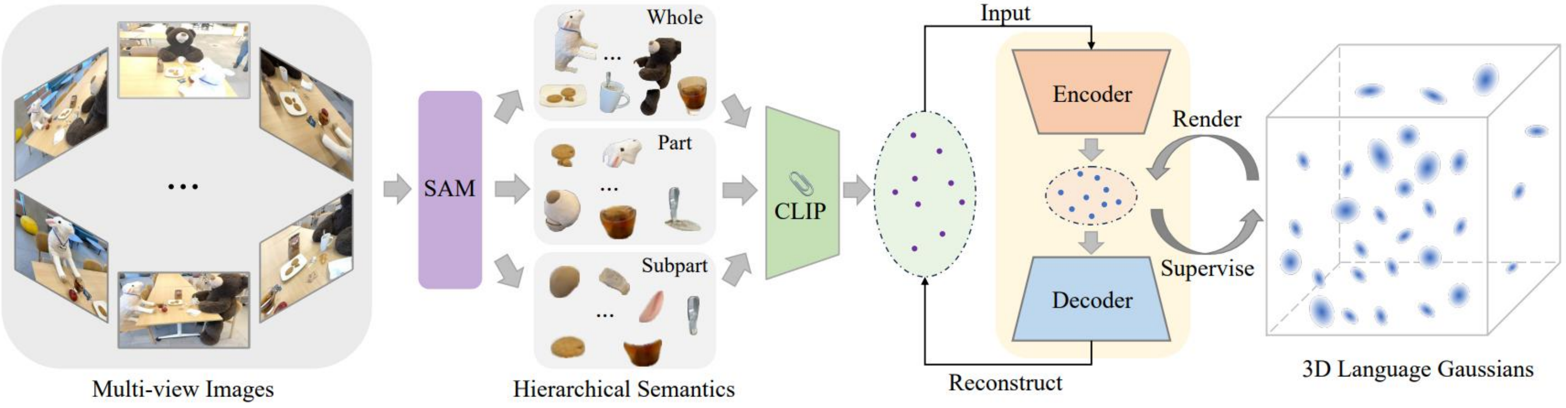
Gaussians can be enhanced by combining additional attributes:

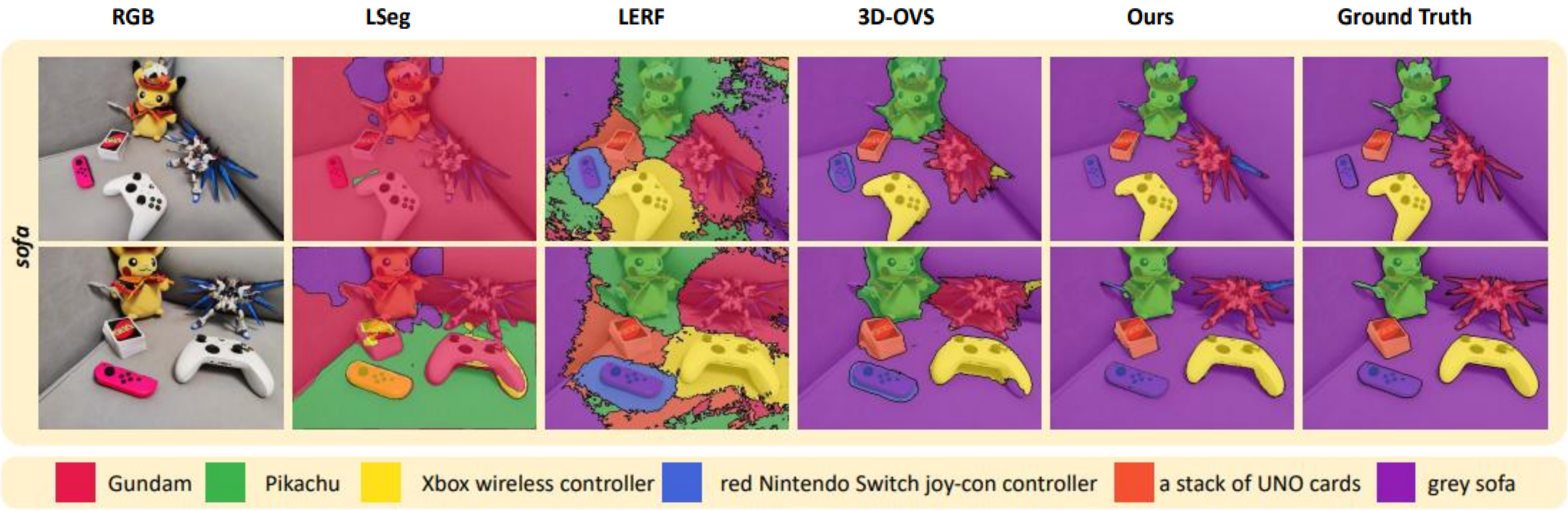
- Semantic Attributes
- Attribute Distributions
- Temporal Attributes
- Displacement Attributes
- Physical Attributes
- Discrete Attributes
- Inferred Attributes
- Weight Attributes
- And more…

# APPLICATIONS AND POSSIBILITIES

- Combines Gaussians with Language Features
- Encodes language features from CLIP to present a language field
- Supports language queries

## LangSplat: 3D Gaussian Language Splatting

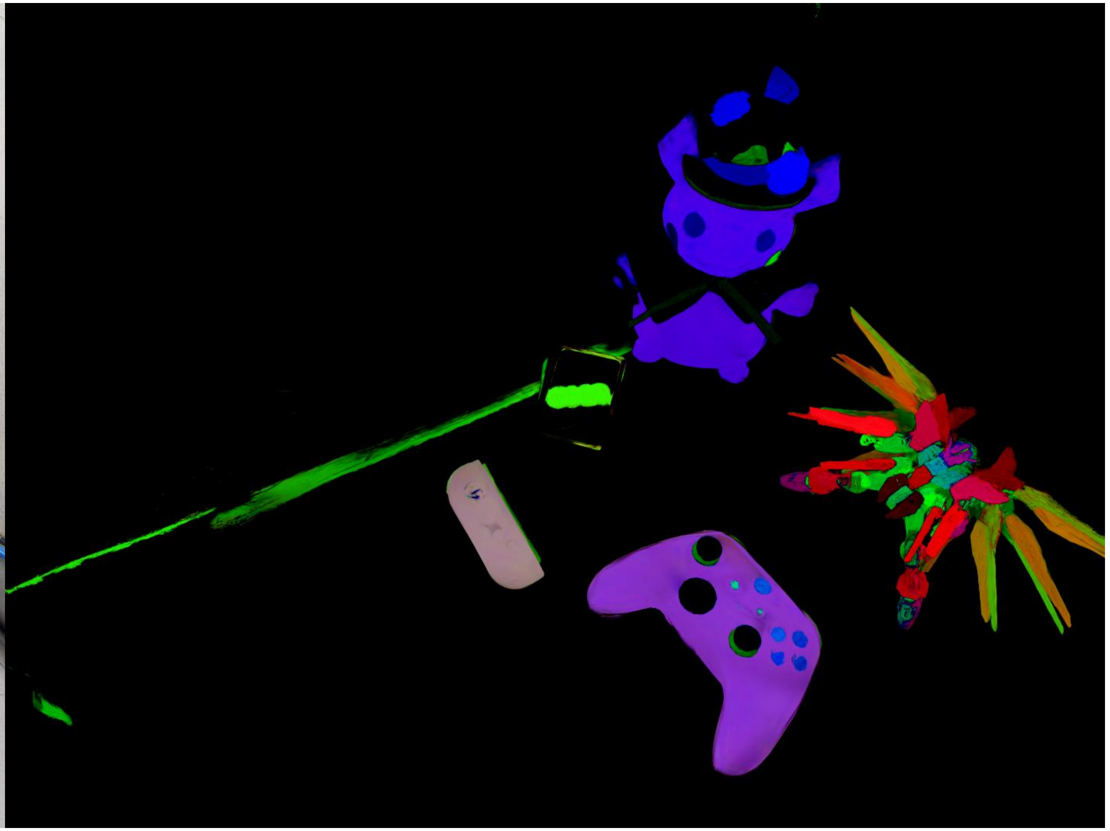


# APPLICATIONS AND POSSIBILITIES
# LANGSPLAT - FEATURE LEVEL 1

Input:

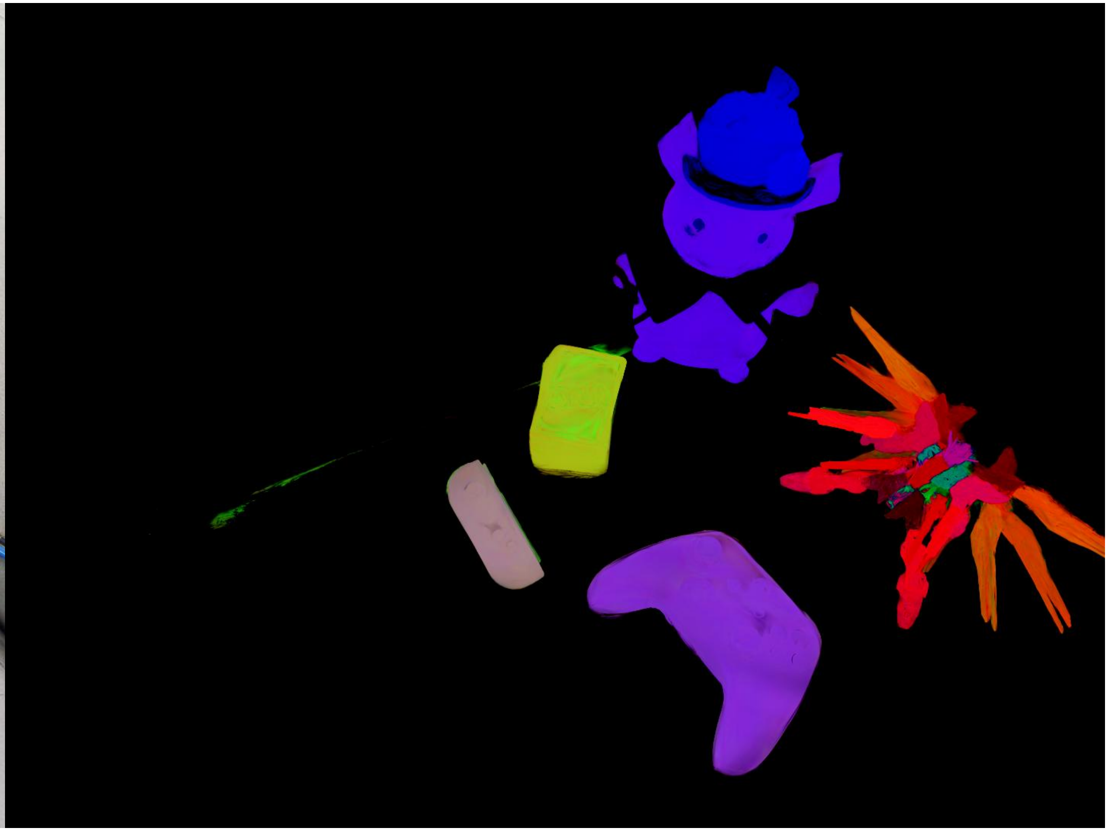
Segmentation Feature Level 1:



# APPLICATIONS AND POSSIBILITIES
# LANGSPLAT - FEATURE LEVEL 2

Input:


Segmentation Feature Level 2:



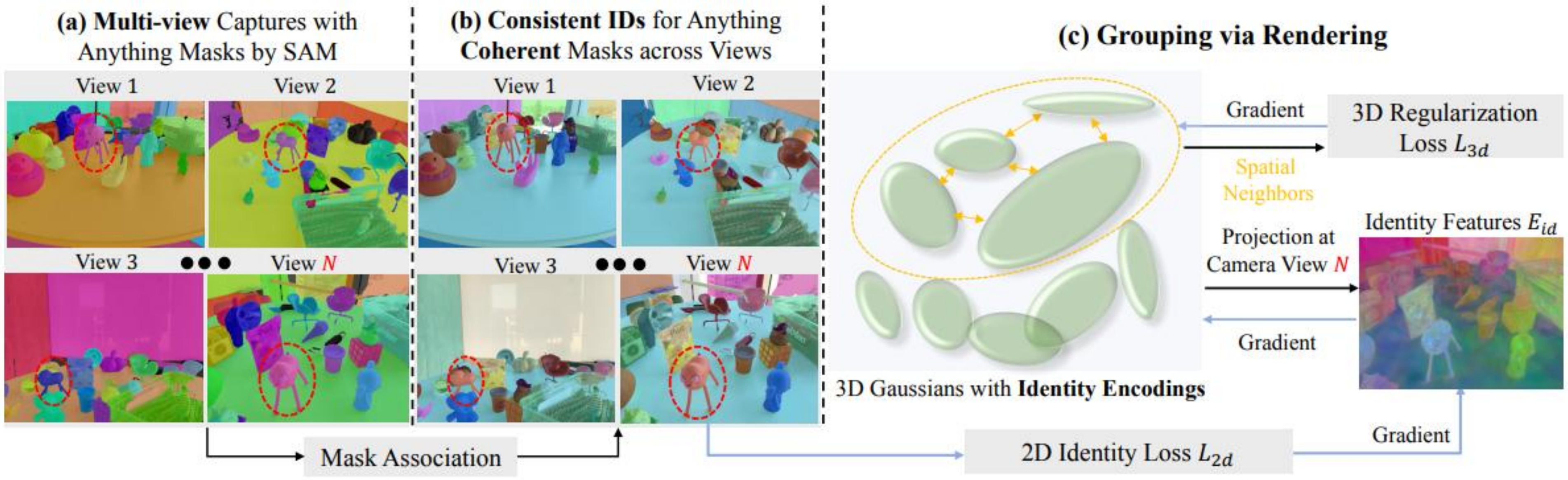
# APPLICATIONS AND POSSIBILITIES
# LANGSPLAT - FEATURE LEVEL 3

Input:


Segmentation Feature Level 3:

# APPLICATIONS AND POSSIBILITIES
# LANGSPLAT – EXAMPLE VIDEO

# APPLICATIONS AND POSSIBILITIES
# ANOTHER EXAMPLE: GAUSSIAN GROUPING




$$E_{\text{id}} = \sum_{i \in \mathcal{N}} e_i \alpha_i' \prod_{j=1}^{i-1} (1 - \alpha_j')$$

$$\mathcal{L}_{\text{render}} = \mathcal{L}_{\text{rec}} + \mathcal{L}_{\text{id}} = \mathcal{L}_{\text{rec}} + \lambda_{2\text{d}} \mathcal{L}_{2\text{d}} + \lambda_{3\text{d}} \mathcal{L}_{3\text{d}}$$

# APPLICATIONS AND POSSIBILITIES GAUSSIAN GROUPING

**Algorithm 1** *Gaussian Grouping*

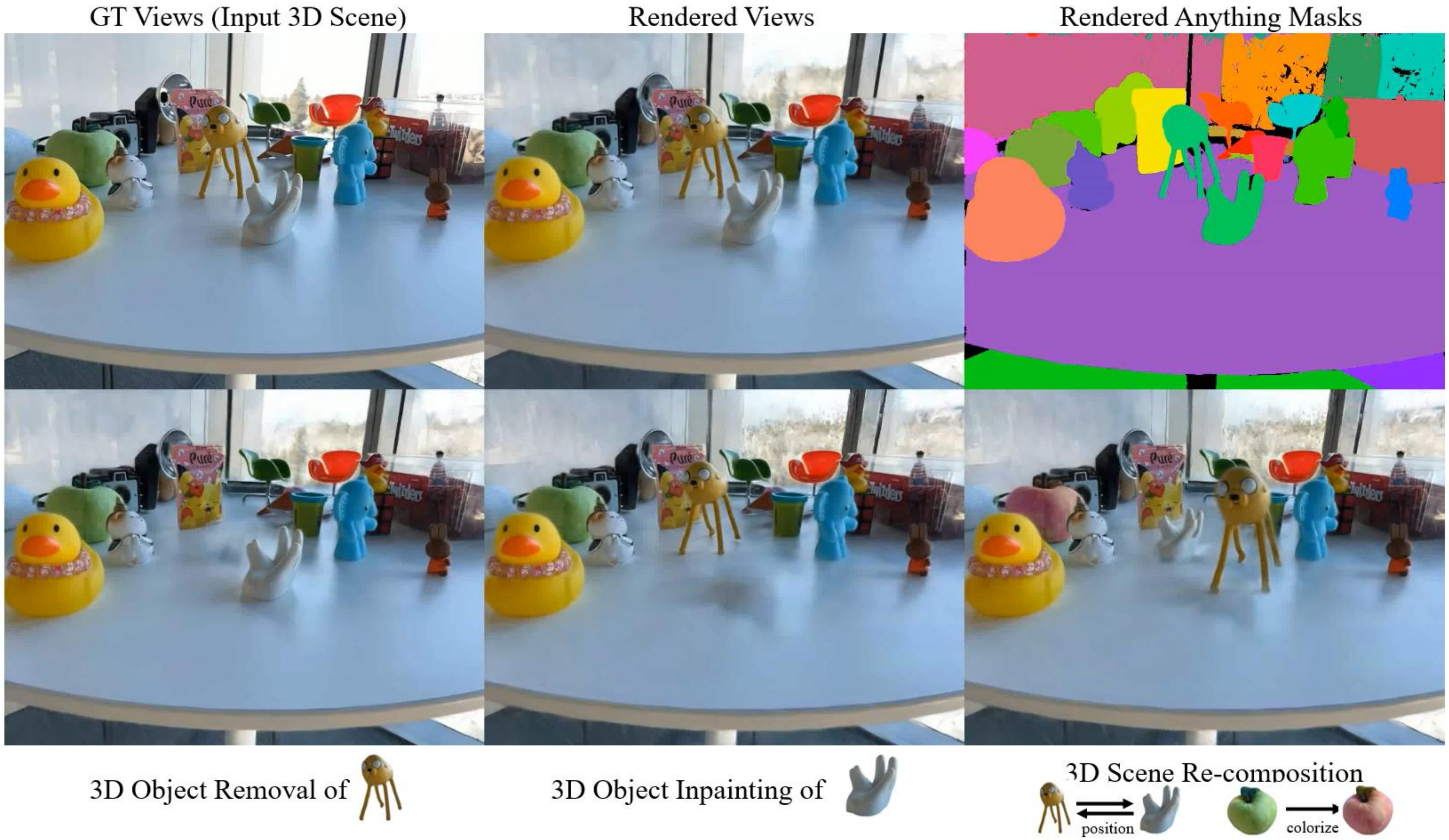
$p \leftarrow$ SfM Points ▷ 3D Positions
$m = (m_1, m_2, \ldots, m_K) \leftarrow$ SAM ▷ **SAM's Masks** at Various $K$ Views
$(\hat{M}_1, \hat{M}_2, \ldots, \hat{M}_K) \leftarrow$ Zero-shot Tracking(m) ▷ Multi-view **Associated Masks**
$s, \alpha, c, \mathbf{e} \leftarrow$ InitAttributes() ▷ Covariances, Opacities, Colors, **Identity Encodings**
$i \leftarrow 0$ ▷ Iteration Count
**while** not converged **do**
  $V, \hat{I}, \hat{M} \leftarrow$ SampleTrainingView() ▷ Camera View $V$, Image and **Mask**
  $I, \mathbf{E}_{\mathrm{id}} \leftarrow$ Rasterize($p, s, a, c, \mathbf{e}, V$) ▷ Rendered Image and **Identity Encoding**
  $\mathcal{L}_{\mathrm{image}} \leftarrow \mathcal{L}(I, \hat{I})$ ▷ Original Image Rendering Loss
  $\mathcal{L}_{\mathrm{id}} \leftarrow \lambda_{2d}\mathcal{L}_{2d}(\mathbf{E}_{\mathrm{id}}, \hat{M}) + \lambda_{3d}\mathcal{L}_{3d}(\mathbf{e})$ ▷ **Identity Grouping Loss**, Eq. 3
  $\mathcal{L} \leftarrow \mathcal{L}_{\mathrm{image}} + \mathcal{L}_{\mathrm{id}}$ ▷ Total Loss
  $p, s, a, c, \mathbf{e} \leftarrow$ Adam($\nabla\mathcal{L}$) ▷ Backprop & Step

**Algorithm 1** Optimization and Densification
$w$, $h$: width and height of the training images

$M \leftarrow$ SfM Points ▷ Positions
$S, C, A \leftarrow$ InitAttributes() ▷ Covariances, Colors, Opacities
$i \leftarrow 0$ ▷ Iteration Count
**while** not converged **do**
  $V, \hat{I} \leftarrow$ SampleTrainingView() ▷ Camera $V$ and Image
  $I \leftarrow$ Rasterize($M, S, C, A, V$) ▷ Alg. 2
  $L \leftarrow Loss(I, \hat{I})$ ▷ Loss
  $M, S, C, A \leftarrow$ Adam($\nabla L$) ▷ Backprop & Step

$$E_{\mathrm{id}} = \sum_{i \in \mathcal{N}} e_i \alpha'_i \prod_{j=1}^{i-1} (1 - \alpha'_j)$$

$$\mathcal{L}_{\mathrm{render}} = \mathcal{L}_{\mathrm{rec}} + \mathcal{L}_{\mathrm{id}} = \mathcal{L}_{\mathrm{rec}} + \lambda_{2\mathrm{d}}\mathcal{L}_{2\mathrm{d}} + \lambda_{3\mathrm{d}}\mathcal{L}_{3\mathrm{d}}$$


GT Views (Input 3D Scene)
Rendered Views
Rendered Anything Masks
3D Object Removal of
3D Object Inpainting of
3D Scene Re-composition
position
colorize

END!

THANKS FOR YOUR ATTENTION! :)