4D Gaussian Splatting for Real-Time Dynamic Scene Rendering

Marcelo de Sousa - Reviewer
Marcelo de Sousa - Archaeologist
Fabricio - Hacker
Esteban - PhD Student

impa

4D Gaussian Splatting

Reviewer - Marcelo de Sousa

1. Introduction

1. Introduction

Motivation:

- Dynamic scene rendering is critical for applications like VR, simulation, and film production.
- Existing methods (e.g., 3DGS) are limited to static or quasi-static scenes, struggling with real-time performance and storage efficiency.

Key Challenges:

- Modeling **complex motion** and deformation with **limited input.**
- Maintaining **real-time rendering speed** without sacrificing quality.
- Balancing **training time**, storage, and computational efficiency.

1. Introduction

Proposed Solution:

- **4D Gaussian Splatting (4D-GS)**:
  - Combines **3D Gaussian points** with a **temporal deformation field** to represent spatial-temporal dynamics.
  - Achieves real-time rendering with efficient training and compact storage.

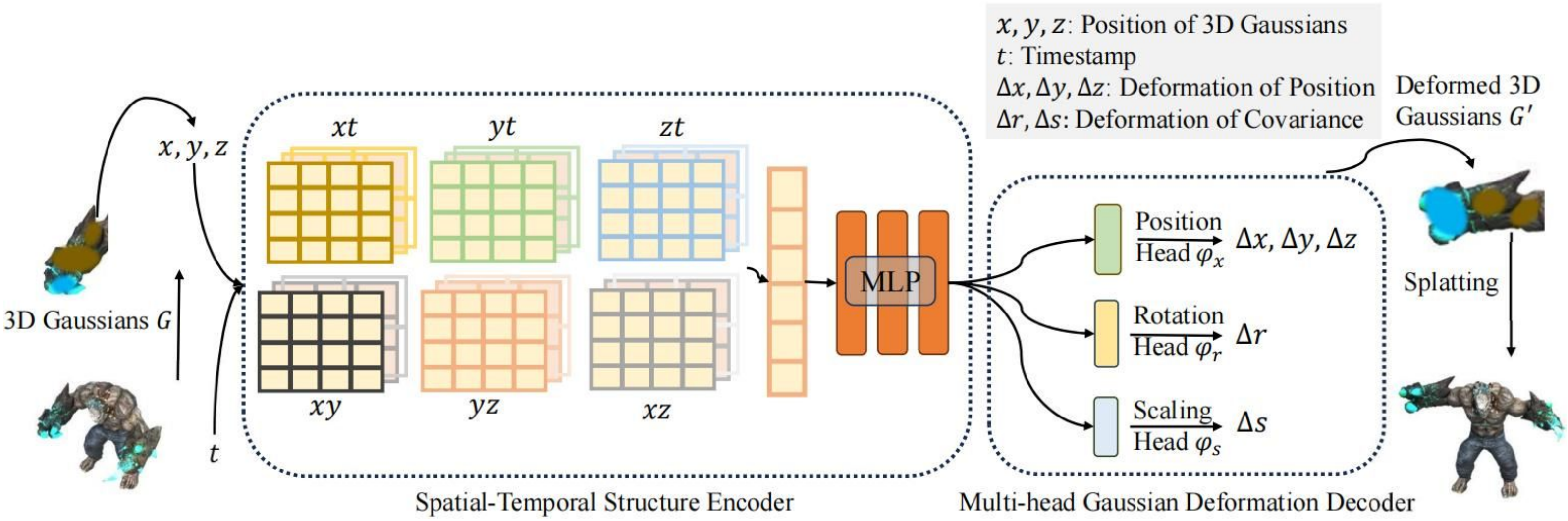
How:

- Temporal-Spatial Structure **Encoder.**
- Multi-head Gaussian deformation **Decoder**.

Contributions:

- Unified framework for modeling **spatial-temporal dynamics**.
- High-quality rendering with **minimal latency** and **low memory consumption O(N+F)**.

2. Method


x, y, z: Position of 3D Gaussians
t: Timestamp
$\Delta x, \Delta y, \Delta z$: Deformation of Position
$\Delta r, \Delta s$: Deformation of Covariance
Deformed 3D Gaussians G'
x, y, z
xt
yt
zt
xy
yz
xz
MLP
3D Gaussians G
t
Position Head φ_x
$\Delta x, \Delta y, \Delta z$
Rotation Head φ_r
Δr
Scaling Head φ_s
Δs
Splatting
Spatial-Temporal Structure Encoder
Multi-head Gaussian Deformation Decoder

2. Method

Overall Framework:

- Combines 3D Gaussians ***G*** with deformation fields ***F*** to model motion and deformation over time.
- Final deformed Gaussians:

$$G' = G + \Delta G$$

Spatial-Temporal Encoder (HexPlane):

- Multi-resolution voxel planes:
  - Encodes features ***R_l(i, j)*** in spatial ***(x, y, z)*** and temporal ***(t)*** dimensions.
  - Interpolates Gaussian features for efficient storage and computation:

$$f_h = \bigcup_l \prod \mathrm{interp}(R_l(i,j)),$$
$$(i,j) \in \{(x,y),(x,z),(y,z),(x,t),(y,t),(z,t)\}.$$

2. Method

Gaussian Deformation Decoder, D={ϕ_x,ϕ_r,ϕ_s}:

- Multi-head MLPs predict deformations:

$$\Delta X = \phi_x(f_h), \quad \Delta r = \phi_r(f_h), \quad \Delta s = \phi_s(f_h)$$

- Updated Gaussian attributes:

$$X' = X + \Delta X, \quad r' = r + \Delta r, \quad s' = s + \Delta s$$

Finally, **G′={X′,s′,r′,σ,C}**

Rendering Process:

- Deformed Gaussians **G′** are projected via differentiable splatting:

$$\hat{I} = S(M, G')$$

S: Splatting operator, **M[R,T]**: View matrix.

2. Method

Optimization: 3D Gaussian Initialization

- **Why Initialization Matters**:
  - Proper initialization ensures efficient training and faster convergence.
  - Avoids excessive deformation during early training phases.
- **Methodology**:
  - **Structure from Motion (SfM)**:
    - Initializes 3D Gaussians using SfM points, leveraging geometric priors.
  - **Warm-Up Phase**:
    - Optimize 3D Gaussians for **3000 iterations** to stabilize the initial configuration.
    - Render images with 3D Gaussians (**$\hat{I}=S(M,G)$**) before transitioning to 4D Gaussians.

2. Method

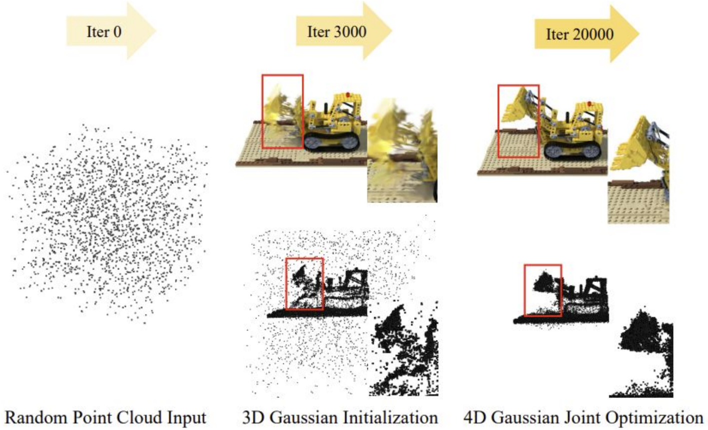


Figure 4. Illustration of the optimization process. With static 3D Gaussian initialization, our model can learn high-quality 3D Gaussians of the motion part.

2. Method

Loss Function

- **Reconstruction Loss**:

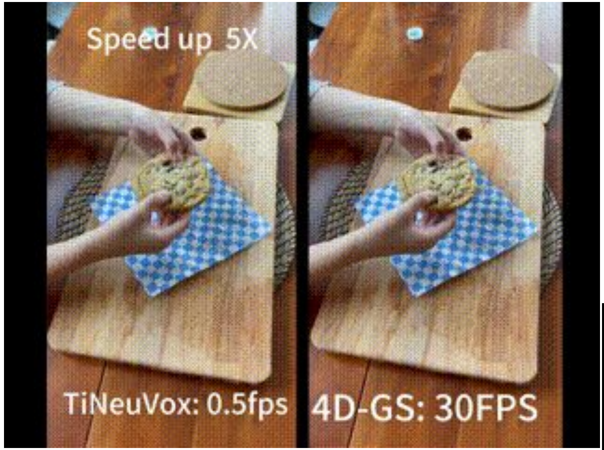
$$L_{\text{color}} = |\hat{I} - I|$$

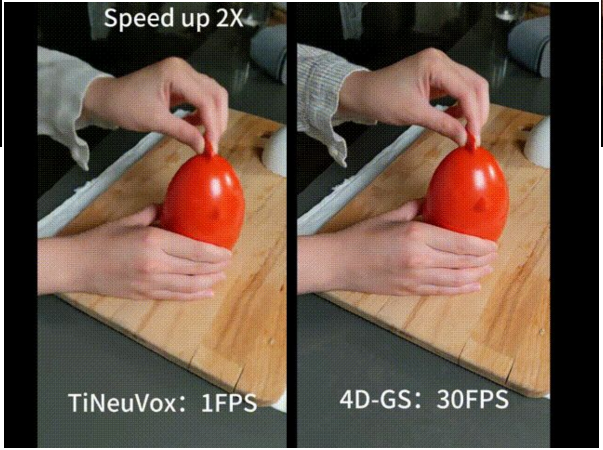
- **Total Variation Loss (L_{tv})**:
  - Regularizes and smooths the voxel grids.
- **Combined Loss**:

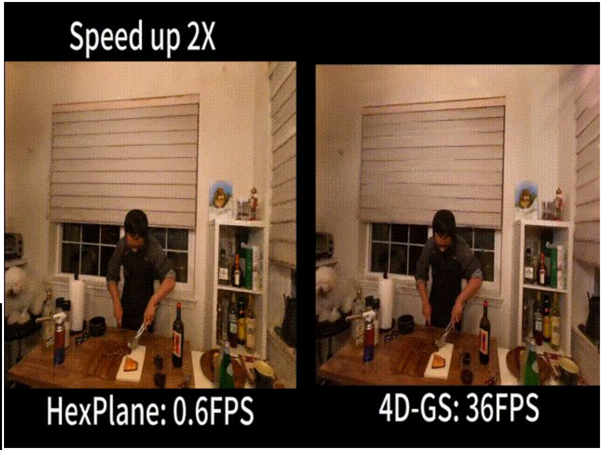
$$L = |\hat{I} - I| + L_{\text{tv}}$$

2. Method

3. Experimentation and Results







3. Experimentation and Results

Setup

- **Hardware**: PyTorch implementation on RTX 3090 GPU.
- **Datasets**:
  - **Synthetic (D-NeRF)**: Monocular scenes with 50–200 frames.
  - **Real-World**: HyperNeRF (simple monocular setups) and Neu3D (multi-camera, complex motion).

3. Experimentation and Results

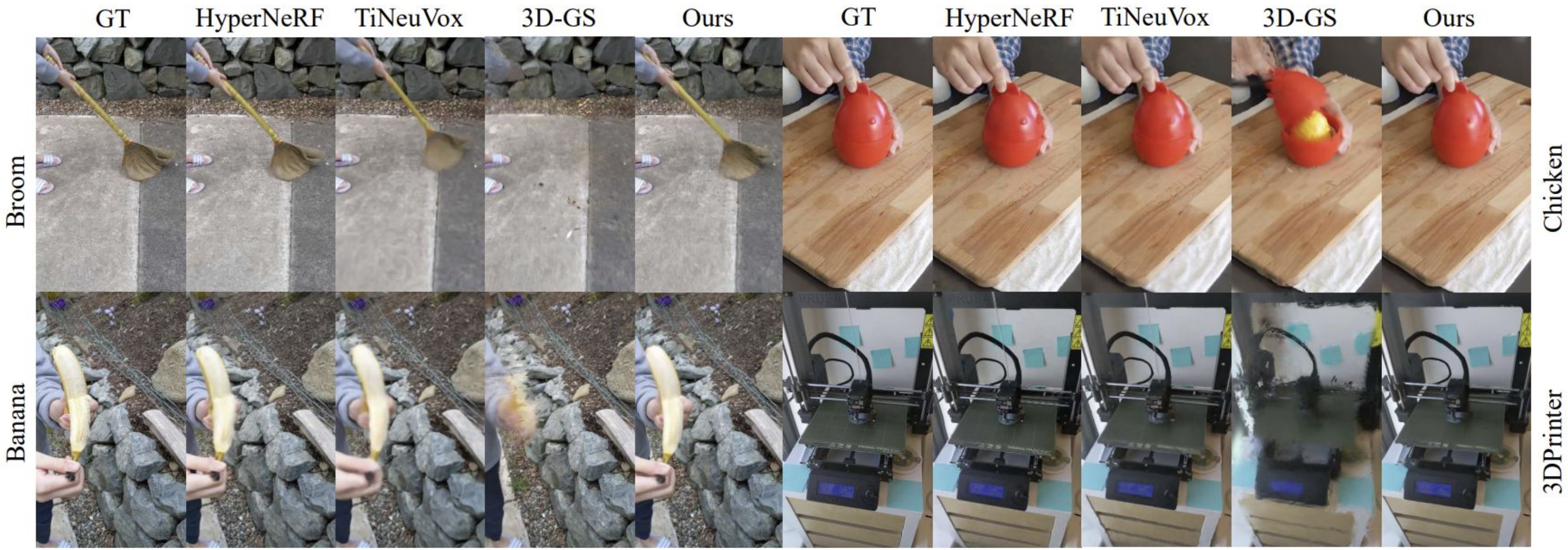


Figure 6. Visualization of the HyperNeRF [39] dataset compared with other methods [9, 19, 22, 39]. 'GT' stands for ground truth images.

3. Experimentation and Results

Table 2. Quantitative results on HyperNeRF [39] vrig dataset with the rendering resolution of 960×540.

Model	PSNR (dB)↑	MS-SSIM↑	Times↓	FPS↑	Storage (MB)↓
Nerfies [38]	22.2	0.803	∼ hours	< 1	-
HyperNeRF [39]	22.4	0.814	32 hours	< 1	-
TiNeuVox-B [9]	24.3	0.836	30 mins	1	48
3D-GS [22]	19.7	0.680	40 mins	55	52
FFDNeRF [19]	24.2	0.842	-	0.05	440
V4D [13]	24.8	0.832	5.5 hours	0.29	377
Ours	25.2	0.845	30 mins	34	61

Table 3. Quantitative results on the Neu3D [25] dataset with the rendering resolution of 1352×1014.

Model	PSNR (dB)↑	D-SSIM↓	LPIPS↓	Time ↓	FPS↑	Storage (MB)↓
NeRFPlayer [49]	30.69	0.034	0.111	6 hours	0.045	-
HyperReel [2]	31.10	0.036	0.096	9 hours	2.0	360
HexPlane-all* [5]	31.70	0.014	0.075	12 hours	0.2	250
KPlanes [12]	31.63	-	-	1.8 hours	0.3	309
Im4D [30]	32.58	-	0.208	28 mins	∼5	93
MSTH [53]	32.37	0.015	0.056	20 mins	2 (15‡)	135
Ours	31.15	0.016	0.049	40 mins	30	90

*: The metrics of the models are tested without "coffee martini" and resolution is set to 1024×768.
‡: The FPS is tested with fixed-view rendering.

3. Experimentation and Results (800x800)

Table 4. Ablation studies on synthetic datasets using our proposed methods.

Model	PSNR(dB)↑	SSIM↑	LPIPS↓	Time↓	FPS↑	Storage (MB)↓
Ours w/o HexPlane $R_l(i,j)$	27.05	0.95	0.05	4 mins	140	12
Ours w/o initialization	31.91	0.97	0.03	7.5 mins	79	18
Ours w/o ϕ_x	26.67	0.95	0.07	8 mins	82	17
Ours w/o ϕ_r	33.08	0.98	0.03	8 mins	83	17
Ours w/o ϕ_s	33.02	0.98	0.03	8 mins	82	17
Ours	34.05	0.98	0.02	8 mins	82	18

3. Experimentation and Results

Limitations

- Large-scale urban reconstructions require further optimization.
- Struggles with large motion and imprecise camera poses.

4. Why 4D Gaussian Splatting Stands Out

- **Unified Representation**: Combines spatial and temporal features seamlessly.
- **Real-Time Performance**: Fast rendering with differentiable splatting, achieving up to 90 FPS.
- **Efficient Encoding**: HexPlane reduces memory usage while capturing fine spatial-temporal details.
- **Dynamic Adaptability**: Models complex deformations like stretching, twisting, and scaling.
- **Monocular Flexibility**: Handles monocular setups effectively, unlike multi-camera-dependent methods.
- 

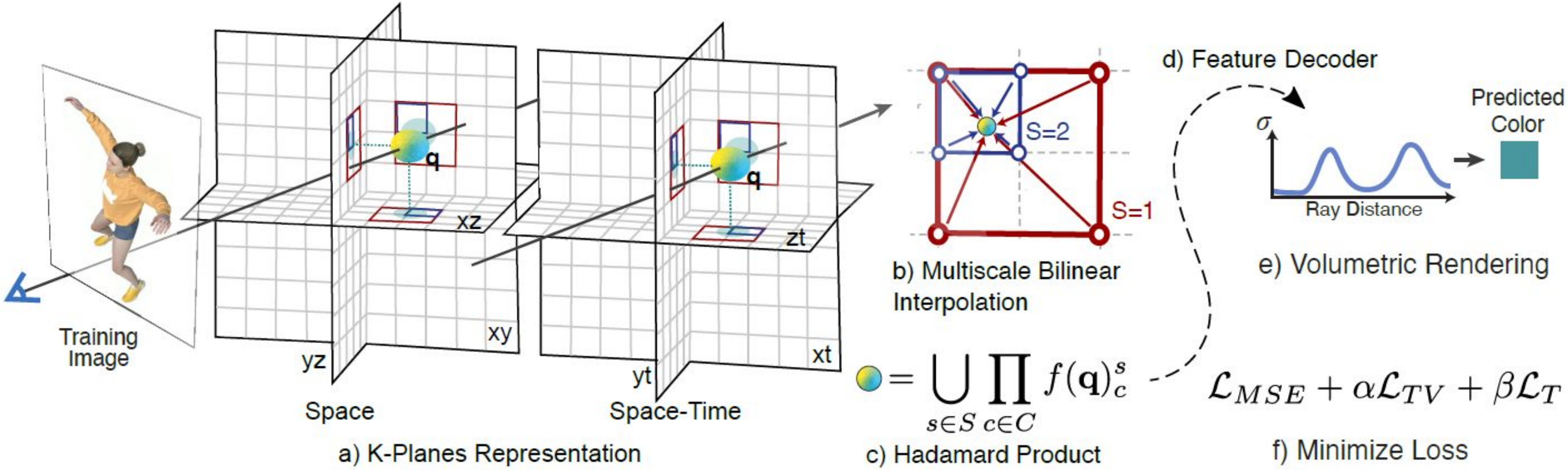
4D Gaussian Splatting

Archaeologist - Marcelo de Sousa

Exploring Key References in Dynamic Scene Representation and Rendering

Year	References
2015	[7] Streamable free-viewpoint video, exploring dynamic scene representation.
2019	[63] Differentiable surface splatting for point-based geometry in dynamic scenes.
2020	[4] Layered mesh representations for light field video in dynamic scenarios.
2021	[35] NeRF techniques for dynamic view synthesis and scene representation. [42] D-NeRF: Neural radiance fields for dynamic scenes.
2022	[22] Real-time 3D Gaussian splatting for radiance field rendering in dynamic scenes.
2023	[5] Scalable HexPlane representation optimized for dynamic environments.

K-Planes: Method

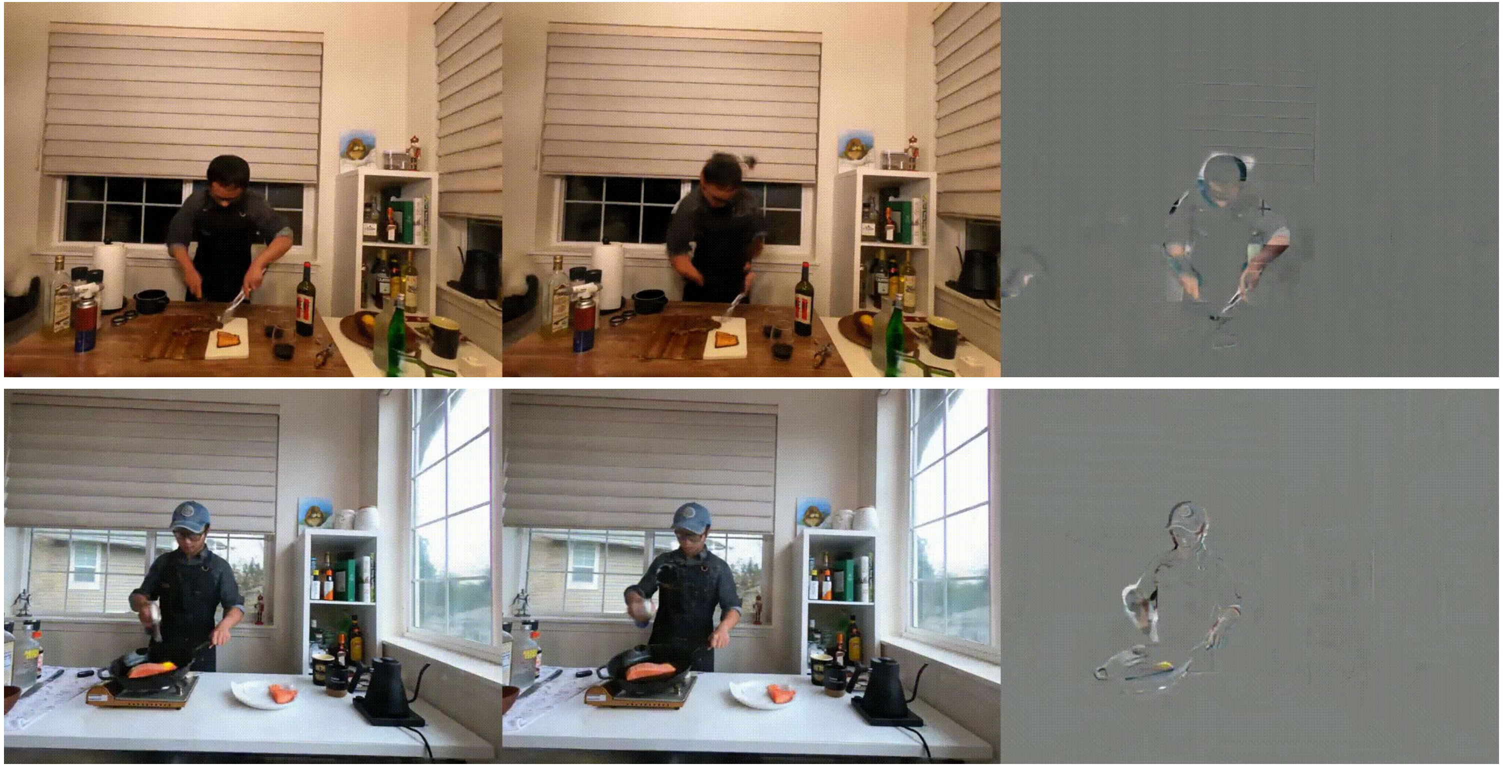


K-Planes: Comparison

Summary of Differences

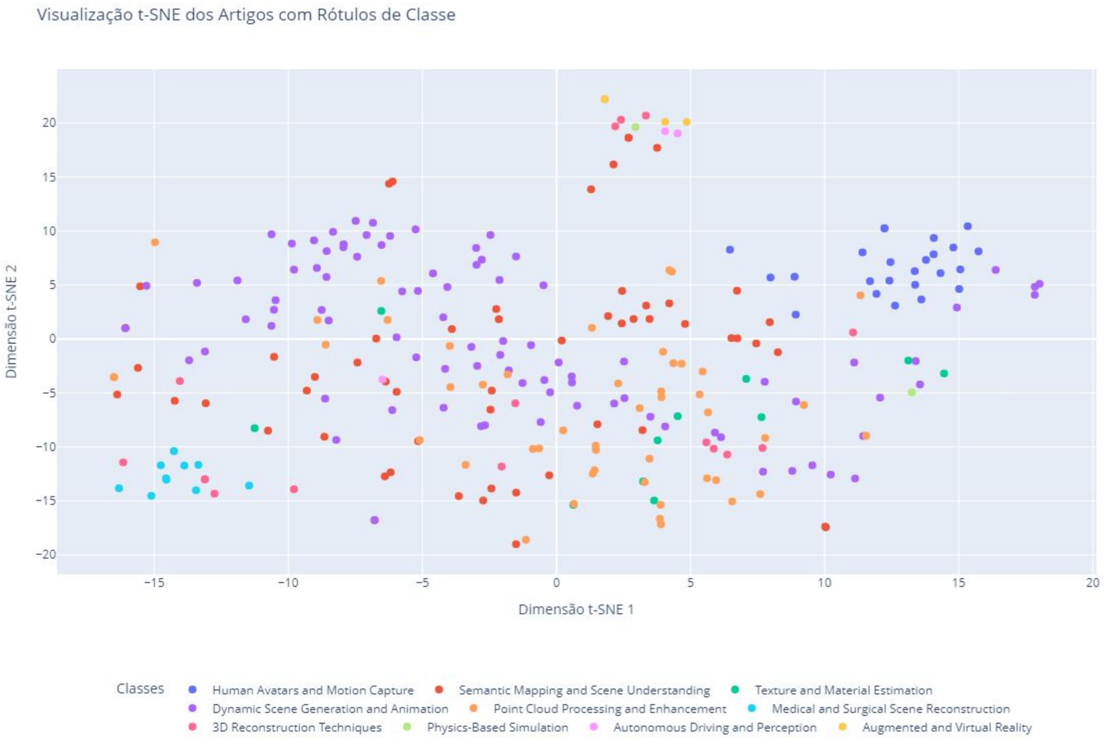
Aspect	K-Plane	4D-GS
Representation	Factorized 2D planes	Deformable 3D Gaussians
Rendering Strategy	Ray Marching	Differentiable Splatting
Efficiency	Moderately fast	Extremely fast (real-time)
Dynamic Scenes	Suitable but limited to space-time planes	Excellent for complex deformations
Complexity	High (factorization, interpolation, decoding)	Moderate (Gaussians and direct deformation)

K-Planes: Space-time Decomposition



K-Planes: Explicit Radiance Fields in Space, Time, and Appearance (CVPR 2023)

Citing papers



Applications: Text-to-4D

An astronaut riding a horse, best quality, 4K, HD

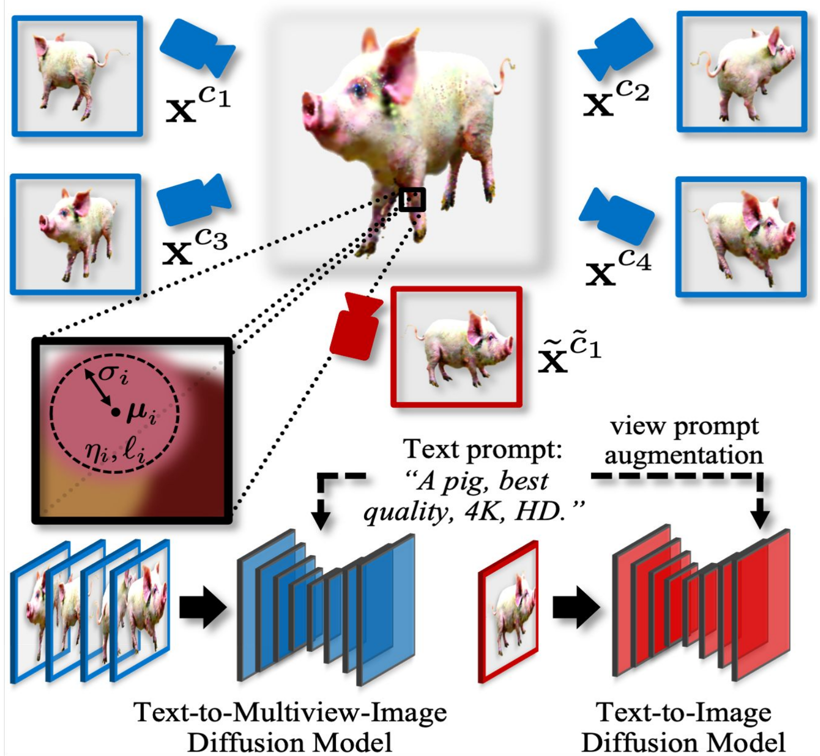
An ancient roman statue dancing, full body, portrait, game, unreal, 4K, HD

A storm trooper walking forward and vacuuming, best quality, 4K, HD

Applications: Text-to-4D

Stage 1: Static 3D Synthesis

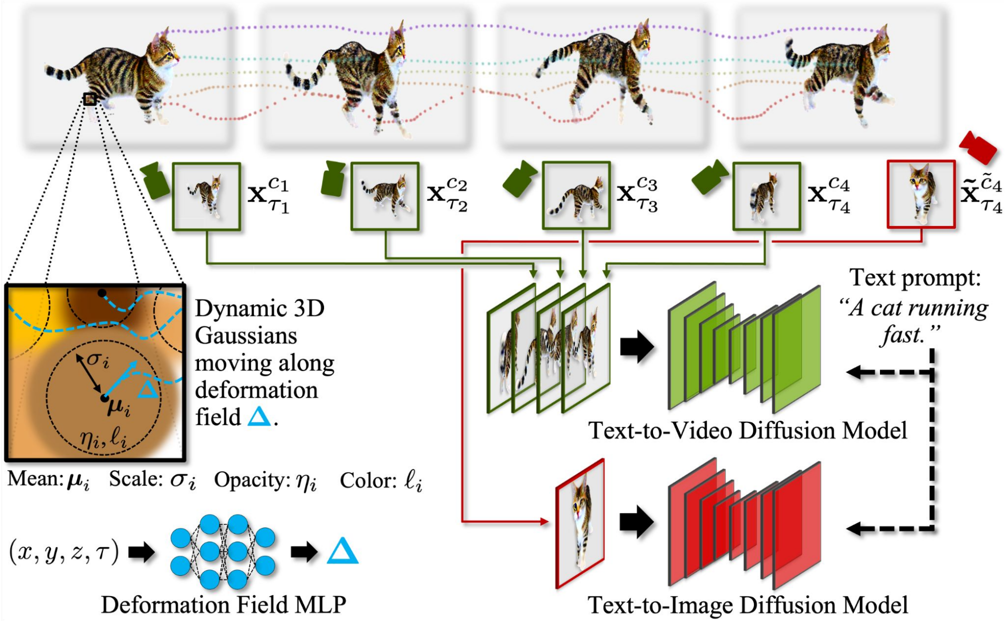
- Optimize 3D images for static scenes using MVDream.
- Enhance with Stable Diffusion for text-to-image quality.



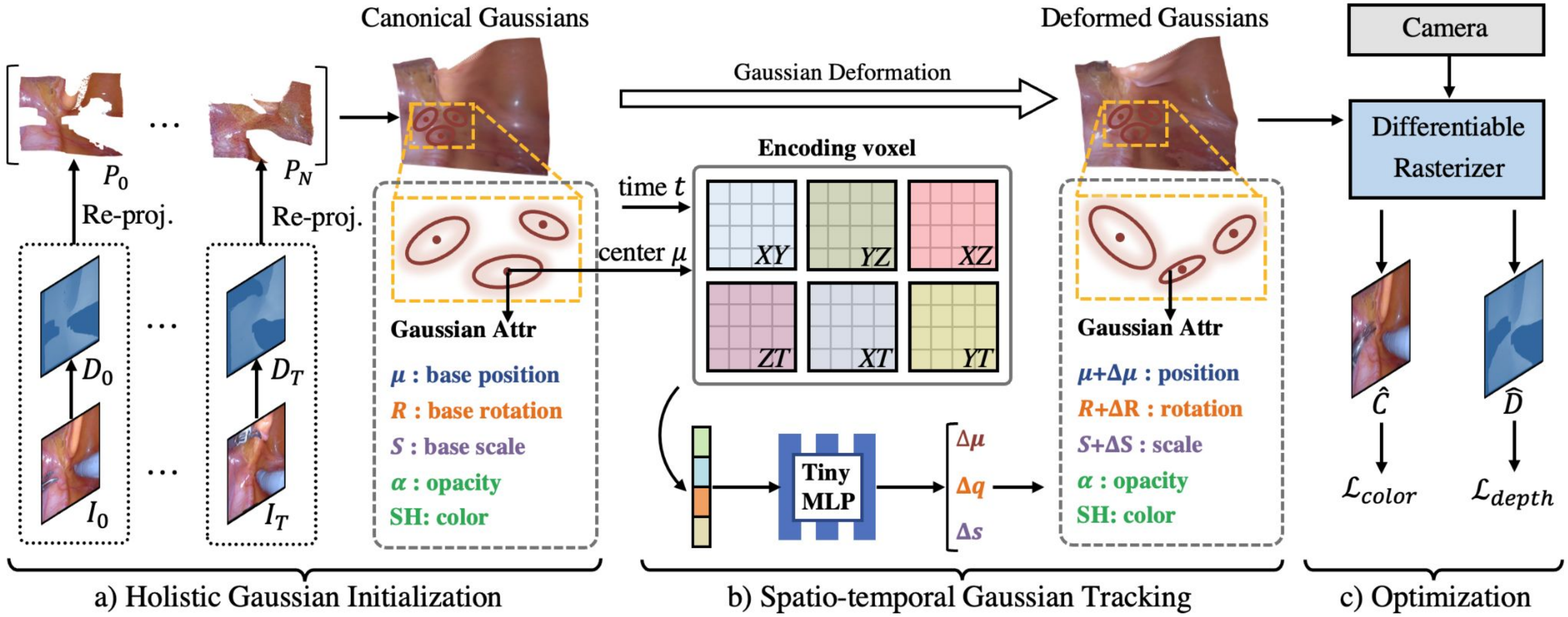
Applications: Text-to-4D

Stage 2: Dynamic 4D Synthesis

- Combine text-to-video and text-to-image models for 4D dynamics.
- Optimize deformation fields while ensuring high-quality frames.

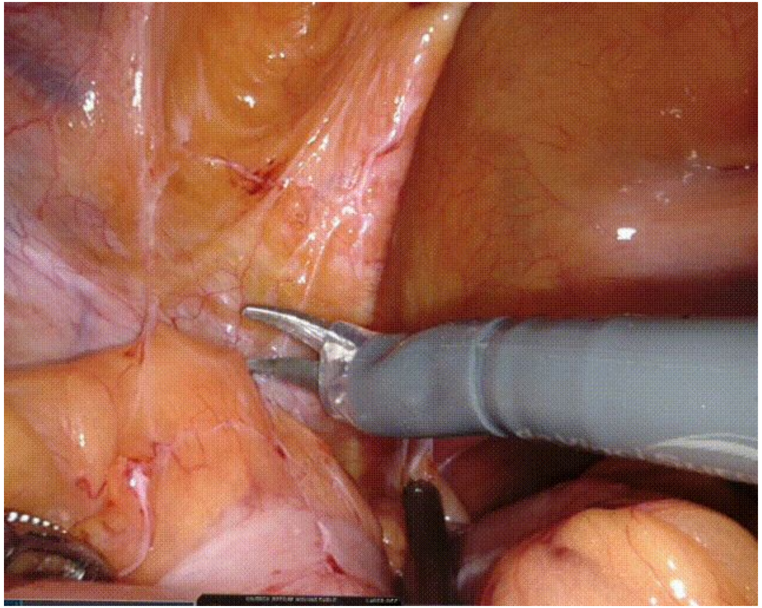


Applications: EndoGaussian

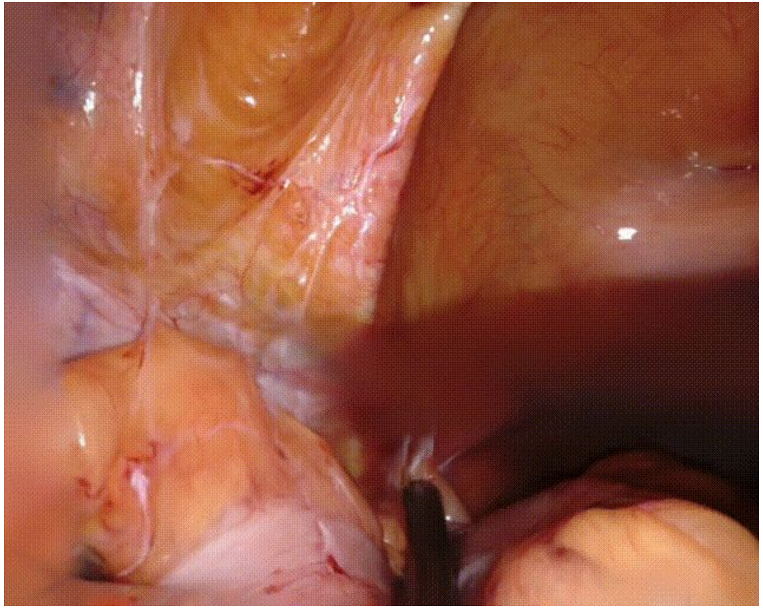


Applications: EndoGaussian

Cutting - GT

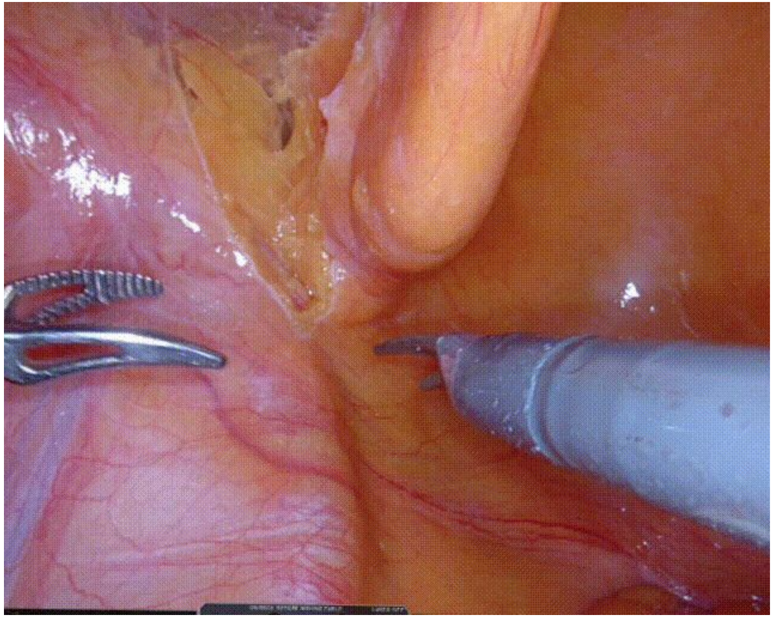


Cutting - Paper

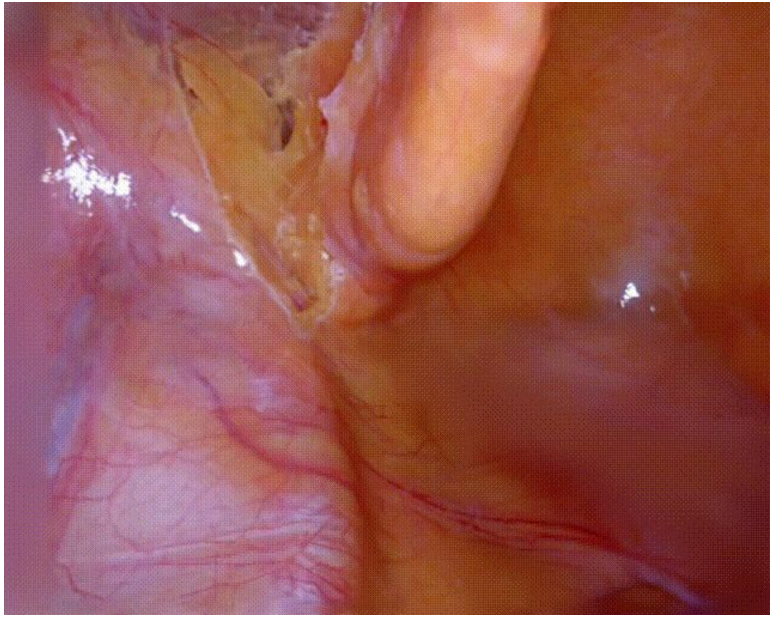


Applications: EndoGaussian

Pulling - GT



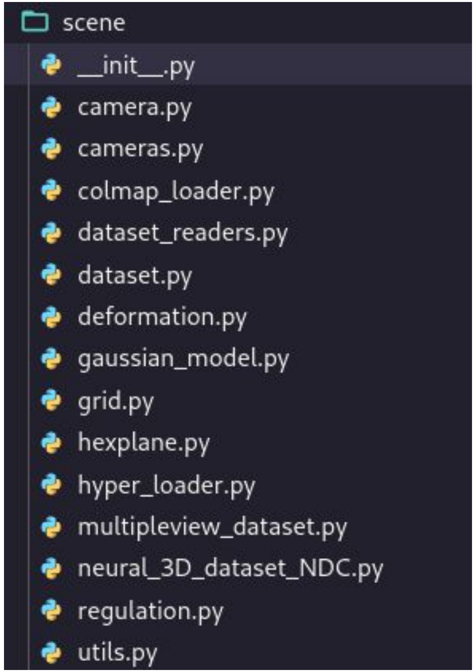
Pulling - Paper



4D Gaussian Splatting

Hacker - Fabricio

The Source Code



Source code file hierarchy

The Source Code

```python
def create_net(self):
    mlp_out_dim = 0
    if self.grid_pe !=0:
        grid_out_dim = self.grid.feat_dim+(self.grid.feat_dim)*2
    else:
        grid_out_dim = self.grid.feat_dim
    if self.no_grid:
        self.feature_out = [nn.Linear(4,self.W)]
    else:
        self.feature_out = [nn.Linear(mlp_out_dim + grid_out_dim ,self.W)]
    for i in range(self.D-1):
        self.feature_out.append(nn.ReLU())
        self.feature_out.append(nn.Linear(self.W,self.W))
    self.feature_out = nn.Sequential(*self.feature_out) # Spatial-temporal Structure Encoder

    # Heads of Multi-head Gaussian Deformation Decoder
    self.pos_deform = nn.Sequential(nn.ReLU(),nn.Linear(self.W,self.W),nn.ReLU(),nn.Linear(self.W, 3))
    self.scales_deform = nn.Sequential(nn.ReLU(),nn.Linear(self.W,self.W),nn.ReLU(),nn.Linear(self.W, 3))
    self.rotations_deform = nn.Sequential(nn.ReLU(),nn.Linear(self.W,self.W),nn.ReLU(),nn.Linear(self.W, 4))
    self.opacity_deform = nn.Sequential(nn.ReLU(),nn.Linear(self.W,self.W),nn.ReLU(),nn.Linear(self.W, 1))
    self.shs_deform = nn.Sequential(nn.ReLU(),nn.Linear(self.W,self.W),nn.ReLU(),nn.Linear(self.W, 16*3))
```

The Source Code

```
dx = self.pos_deform(hidden)
ds = self.scales_deform(hidden)
do = self.opacity_deform(hidden)
dr = self.rotations_deform(hidden)
dshs = self.shs_deform(hidden).reshape([shs_emb.shape[0],16,3])
```

$$\Delta X = \phi_x(f_h), \quad \Delta r = \phi_r(f_h), \quad \Delta s = \phi_s(f_h)$$

The Source Code

```
pts = rays_pts_emb[:,:3]*mask + dx
scales = scales_emb[:,:3]*mask + ds
opacity = opacity_emb[:,:1]*mask + do
rotations = rotations_emb[:,:4] + dr
shs = shs_emb*mask.unsqueeze(-1) + dshs
```

$$X' = X + \Delta X, \quad r' = r + \Delta r, \quad s' = s + \Delta s$$

Reproducibility

- Easy reproduction on synthetic dataset from Dnerf, previously pre-processed by the authors;
- Other datasets can be used, such as Hypernerf’s or Dynerf’s, however extra steps are required to pre-process it, with some dependencies not stated in the README;

Experiments

- The experiments were based on the Dnerf's Trex dataset;
- Focused hyperparameters: grid learning rate (encoder), deformation learning rate (decoder), iterations, pruning interval;

Experiments

Default hyperparameters

Half iterations

Experiments

Default hyperparameters

Double iterations

Experiments

Default hyperparameters

Double grid learning rate

Experiments

Half iterations

Half iterations half grid lr

Experiments

Half iterations

Half iterations half pruning interval

Experiments

Half iterations

Half iterations half deformation lr

Experiments

Half iterations

Half iterations double deformation lr

Experiments

Default hyperparameters

Half iterations 5x deformation lr double grid lr

The Source Code: overall opinions

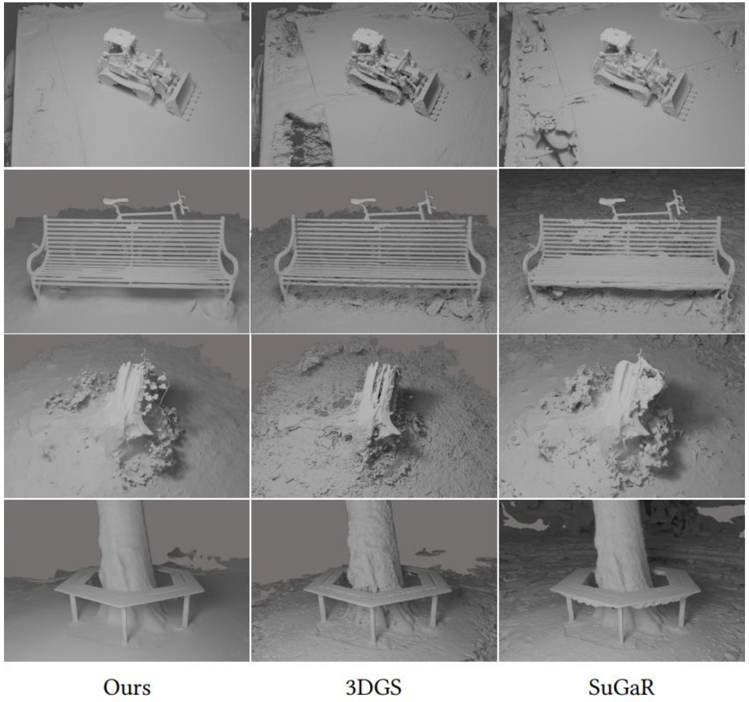
- Too much dead code / long methods / unorganized classes, leading to unnecessarily difficult analysis of source code;
- Hyperparameters declared as .py files, instead of .json or .env;
- Poor repository organization, file contents are not immediate by location and name alone;

4D Gaussian Splatting

PhD Student - Esteban

Identify area of improvement and possible solution

- Area of Improvement:
    - Lack of background points
    - Inability to split between dynamic and static gaussians
- Possible solution:
    - Use 2D Gaussians



Expected improvement over original paper

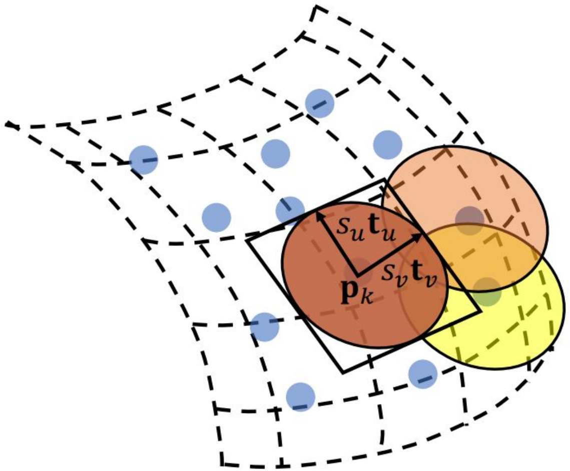
- 2D Gaussians are able to create a mesh with more depth accuracy.
  - This introduces the capability of identifying background points.

- Improve ability to distinguish static and dynamic gaussians.

- Could improve segmentation for directed edition over video.

How we will do it

- Change the tracking of the variance at a each time t to tracking the two tangential vectors of a 2D Gaussian over a surface.
- Now the transformation of the Gaussians over time is represented as a translation and rotation.



How we will do it

- Introduce depth consistency and normal distortion loss functions to make sure the 2D Gaussians give an accurate representation of the scene.

- Create a dynamic mesh that allows to separate the object from the background.
  - This may also allow to have segmentation to edit a certain object in a video for future applications.

Thank you!

Question?