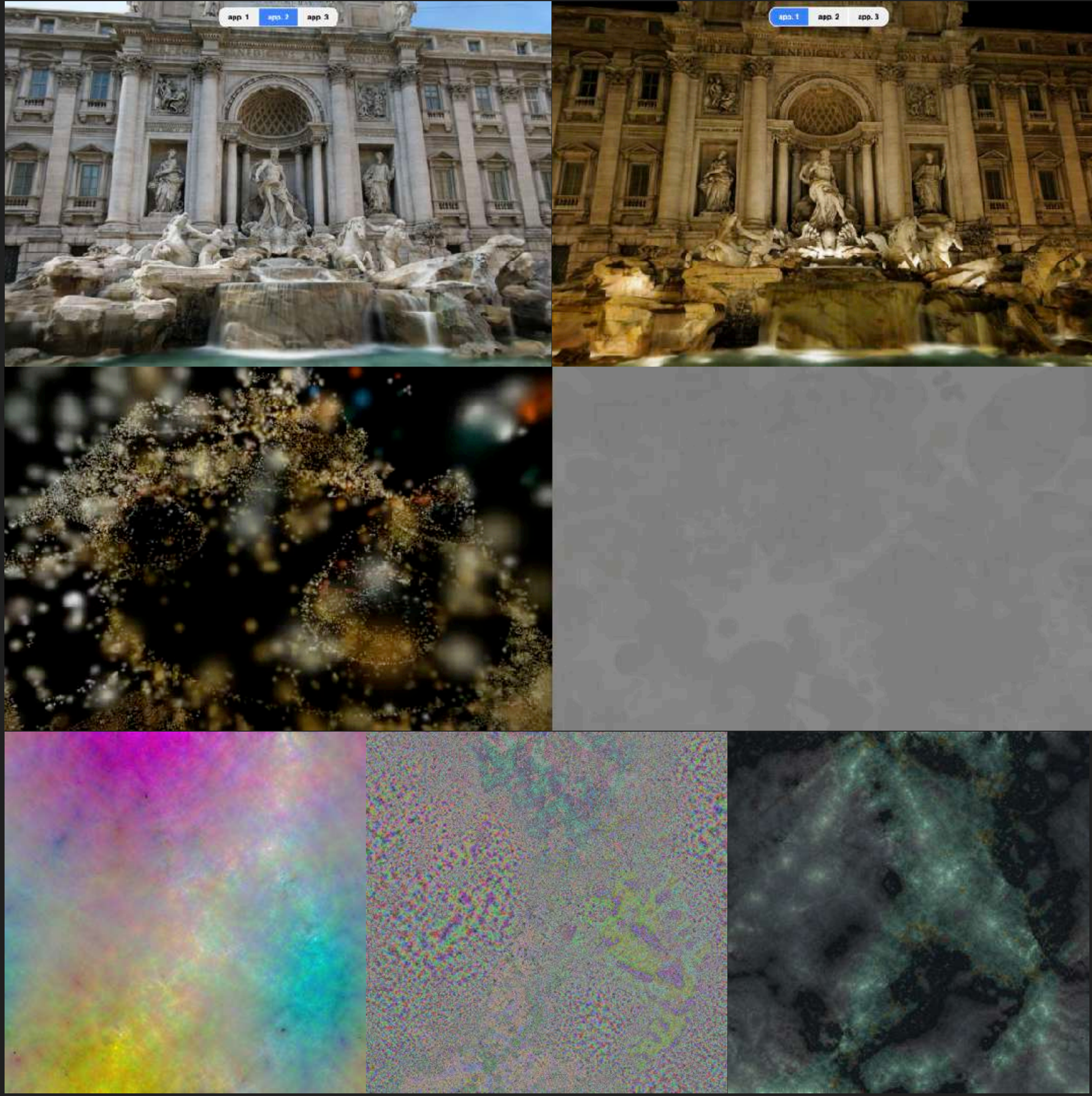
gsplat + MLPs

Jeffrey Hu | November 22, 2024

@jefequien



Outline

1. About Me
2. Appearance + MLP
3. Deblurring + MLPs
4. Compression + MLP
5. Recommendations

About Me

- 2017 - 2019 – MIT CSAIL
- 2020 - 2022 – TuSimple
- 2022 - 2024 – Parallel Systems
- 2024 - now – gsplat + PhD apps

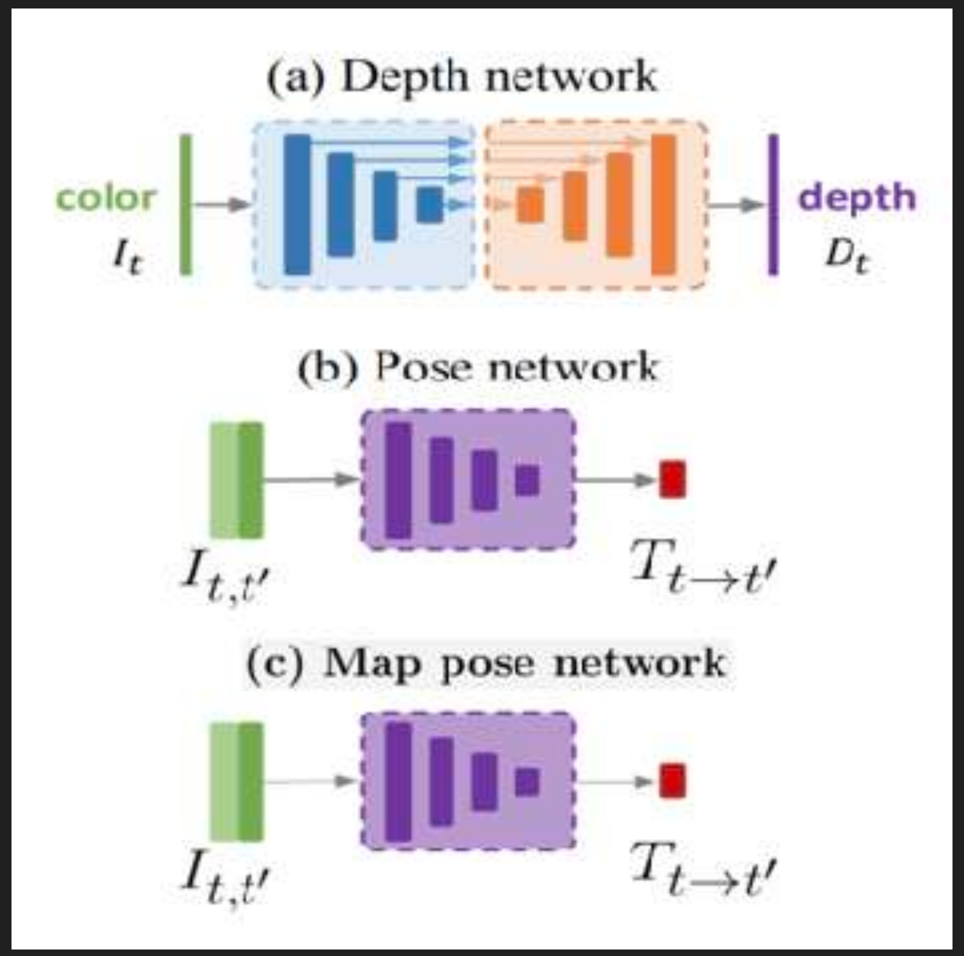
About Me

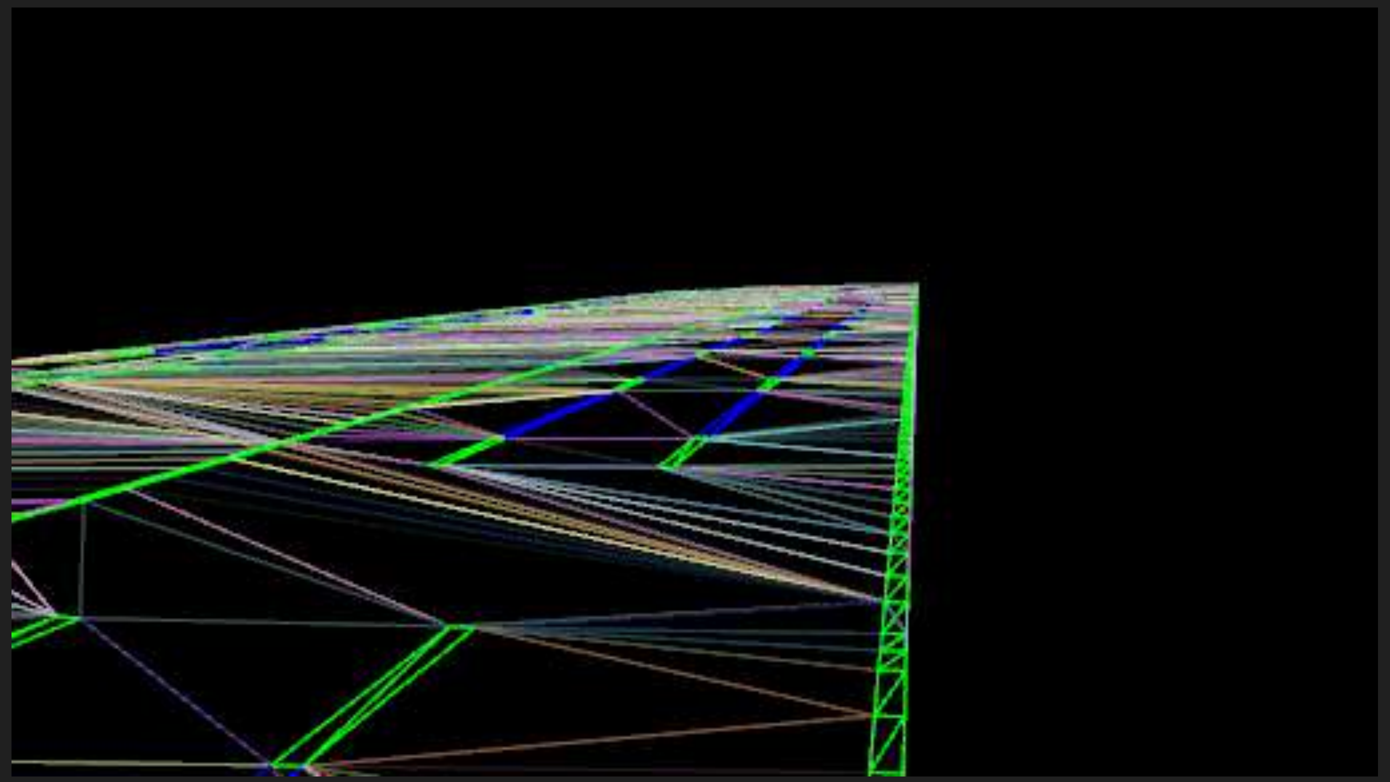
- **2017 - 2019 – MIT CSAIL**
  - Segmentation, Pose Detection
  - YOLOs, MaskRCNN, AlphaPose
- 2020 - 2022 – TuSimple
- 2022 - 2024 – Parallel Systems
- 2024 - now – gsplat + PhD apps

MIT CSAIL

About Me

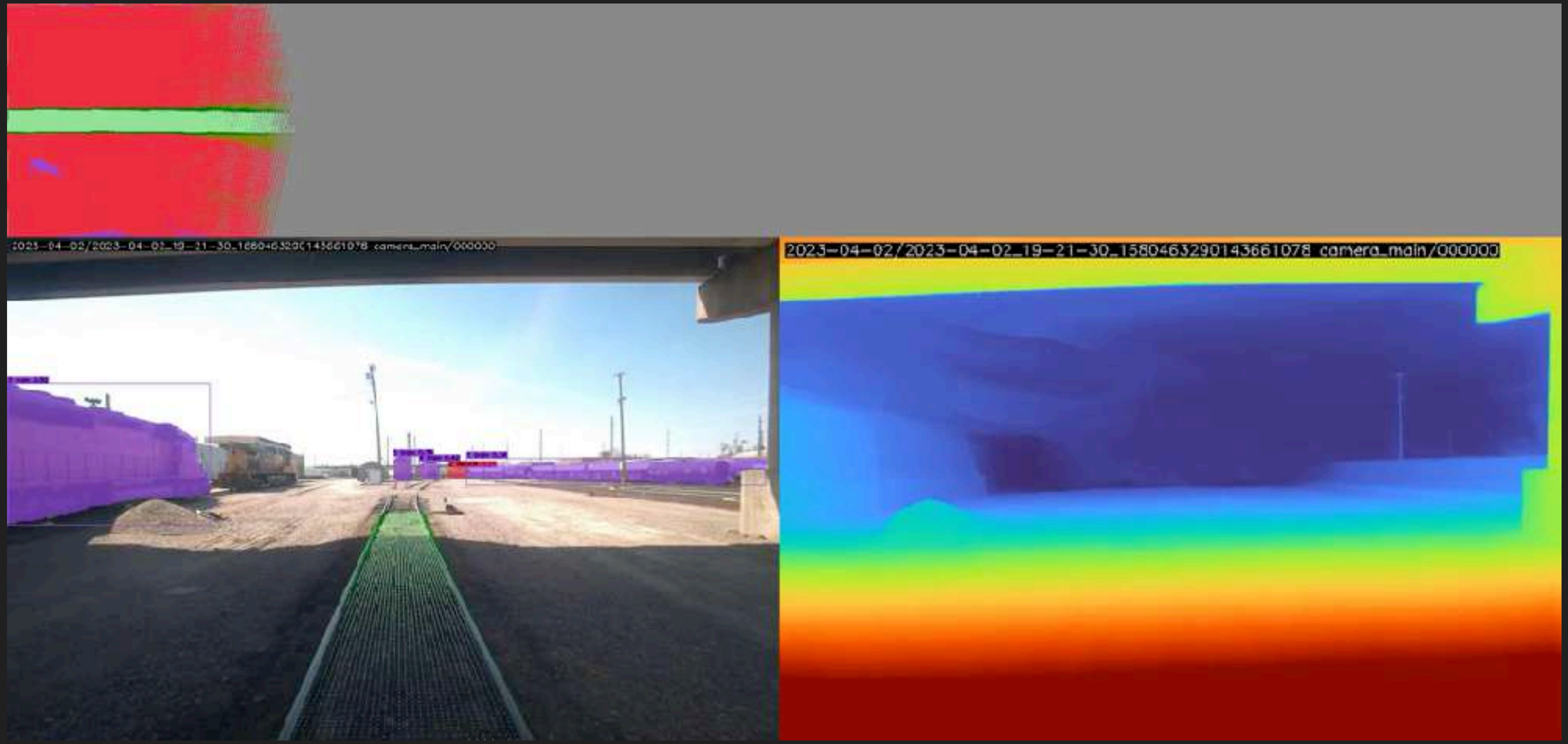
- 2017 - 2019 – MIT CSAIL
- **2020 - 2022 – TuSimple**
    - Localization
    - Self-supervised monocular depth
- 2022 - 2024 – Parallel Systems
- 2024 - now – gsplat + PhD apps





About Me

- 2017 - 2019 – MIT CSAIL
- 2020 - 2022 – TuSimple
- **2022 - 2024 – Parallel Systems**
  - 3D auto-labeling with SAM, GroundingDINO, Midas, Marigold, DPVO, Instant-NGP
  - HydraNet with ResNet backbone and task-specific heads
- 2024 - now – gsplat + PhD apps



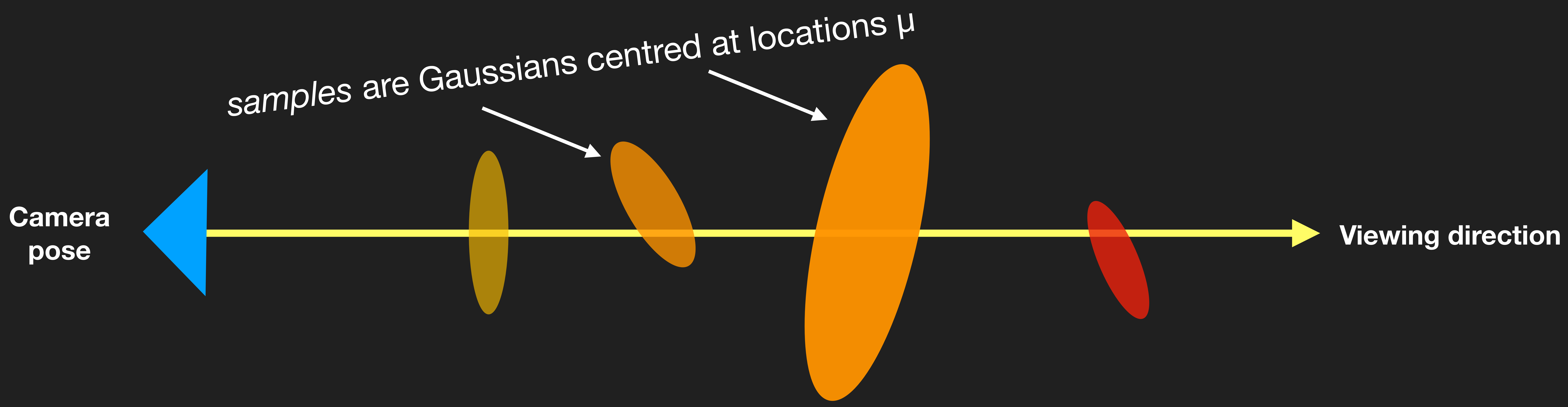
About Me

- 2017 - 2019 – MIT CSAIL
- 2020 - 2022 – TuSimple
- 2022 - 2024 – Parallel Systems
- **2024 - now – gsplat + PhD apps**
    - ~~Video diffusion models~~
    - Gaussian splatting
        - MCMC, bilateral, 2.5DGS, Fisheye-GS

Gaussian Splatting

MCMC, 2M,
bilateral grid

Gaussian Splatting



A Gaussian has properties: center position, color, covariance matrix, and opacity

Slide from Matias Turkulainen

Why MLPs?

- Relighting
  - RNG: Relightable Neural Gaussians
  - A Diffusion Approach to Radiance Field Relighting using Multi-Illumination Synthesis
- Ambient Motion
  - Modeling Ambient Scene Dynamics for Free-view Synthesis
- Level of Detail
  - Scaffold-GS, Octree-GS
- Background Modeling
- Appearance
- Deblurring
- Compression

Appearance

Screenshot taken from: https://nerfbaselines.github.io/

Appearance

```
  means: (N, 3)
 scales: (N, 3)
  quats: (N, 3)
opacities: (N,)
 colors: (N, 1, 3)
features: (N, 32)
```

```python
class AppearanceOptModule(torch.nn.Module):
    """Appearance optimization module."""

        self.embeds = torch.nn.Embedding(n, embed_dim)
        self.color_mlp = create_mlp(
            in_dim=embed_dim + feature_dim + (sh_degree + 1) ** 2,
            num_layers=mlp_depth + 1,
            layer_width=mlp_width,
            out_dim=3,
            initialize_last_layer_zeros=True,
        )
```

```python
if TCNN_EXISTS:
    return _create_mlp_tcnn(
        in_dim,
        num_layers,
        layer_width,
        out_dim,
        initialize_last_layer_zeros=initialize_last_layer_zeros,
    )
else:
    return _create_mlp_torch(
```

Appearance

```
   means: (N, 3)
  scales: (N, 3)
   quats: (N, 3)
opacities: (N,)
  colors: (N, 1, 3)
features: (N, 32)
```

```python
class AppearanceOptModule(torch.nn.Module):
    """Appearance optimization module."""

    def forward(self, features: Tensor, image_ids: Tensor, dirs: Tensor) -> Tensor:
        embeds = self.embeds(image_ids).repeat(*features.shape[:-1], 1)
        dirs = encode_dirs(dirs)
        mlp_in = torch.cat([embeds, features, dirs], dim=-1)
        colors = self.color_mlp(mlp_in)
        return colors
```

```python
colors = colors + self.params["colors"]
colors = torch.sigmoid(colors)
```

Appearance

Photo Tourism

Photo Tourism is a dataset of images of famous landmarks, such as the Sacre Coeur, the Trevi Fountain, and the Brandenburg Gate. The images were captured by tourist at different times of the day and year, images have varying lighting conditions and occlusions. The evaluation protocol is based on NeRF-W, where the image appearance embeddings are optimized on the left side of the image and the metrics are computed on the right side of the image.

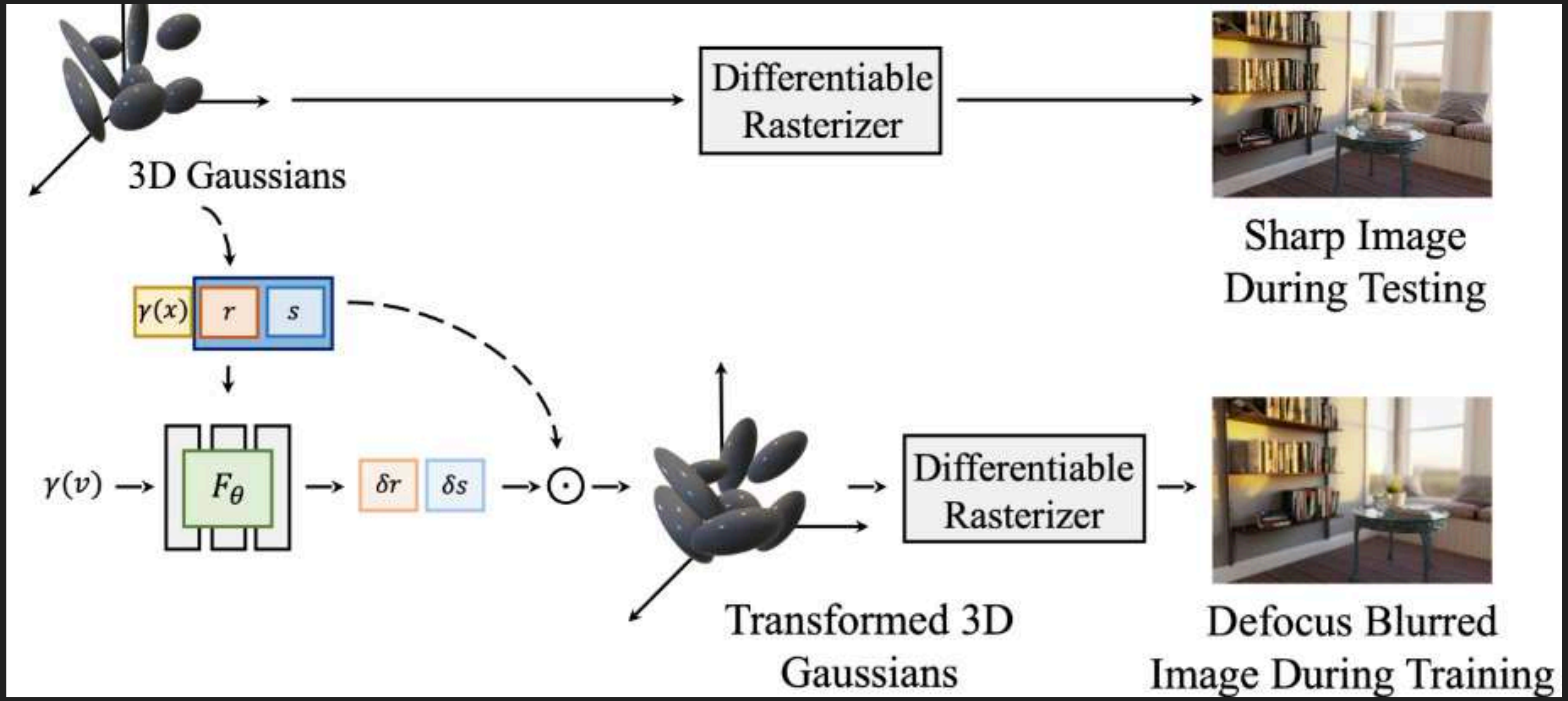
	Method	PSNR	SSIM	LPIPS	Time	GPU mem.
	K-Planes	21.10	0.761	0.313	24m 37s	3.59 GB
	GS-W	21.38	0.817	0.213	1h 13m 50s	21.93 GB
	NeRF-W (reimplementation)	21.75	0.790	0.268	44h 23m 46s	98.80 GB
	Scaffold-GS	23.50	0.854	0.170	1h 27m 49s	18.34 GB
	gsplat	23.66	0.857	0.162	1h 44m 24s	4.68 GB
	WildGaussians	24.65	0.851	0.179	10h 18m 16s	18.24 GB

Screenshot taken from: https://nerfbaselines.github.io/

Deblurring

Deblurring

Original Paper

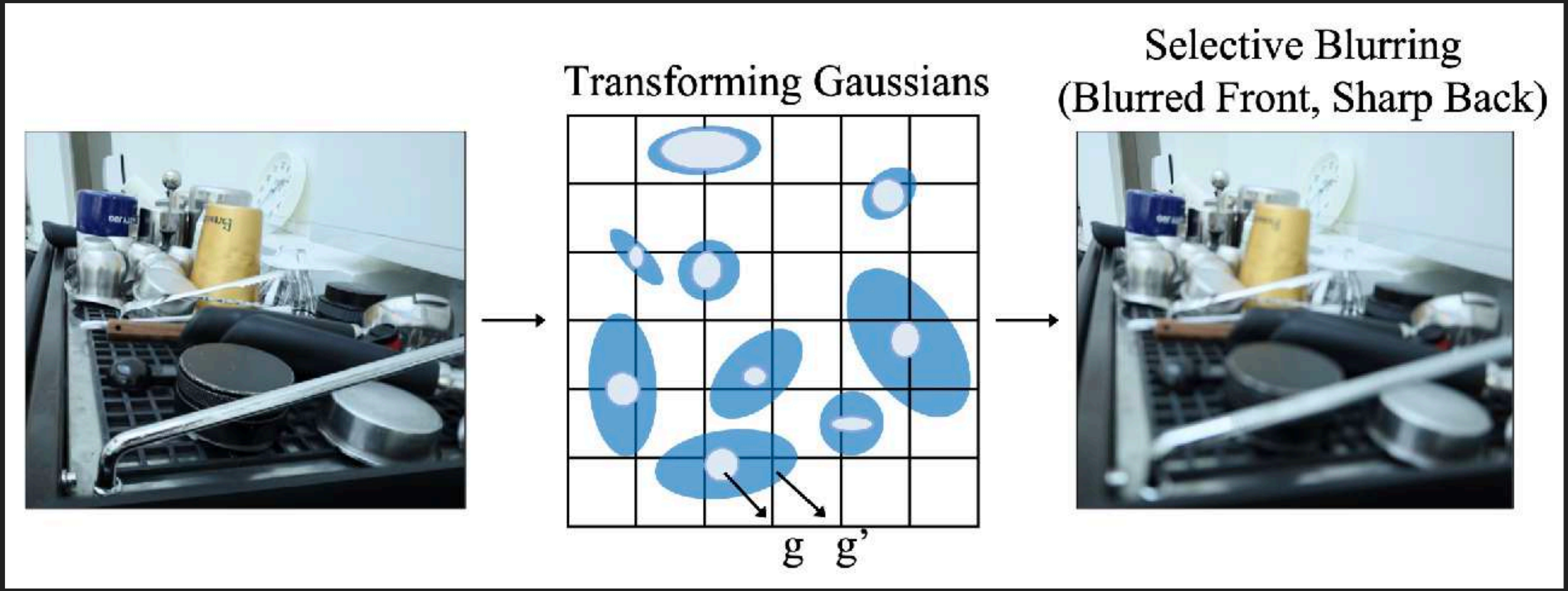


$$\hat{r}_j = r_j \cdot \min(1.0,\ \lambda_s \delta r_j + (1 - \lambda_s))$$
$$\hat{s}_j = s_j \cdot \min(1.0,\ \lambda_s \delta s_j + (1 - \lambda_s))$$

Screenshots taken from: https://benhenryl.github.io/Deblurring-3D-Gaussian-Splatting/

Deblurring

Original Paper



Screenshots taken from: https://benhenryl.github.io/Deblurring-3D-Gaussian-Splatting/

Deblurring

Dataset

Deblurring

Implementation

```
means: (N, 3)
scales: (N, 3)
quats: (N, 3)
opacities: (N,)
sh0: (N, 1, 3)
shN: (N, 15, 3)
```

```python
class BlurOptModule(nn.Module):
    """Blur optimization module."""

    def __init__(self, n: int, embed_dim: int = 4):
        self.embeds = torch.nn.Embedding(n, embed_dim)
        self.means_encoder = get_encoder(num_freqs=3, input_dims=3)
        self.blur_deltas_mlp = create_mlp(
            in_dim=embed_dim + self.means_encoder.out_dim + 7,
            num_layers=5,
            layer_width=64,
            out_dim=7,
        )

    def forward(self, image_ids: Tensor, means: Tensor, scales: Tensor, quats: Tensor):
        quats = F.normalize(quats, dim=-1)
        means_emb = self.means_encoder.encode(log_transform(means))
        images_emb = self.embeds(image_ids).repeat(means.shape[0], 1)
        mlp_out = self.blur_deltas_mlp(
            torch.cat([images_emb, means_emb, scales, quats], dim=-1)
        ).float()
        scales_delta = torch.clamp(mlp_out[:, :3], min=0.0, max=0.1)
        quats_delta = torch.clamp(mlp_out[:, 3:], min=0.0, max=0.1)
        scales = torch.exp(scales + scales_delta)
        quats = quats + quats_delta
        return scales, quats
```

Deblurring

Not working…

	Train PSNR	Val PSNR
3DGS-MCMC	29.61	24.73
With blur optimization	34.36	24.32

Deblurring

Not working…

	Train PSNR	Val PSNR
3DGS-MCMC	29.61	24.73
With blur optimization	34.36	24.32

Uses view direction as a proxy for per-image embedding

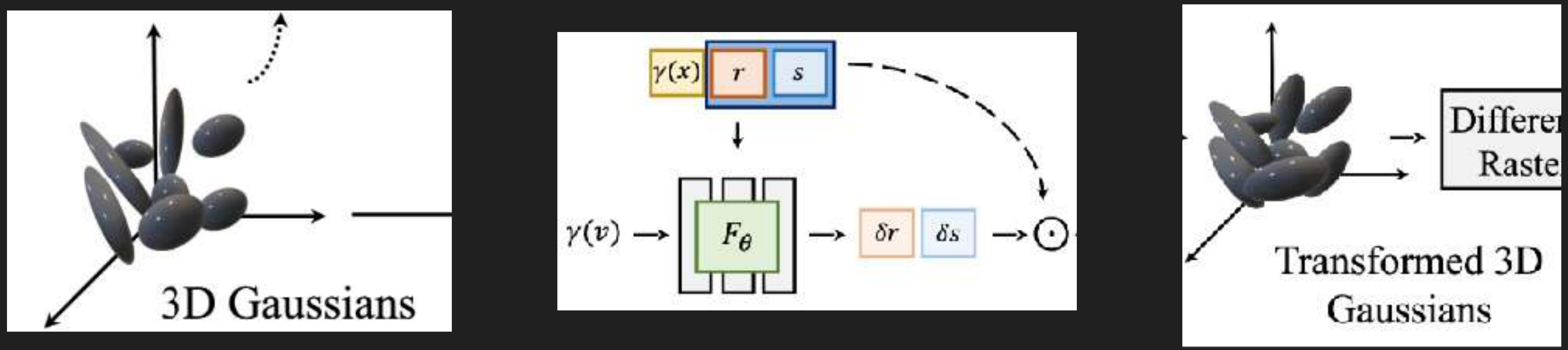
Deblurring

Not working…

	Train PSNR	Val PSNR
3DGS-MCMC	29.61	24.73
With blur optimization	34.36	24.32

Deblurring

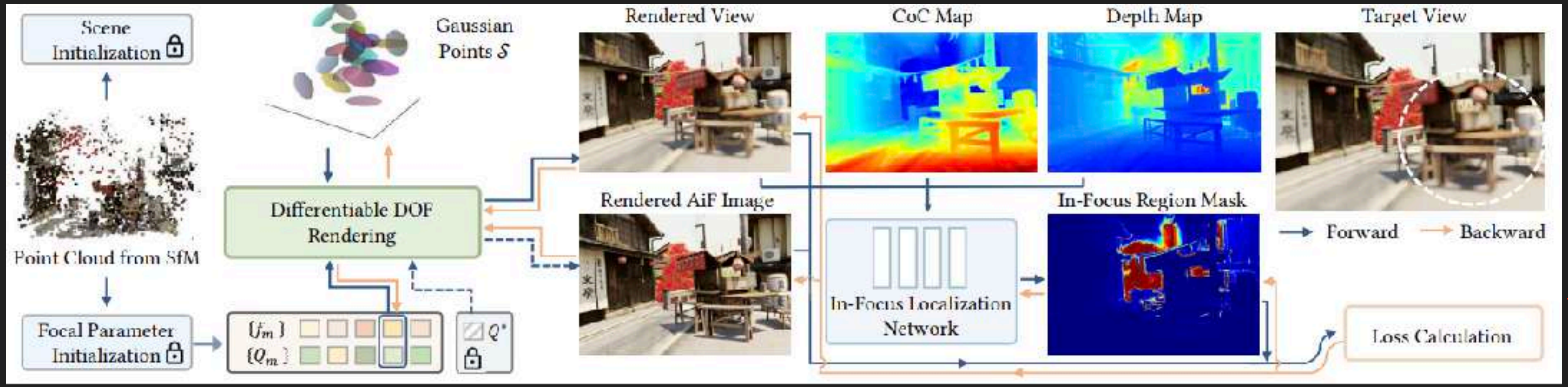
Not working…



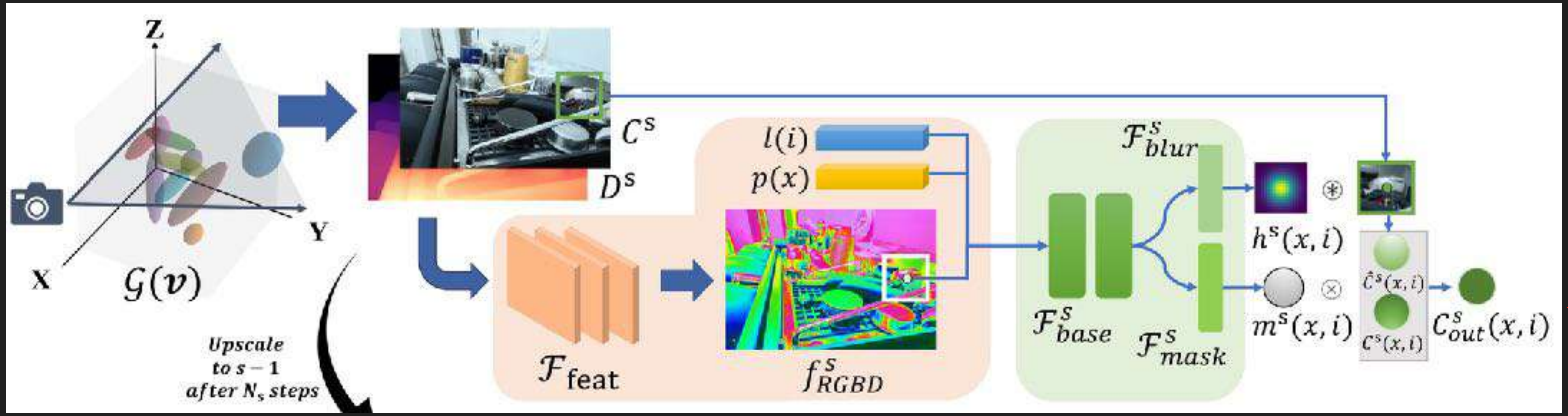
Deblurring

Other papers

- DOF-GS: Explicit modeling for defocus blur.



- BAGS: Fit 2D blur kernels at the same time as 3DGS.



Deblurring

Blur Mask

```python
class BlurOptModule(nn.Module):
    """Blur optimization module."""
    def __init__(self, n: int, embed_dim: int = 4):
        self.blur_masks = torch.nn.Parameter(torch.zeros(n, 400, 600, 1))

    def predict_mask(self, image_ids: Tensor):
        blur_mask = torch.sigmoid(self.blur_masks[image_ids])
```

```python
blur_mask = self.blur_module.predict_mask(image_ids, depths)
renders_blur, _, _ = self.rasterize_splats(
    camtoworlds=camtoworlds,
    Ks=Ks,
    width=width,
    height=height,
    sh_degree=sh_degree_to_use,
    near_plane=cfg.near_plane,
    far_plane=cfg.far_plane,
    image_ids=image_ids,
    render_mode="RGB",
    masks=masks,
    blur=True,
)
colors = (1 - blur_mask) * colors + blur_mask * renders_blur[..., 0:3]
```

```python
if self.cfg.blur_opt and blur:
    scales, quats = self.blur_module(
        image_ids=image_ids,
        means=self.splats["means"],
        scales=self.splats["scales"],
        quats=self.splats["quats"],
    )
else:
    scales = torch.exp(self.splats["scales"])
    quats = self.splats["quats"]  # [N, 4]
```

Deblurring

Blur Mask

```python
class BlurOptModule(nn.Module):
    """Blur optimization module."""
    def __init__(self, n: int, embed_dim: int = 4):
        self.blur_masks = torch.nn.Parameter(torch.zeros(n, 400, 600, 1))
    def predict_mask(self, image_ids: Tensor):
        blur_mask = torch.sigmoid(self.blur_masks[image_ids])
```

Deblurring

Blur Mask

```python
class BlurOptModule(nn.Module):
    """Blur optimization module."""
    def __init__(self, n: int, embed_dim: int = 4):
        self.blur_masks = torch.nn.Parameter(torch.zeros(n, 40, 60, 1))

    def predict_mask(self, image_ids: Tensor):
        x = self.blur_masks[image_ids]
        x = F.interpolate(x.permute(0, 3, 1, 2), scale_factor=(10, 10), mode='bilinear').permute(0, 2, 3, 1)
        blur_mask = torch.sigmoid(x)
```

Better than baseline!

Deblurring

Blur MLP

```python
class BlurOptModule(nn.Module):
    """Blur optimization module."""
    def __init__(self, n: int, embed_dim: int = 4):
        self.embeds = torch.nn.Embedding(n, embed_dim)
        self.depths_encoder = get_encoder(num_freqs=3, input_dims=1)
        self.grid_encoder = get_encoder(num_freqs=1, input_dims=2)
        self.blur_mask_mlp = create_mlp(
            in_dim=embed_dim + self.depths_encoder.out_dim + self.grid_encoder.out_dim,
            num_layers=5,
            layer_width=64,
            out_dim=1,
        )
    def predict_mask(self, image_ids: Tensor):
```




```python
        blur_mask = torch.sigmoid(mlp_out)
```

Deblurring

Blur MLP Regularization

```python
class BlurOptModule(nn.Module):
    """Blur optimization module."""
    def mask_loss(self, blur_mask: Tensor):
        x = blur_mask.mean()
        maskloss = torch.abs(x)
```

$$\mathcal{L}_s = \lambda_{\text{photo}} \|C^s_{\text{out}} - C^s_{\text{obs}}\| + \lambda_{\text{DS}}\mathcal{L}_{\text{D-SSIM}}(C^s_{\text{out}}, C^s_{\text{obs}}) + \lambda_{\text{mask}}\|m^s\|, \qquad (8)$$

From BAGS

Deblurring

Blur MLP Regularization

```
class BlurOptModule(nn.Module):
    """Blur optimization module."""
    def mask_loss(self, blur_mask: Tensor):
        x = blur_mask.mean()
        maskloss = torch.abs(x)
        maskloss = x**2
```

$$\mathcal{L}_s = \lambda_{\text{photo}} \| C^s_{\text{out}} - C^s_{\text{obs}} \| + \lambda_{\text{DS}} \mathcal{L}_{\text{D-SSIM}}(C^s_{\text{out}}, C^s_{\text{obs}}) + \lambda_{\text{mask}} \| m^s \|, \quad (8)$$

From BAGS

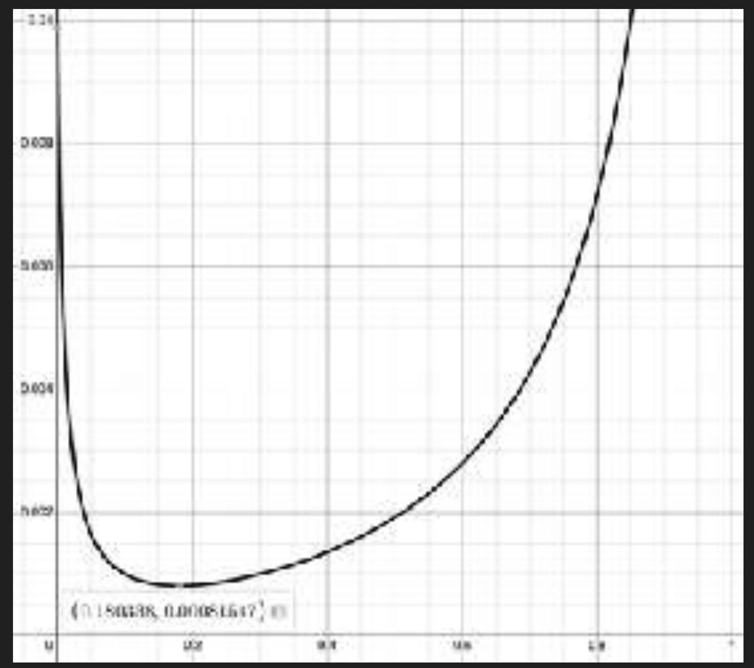
Deblurring

Blur MLP Regularization

```python
class BlurOptModule(nn.Module):
    """Blur optimization module."""
    def mask_loss(self, blur_mask: Tensor):
        x = blur_mask.mean()
        maskloss = torch.abs(x)
        maskloss = x**2
        maskloss = lambda_a * 1 / (1 - x + eps) * lambda_b * (1 / (x + eps))
```

$$\mathcal{L}_s = \lambda_{\text{photo}} \| C^s_{\text{out}} - C^s_{\text{obs}} \| + \lambda_{\text{DS}} \mathcal{L}_{\text{D-SSIM}} (C^s_{\text{out}}, C^s_{\text{obs}}) + \lambda_{\text{mask}} \| m^s \|, \quad (8)$$

From BAGS



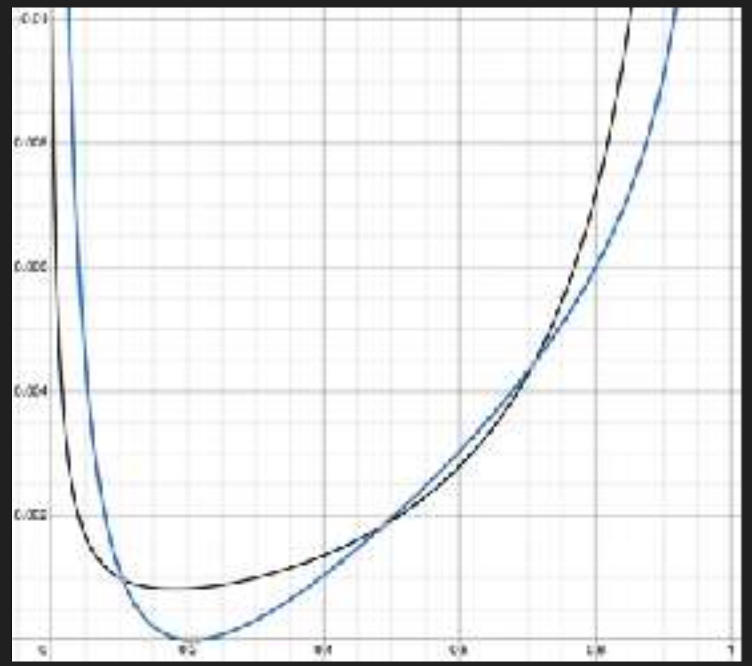
Deblurring

Blur MLP Regularization

```python
class BlurOptModule(nn.Module):
    """Blur optimization module."""
    def mask_loss(self, blur_mask: Tensor):
        x = blur_mask.mean()
        maskloss = torch.abs(x)
        maskloss = x**2
        maskloss = lambda_a * 1 / (1 - x + eps) * lambda_b * (1 / (x + eps))
        maskloss = lambda_a * x + lambda_b * (1 / (1 - x + eps) + 1 / (x + eps)) + c
```

$$\mathcal{L}_s = \lambda_{\text{photo}} \| C^s_{\text{out}} - C^s_{\text{obs}} \| + \lambda_{\text{DS}} \mathcal{L}_{\text{D-SSIM}}(C^s_{\text{out}}, C^s_{\text{obs}}) + \lambda_{\text{mask}} \| m^s \|, \quad (8)$$

From BAGS



```python
self.bounded_l1_loss = bounded_l1_loss(10.0, 0.5)
maskloss = self.bounded_l1_loss(x)
```

Deblurring

Results

	defocuscake	defocuscaps	defocuscisco	defocuscoral	defocuscupcake	defocuscups	defocusdaisy	defocussausage	defocusseal	defocustools	AVERAGE
Ours	26.80	24.30	20.47	19.37	22.25	25.28	23.63	18.47	25.75	27.22	23.35 😢
Deblur-GS	26.88	24.50	20.83	19.78	22.11	26.28	23.54	18.99	26.18	27.96	23.71
BAGS	27.21	24.16	20.79	20.53	22.93	26.27	23.74	18.76	26.52	28.60	23.95
DOF-GS	-	-	-	-	-	-	-	-	-	-	24.12

Deblurring

Results

Deblurring

Results

Deblurring

Failed Experiments

- Use ground truth as input to mask MLP.

- Add depth and depth_blur as input to mask MLP.

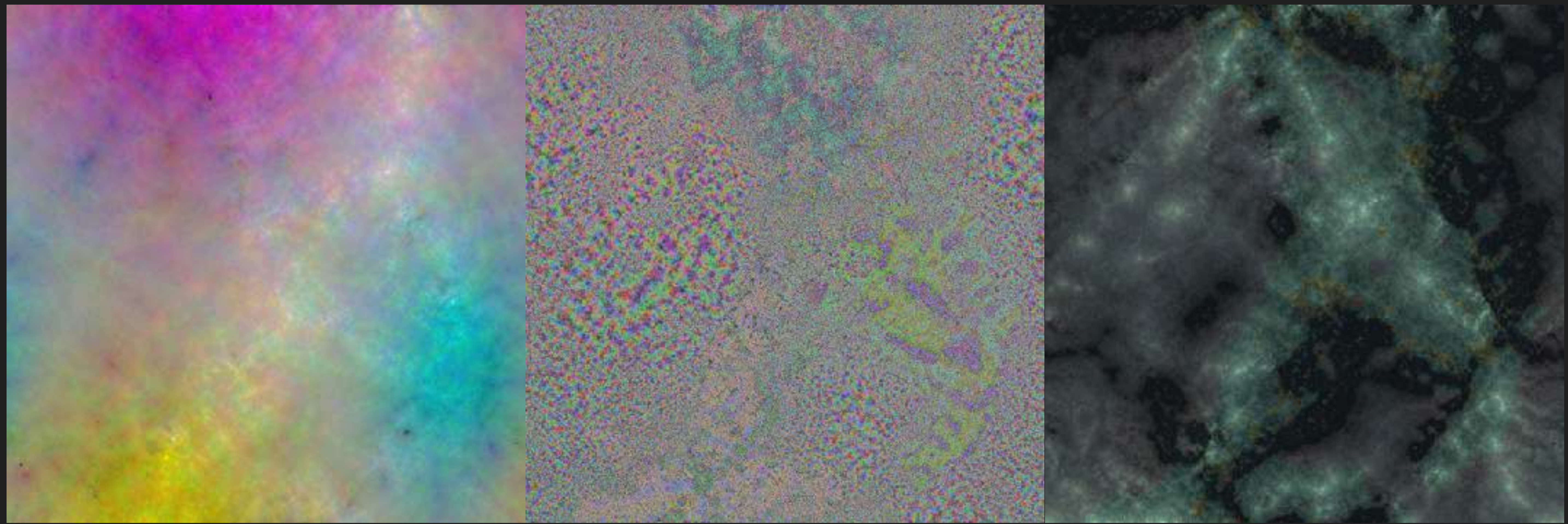
Deblurring

Failed Experiments

- Add color as input to mask MLP.

- Use a mask CNN instead of mask MLP.

Compression



Compression

`class PngCompression(use_sort: bool = True, verbose: bool = True)` [source]

Uses quantization and sorting to compress splats into PNG files and uses K-means clustering to compress the spherical harmonic coefficents.

> **Warning**
>
> This class requires the imageio, plas and torchpq packages to be installed.

> **Warning**
>
> This class might throw away a few lowest opacities splats if the number of splats is not a square number.

> **Note**
>
> The splats parameters are expected to be pre-activation values. It expects the following fields in the splats dictionary: "means", "scales", "quats", "opacities", "sh0", "shN". More fields can be added to the dictionary, but they will only be compressed using NPZ compression.

REFERENCES

- Compact 3D Scene Representation via Self-Organizing Gaussian Grids
- Making Gaussian Splats more smaller

Screenshot taken from: https://docs.gsplat.studio/main/apis/compression.html

Compression

Default vs MCMC

	PSNR	SSIM	LPIPS	Num GSs	Mem (GB)	Time (min)
gsplat (default settings)	29.00	0.87	0.14	3237318	5.62	19.39
absgrad	29.11	0.88	0.12	2465986	4.40	18.10
antialiased	29.03	0.87	0.14	3377807	5.87	19.52
mcmc (1 mill)	29.18	0.87	0.14	1000000	1.98	15.42
mcmc (2 mill)	29.53	0.88	0.13	2000000	3.43	21.79
mcmc (3 mill)	29.65	0.89	0.12	3000000	4.99	27.63
absgrad & antialiased	29.14	0.88	0.13	2563156	4.57	18.43
mcmc & antialiased	29.23	0.87	0.14	1000000	2.00	15.75

Screenshot taken from: https://docs.gsplat.studio/main/tests/eval.html

Compression

MCMC

```
   means: (N, 3)
  scales: (N, 3)
   quats: (N, 3)
opacities: (N,)
     sh0: (N, 1, 3)
     shN: (N, 15, 3)
```

```python
def _compress_npz(
    compress_dir: str, param_name: str, params: Tensor, **kwargs
) -> Dict[str, Any]:
    """Compress parameters with numpy's NPZ compression."""
    npz_dict = {"arr": params.detach().cpu().numpy()}
    save_fp = os.path.join(compress_dir, f"{param_name}.npz")
    os.makedirs(os.path.dirname(save_fp), exist_ok=True)
    np.savez_compressed(save_fp, **npz_dict)
    meta = {
        "shape": params.shape,
        "dtype": str(params.dtype).split(".")[1],
    }
    return meta
```

```
 11M  means.npz
 329  meta.json
3.5M  opacities.npz
 14M  quats.npz
 11M  scales.npz
 11M  sh0.npz
160M  shN.npz
```

```
208M  compression.zip
```

```
"psnr": 26.94044303894043,
"ssim": 0.8427160978317261,
"lpips": 0.14394041895866394,
```

-450 MB
+0.18 PSNR

All stats are on garden scene.

Compression

Quantization

```
def log_transform(x):
    return torch.sign(x) * torch.log1p(torch.abs(x))

grid = params.reshape((n_sidelen, n_sidelen, -1))
mins = torch.amin(grid, dim=(0, 1))
maxs = torch.amax(grid, dim=(0, 1))
grid_norm = (grid - mins) / (maxs - mins)
img_norm = grid_norm.detach().cpu().numpy()

img = (img_norm * (2**8 - 1)).round().astype(np.uint8)
img = img.squeeze()
imageio.imwrite(os.path.join(compress_dir, f"{param_name}.png"), img)

meta = {
    "shape": list(params.shape),
    "dtype": str(params.dtype).split(".")[1],
    "mins": mins.tolist(),
    "maxs": maxs.tolist(),
}
return meta
```

```
2.9M   means_l.png
2.3M   means_u.png
2.8K   meta.json
745K   opacities.png
3.8M   quats.png
2.8M   scales.png
2.6M   sh0.png
 34M   shN.npz

 49M   compression.zip
```

```
"psnr": 26.902538299560547,
"ssim": 0.8414101004600525,
"lpips": 0.14493945240974426,
```

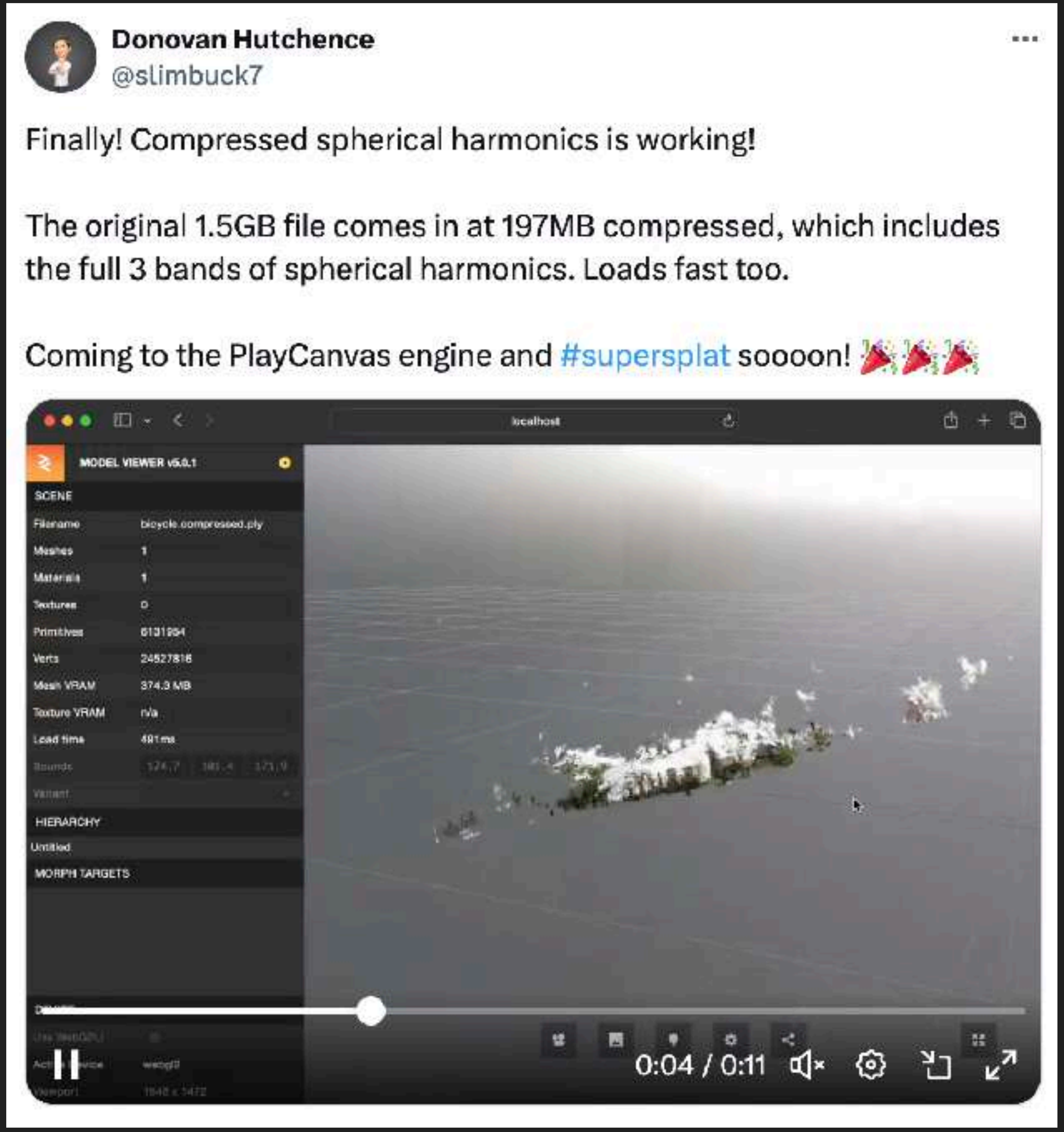
-159 MB

-0.04 PSNR

All stats are on garden scene.

Compression

KMeans Clustering



Compression

KMeans Clustering

```
n_clusters: int = 65536,
quantization: int = 6,
```

```
kmeans = KMeans(n_clusters=n_clusters, distance="manhattan", verbose=verbose)
x = params.reshape(params.shape[0], -1).permute(1, 0).contiguous()
labels = kmeans.fit(x)
```

```
npz_dict = {
    "centroids": centroids_quant,
    "labels": labels,
}
np.savez_compressed(os.path.join(compress_dir, f"{param_name}.npz"), **npz_dict)
```

```
params = centroids[labels]
```

```
2.9M   means_l.png
2.3M   means_u.png
1.1K   meta.json
745K   opacities.png
3.8M   quats.png
2.8M   scales.png
2.6M   sh0.png
3.4M   shN.npz
```

```
19M   compression.zip
```

```
"psnr": 26.571739196777344,
"ssim": 0.8375915288925171,
"lpips": 0.15935097634792328,
```

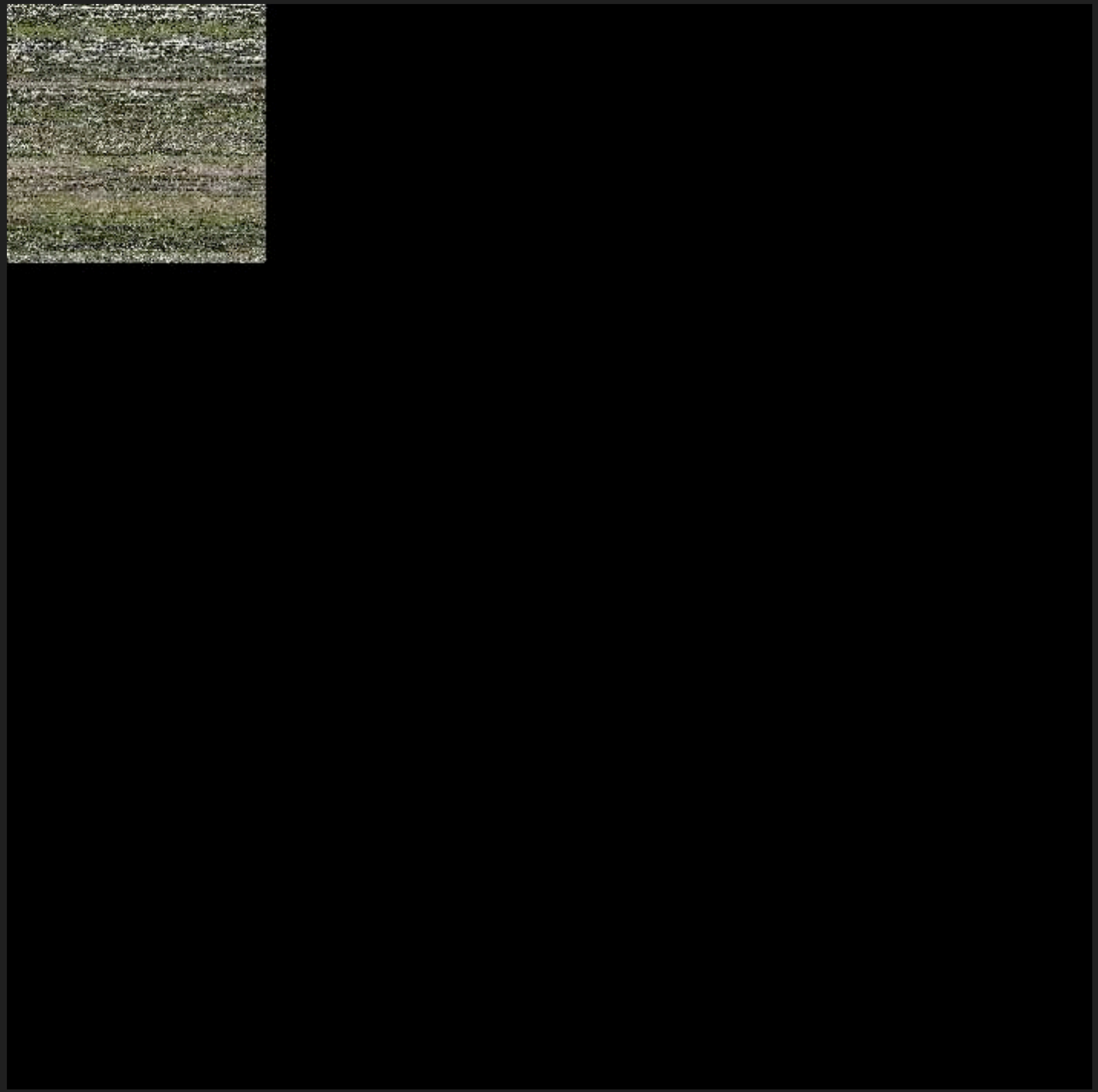
-30 MB

-0.33 PSNR

All stats are on garden scene.

Compression

Sorting



```
2.7M   means_l.png
382K   means_u.png
1.1K   meta.json
734K   opacities.png
3.7M   quats.png
2.7M   scales.png
2.5M   sh0.png
3.4M   shN.npz
 16M   compression.zip
```

```
"psnr": 26.571739196777344,
"ssim": 0.8375915288925171,
"lpips": 0.15935097634792328,
```

-3 MB

-0.0 PSNR

All stats are on garden scene.

Compression

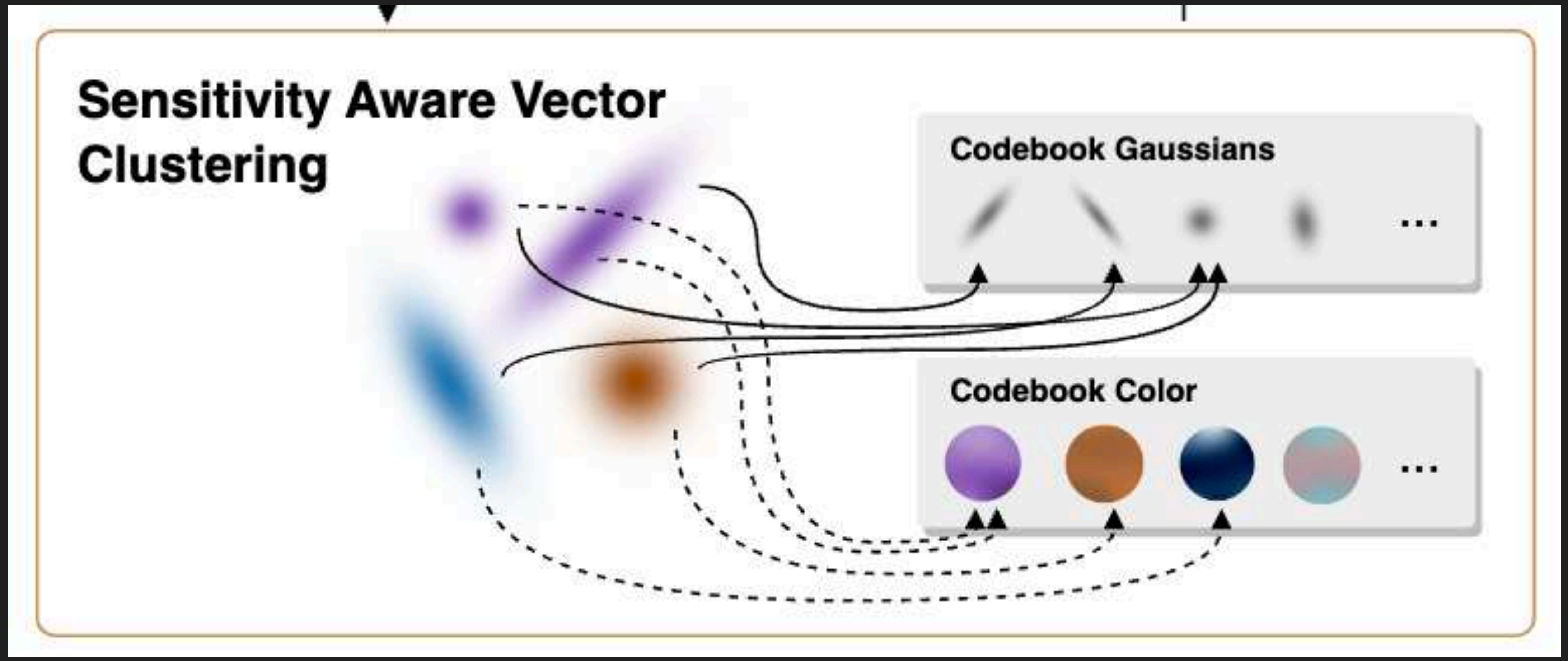
Ranking

Method	Rank	TanksAndTemples				MipNeRF360			
		PSNR	SSIM	LPIPS	Size [MB]	PSNR	SSIM	LPIPS	Size [MB]
HAC-highrate	4.8	24.40	0.853	0.177	11.2	27.59	0.809	0.234	22.5
HAC-lowrate	4.9	24.04	0.846	0.187	8.1	27.30	0.803	0.246	14.4
gsplat-1.00M	5.1	24.03	0.857	0.163	15.4	27.29	0.811	0.229	15.3
IGS-Low	5.6	23.70	0.836	0.227	8.4	27.33	0.809	0.257	12.5
IGS-High	5.7	24.05	0.849	0.210	12.5	27.62	0.819	0.247	25.4
Morgenstern et al. w/o SH	5.9	25.27	0.857	0.217	8.2	27.02	0.803	0.232	16.7
Morgenstern et al.	7.4	25.63	0.864	0.208	21.4	27.64	0.814	0.220	40.3
Navaneet et al. 32K	7.6	23.44	0.838	0.198	13.0	27.12	0.806	0.240	19.0
Navaneet et al. 16K	7.8	23.39	0.836	0.200	12.0	27.03	0.804	0.243	18.0
RDO-Gaussian	8.2	23.34	0.835	0.195	11.5	27.05	0.802	0.239	22.4

Ranking taken from: https://w-m.github.io/3dgs-compression-survey/

Compression

shN Codebook



Not used

```
self.splats["shN_codebook"] = npz["centroids"]
self.splats["shN_indices"] = npz["labels"]
shN = self.splats["shN_codebook"][self.splats["shN_indices"].int()]

if cfg.shN_reg > 0.0:
    loss += cfg.shN_reg * torch.abs(self.splats["shN_codebook"]).mean()
```

Compression

JPG vs PNG

Paper's PSNR: 27.79

Authors evaluated on larger images which were downscaled to the target size (avoiding JPEG compression artifacts) instead of using the official provided downscaled images. As mentioned in the 3DGS paper, this increases results slightly ~0.5 dB PSNR.

images_4
- DSC07956.JPG

images_4
- DSC07956.png

```
2.9M   means_l.png
2.0M   means_u.png
1.1K   meta.json
505K   opacities.png
3.7M   quats.png
1.7M   scales.png
2.0M   sh0.png
3.4M   shN.npz
```

```
16M   compression.zip
```

```
"psnr": 26.887527465820312,
"ssim": 0.8451383113861084,
"lpips": 0.15079563856124878,
```

-0 MB

+0.31 PSNR

All stats are on garden scene.

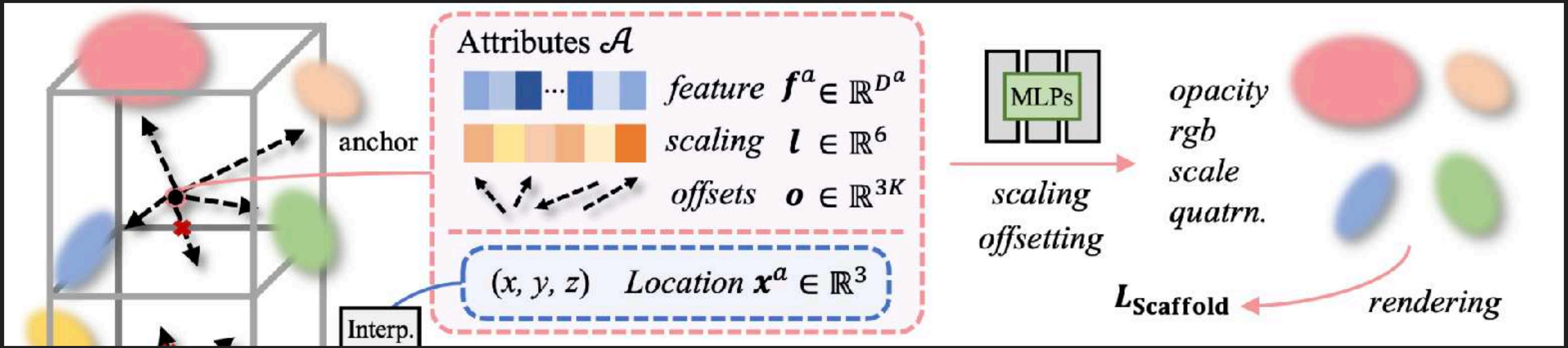
Compression

JPG vs PNG

	psnr	ssim	lpips	Size [MB]
HAC-lowrate	27.53	0.807	0.238	16.0
HAC-highrate	27.77	0.811	0.230	22.9
gsplat-1M	27.29	0.811	0.229	16.0
gsplat-1M w/ png_data	27.54	0.821	0.215	16.0

Compression

HAC



Screenshot taken from: https://yihangchen-ee.github.io/project_hac/

Compression

shN MLP

```python
class MlpOptModule(torch.nn.Module):
    """MLP optimization module."""

 self.shN_mlp = create_mlp(
     in_dim=self.means_encoder.out_dim + 3 + 8,
     num_layers=5,
     layer_width=64,
     out_dim=((sh_degree + 1) ** 2 - 1) * 3,
 )
```

```python
means_emb = self.means_encoder.encode(log_transform(means))
sh0_emb = sh0[:, 0, :]
quats_emb = F.normalize(quats, dim=-1)
opacities_emb = opacities[:, None]
mlp_in = torch.cat(
    [means_emb, sh0_emb, quats_emb, scales, opacities_emb], dim=-1
)
mlp_out = self.shN_mlp(mlp_in)
shN = mlp_out.reshape(means.shape[0], -1, 3)
return shN
```

```
2.9M   means_l.png
1.8M   means_u.png
 932   meta.json
 36K   mlp_module.pt
376K   opacities.png
3.7M   quats.png
1.6M   scales.png
1.7M   sh0.png
 12M   compression.zip
```

```
"psnr": 27.07998275756836,
"ssim": 0.8467192649841309,
"lpips": 0.14368483424186707,
```

-4 MB
+0.19 PSNR

All stats are on garden scene.

Compression

shN MLP

	psnr	ssim	lpips	Size [MB]	
HAC-lowrate	27.53	0.807	0.238	16.0	
HAC-highrate	27.77	0.811	0.230	22.9	
gsplat-1M	27.29	0.811	0.229	16.0	
gsplat-1M w/ png_data	27.54	0.821	0.215	16.0	😎
gsplat-1M w/ shN_mlp	27.50	0.817	0.219	12.2	😈

Compression

quats+scales MLP

```
    means: (N, 3)
opacities: (N,)
      sh0: (N, 1, 3)
 features: (N, 4)
```

```python
self.mlp = create_mlp(
    in_dim=self.means_encoder.out_dim + feature_dim + 4,
    num_layers=5,
    layer_width=64,
    out_dim=7 + ((sh_degree + 1) ** 2 - 1) * 3,
    initialize_last_layer_zeros=True,
)
```

```python
def forward(self, means: Tensor, opacities: Tensor, sh0: Tensor, features: Tensor):
    means_emb = self.means_encoder.encode(log_transform(means))
    opacities_emb = opacities[:, None]
    sh0_emb = sh0[:, 0, :]
    mlp_in = torch.cat([means_emb, opacities_emb, sh0_emb, features], dim=-1)
    mlp_out = self.mlp(mlp_in).float()

    quats = mlp_out[:, :4]
    scales = mlp_out[:, 4:7]
    shN = mlp_out[:, 7:]
```

```
2.6M   features.png
2.9M   means_l.png
1.7M   means_u.png
 741   meta.json
 66K   mlp_module.pt
450K   opacities.png
1.6M   sh0.png
```

```
9.1M  compression.zip
```

```
"psnr": 27.047954559326172,
"ssim": 0.8433372974395752,
"lpips": 0.14846640825271606,
```

-2.9 MB

-0.04 PSNR

All stats are on garden scene.

Compression

quats+scales MLP

	psnr	ssim	lpips	Size [MB]	
HAC-lowrate	27.53	0.807	0.238	16.0	
HAC-highrate	27.77	0.811	0.230	22.9	
gsplat-1M	27.29	0.811	0.229	16.0	
gsplat-1M w/ png_data	27.54	0.821	0.215	16.0	😎
gsplat-1M w/ shN_mlp	27.50	0.817	0.219	12.2	😈
gsplat-1M w/ cov_mlp	-	-	-	9.1	🤯

Unstable. MCMC requires quats and scales to compute noise.

Compression

Results

	psnr	ssim	lpips	Size [MB]	
HAC-lowrate	27.53	0.807	0.238	16.0	
HAC-highrate	27.77	0.811	0.230	22.9	
gsplat-1M	27.29	0.811	0.229	16.0	
gsplat-1M w/ png_data	27.54	0.821	0.215	16.0	😎
gsplat-1M w/ shN_mlp	27.50	0.817	0.219	12.2	😈
gsplat-2M w/ shN_mlp	27.85	0.827	0.198	24.1	😈

Recommendations

- Rerun MipNerf360 evaluation with PNGs.
- Promote a compression format.
  - MLP-decoded shN?
- Create OptModules interface.
  - Simplify simple_trainer.py.
  - Abstract away module-specifics configs, optimizers, loss, schedules, etc.
  - Share code with Nerfstudio.
  - Create a zoo of modules.
    - CameraOptModule, AppearanceOptModule, BilateralOptModule, MLPOptModule, BlurOptModule, BackgroundOptModule

Thank you!