OpenDriveLab

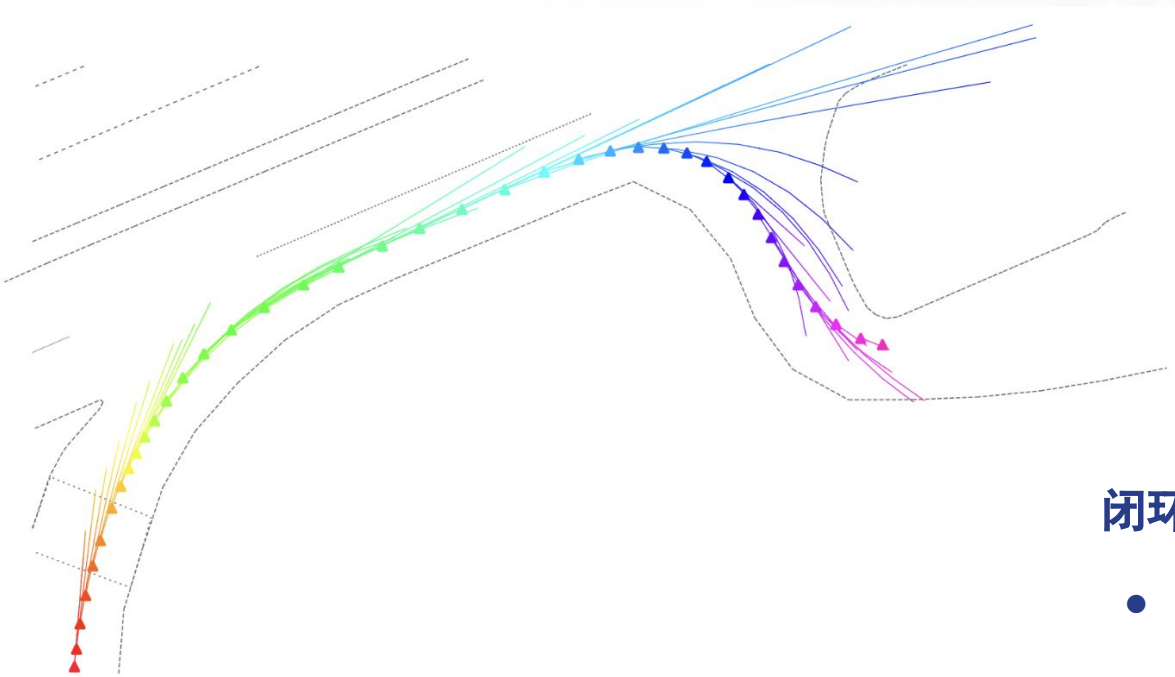
上海人工智能实验室
Shanghai Artificial Intelligence Laboratory

自动驾驶场景的三维重建

李天羽 | OpenDriveLab

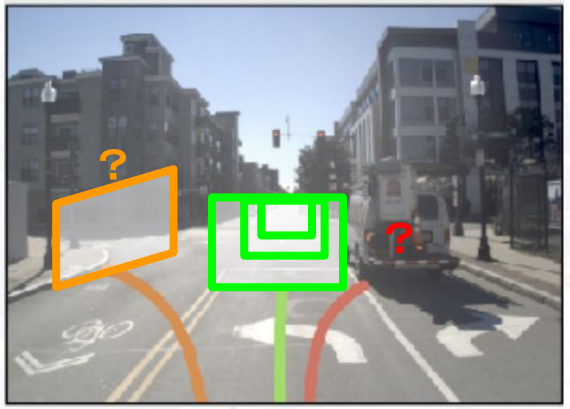
2024.06.09

自动驾驶中的三维重建丨背景介绍



From: Is Ego Status All You Need for Open-Loop End-to-End Autonomous Driving?

端到端开环 vs 闭环



闭环模拟器 基础功能:

- 新视角合成
- 移动、添加、删除物体

自动驾驶中的三维重建丨背景介绍

渲染引擎 (UE/ Blender)

真实性？

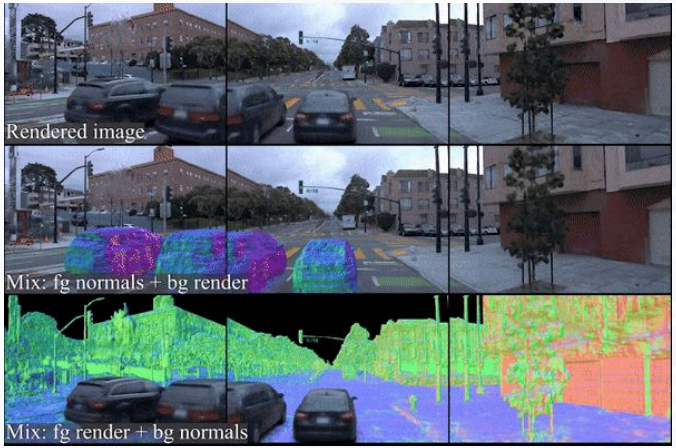
可控性？

GAIA-1 / Sora / Video Generator

From: Is Ego Status All You Need for Open-Loop End-to-End Autonomous Driving?

闭环模拟器 基础功能：

- 新视角合成
- 移动、添加、删除物体

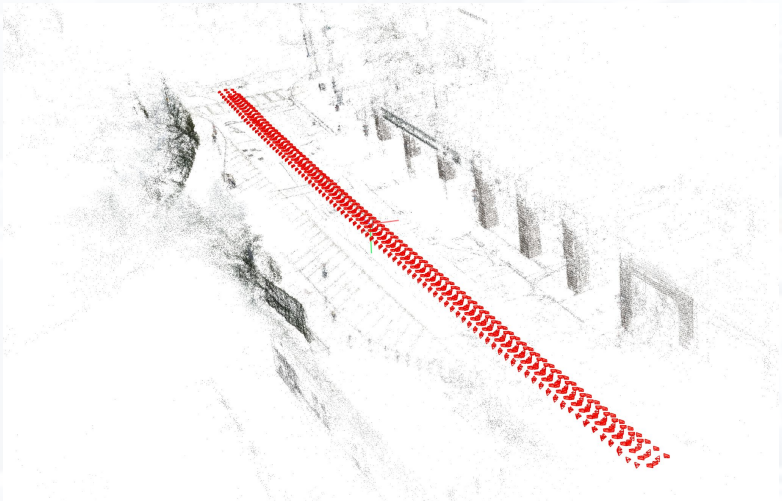


NeRF / 3D GS / Reconstruction

✓

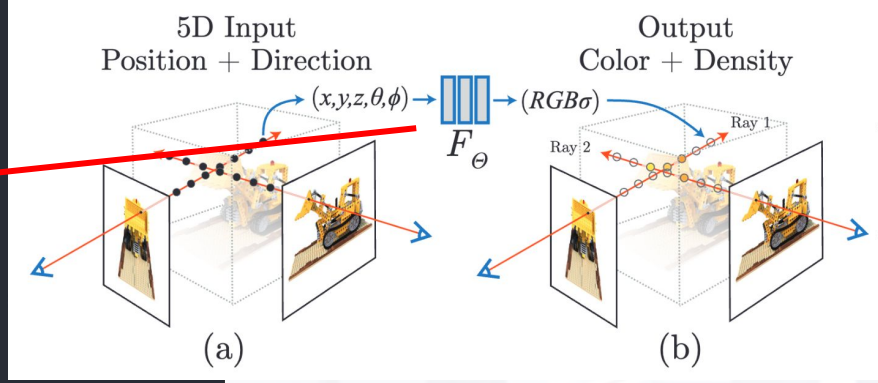
自动驾驶中的三维重建 | NeRF

- 输入:一组多视角图像
- 输出:能根据视角渲染图像的模型
- Neural Radiance Fields (NeRF)

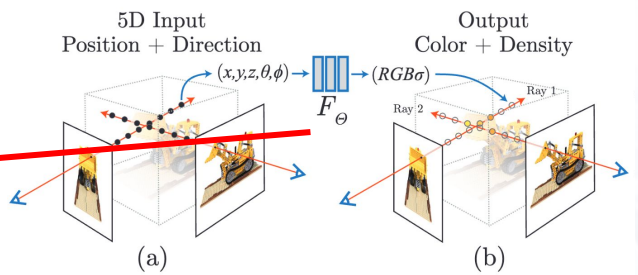


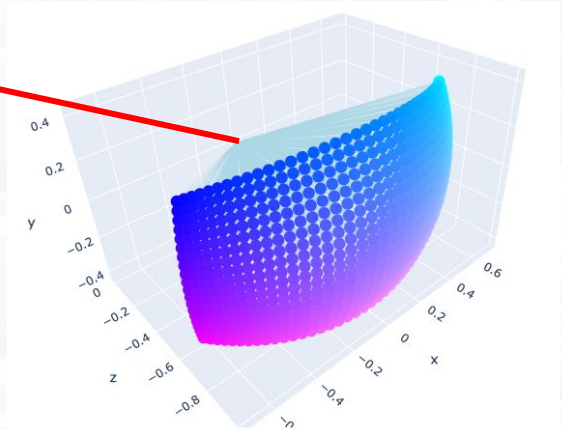
正在前进的自动驾驶车辆，具有环视摄像头

```python
class SimpleNeRF(torch.nn.Module):

    def __init__( ... ):
        self.field = MLP(input_dim=5, output_dim=4)

    def forward(camera_ray):
        # camera_ray: some class that contains (origin,

        # (n_samples, 5) x, y, z, r, g
        sample_points = sample_points_along_ray(camera_ray)

        # (n_samples, 4)
        color_with_density = self.field(sample_points)
        pred_rgb = volume_rendering(color_with_density)

        return pred_rgb
        # output.shape : (n_samples, 4)

    def forward_train(camera_ray, gt_rgb):
        pred_rgb = self(camera_ray)
        loss = MSE(pred_rgb, gt_rgb)
        return loss

```

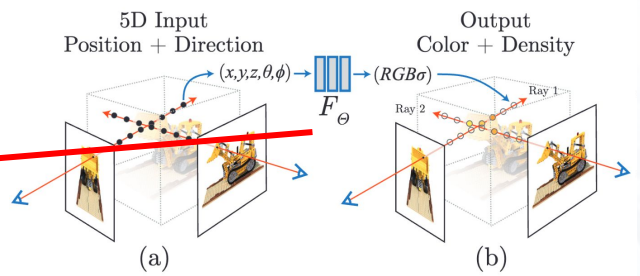


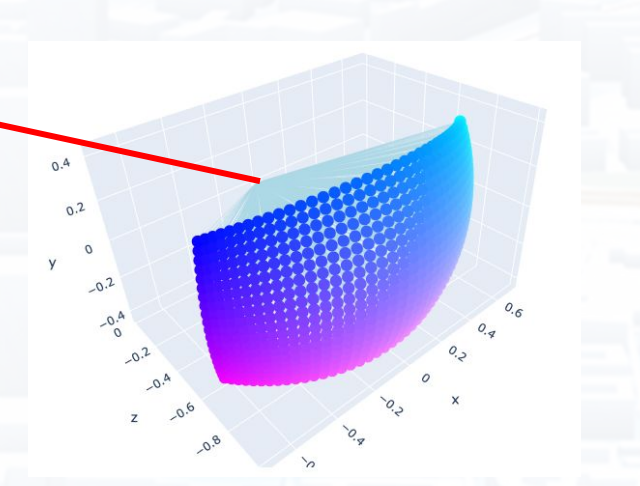
```python
class SimpleNeRF(torch.nn.Module):

    def __init__( ... ):
        self.field = MLP(input_dim=5, output_dim=4)

    def forward(camera_ray):
        # camera_ray: some class that contains (origin, direction)

        # (n_samples, 5) x, y, z, r, g
        sample_points = sample_points_along_ray(camera_ray)

        # (n_samples, 4)
        color_with_density = self.field(sample_points)
        pred_rgb = volume_rendering(color_with_density)

        return pred_rgb
        # output.shape : (n_samples, 4)

    def forward_train(camera_ray, gt_rgb):
        pred_rgb = self(camera_ray)
        loss = MSE(pred_rgb, gt_rgb)
        return loss
```

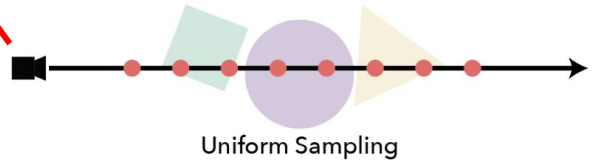




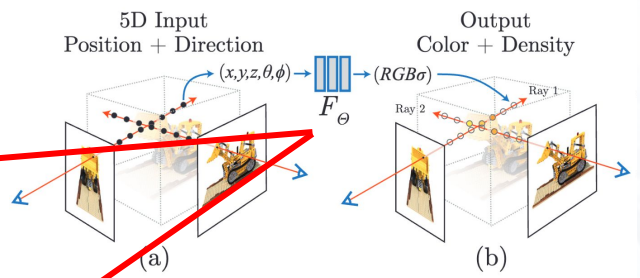
```python
class SimpleNeRF(torch.nn.Module):

    def __init__( ... ):
        self.field = MLP(input_dim=5, output_dim=4)

    def forward(camera_ray):
        # camera_ray: some class that contains (origin, direction)

        # (n_samples, 5) x, y, z, r, g
        sample_points = sample_points_along_ray(camera_ray)

        # (n_samples, 4)
        color_with_density = self.field(sample_points)
        pred_rgb = volume_rendering(color_with_density)

        return pred_rgb
        # output.shape : (n_samples, 4)

    def forward_train(camera_ray, gt_rgb):
        pred_rgb = self(camera_ray)
        loss = MSE(pred_rgb, gt_rgb)
        return loss
```

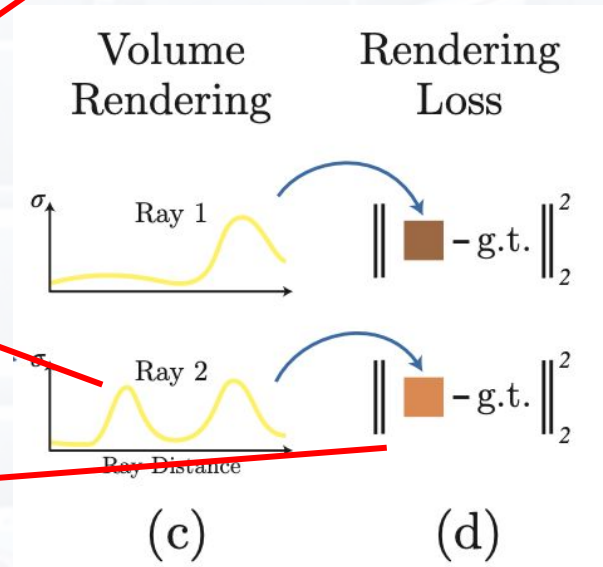




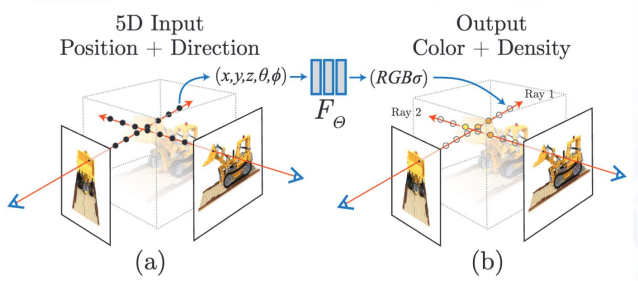


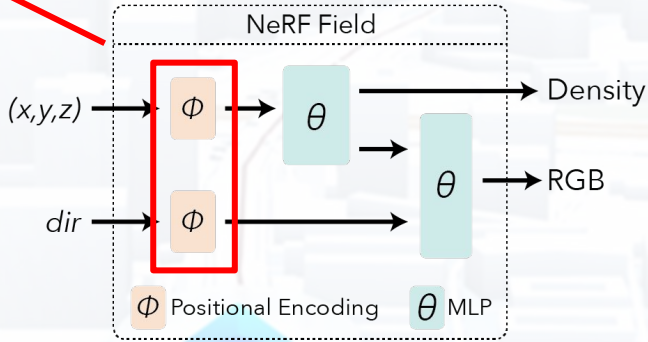
```python
class SimpleNeRF(torch.nn.Module):

    def __init__( ... ):
        self.field = MLP(input_dim=5, output_dim=4)

    def forward(camera_ray):
        # camera_ray: some class that contains (origin, direction)

        # (n_samples, 5) x, y, z, r, g
        sample_points = sample_points_along_ray(camera_ray)

        # (n_samples, 4)
        color_with_density = self.field(sample_points)
        pred_rgb = volume_rendering(color_with_density)

        return pred_rgb
        # output.shape : (n_samples, 4)

    def forward_train(camera_ray, gt_rgb):
        pred_rgb = self(camera_ray)
        loss = MSE(pred_rgb, gt_rgb)
        return loss
```





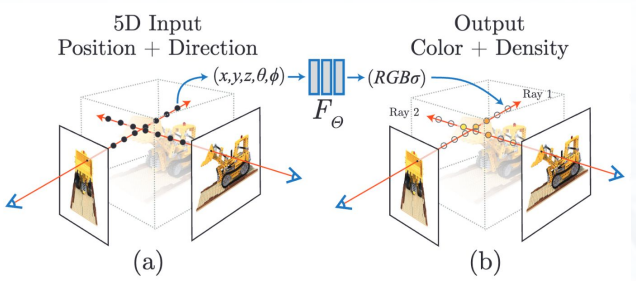
```python
class SimpleNeRF(torch.nn.Module):

    def __init__( ... ):
        self.field = MLP(input_dim=5, output_dim=4)

    def forward(camera_ray):
        # camera_ray: some class that contains (origin, direction)

        # (n_samples, 5) x, y, z, r, g
        sample_points = sample_points_along_ray(camera_ray)

        # (n_samples, 4)
        color_with_density = self.field(sample_points)
        pred_rgb = volume_rendering(color_with_density)

        return pred_rgb
        # output.shape : (n_samples, 4)
```

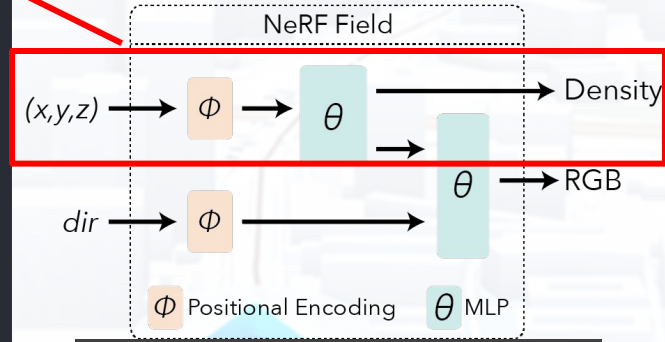




$$\gamma(p) = \left(\sin\left(2^0\pi p\right), \cos\left(2^0\pi p\right), \cdots, \sin\left(2^{L-1}\pi p\right), \cos\left(2^{L-1}\pi p\right)\right).$$

```python
class SimpleNeRF(torch.nn.Module):

    def __init__( ... ):
        self.field = MLP(input_dim=5, output_dim=4)

    def forward(camera_ray):
        # camera_ray: some class that contains (origin, direction)

        # (n_samples, 5) x, y, z, r, g
        sample_points = sample_points_along_ray(camera_ray)

        # (n_samples, 4)
        color_with_density = self.field(sample_points)
        pred_rgb = volume_rendering(color_with_density)

        return pred_rgb
        # output.shape : (n_samples, 4)

    def forward_train(camera_ray, gt_rgb):
        pred_rgb = self(camera_ray)
        loss = MSE(pred_rgb, gt_rgb)
        return loss
```





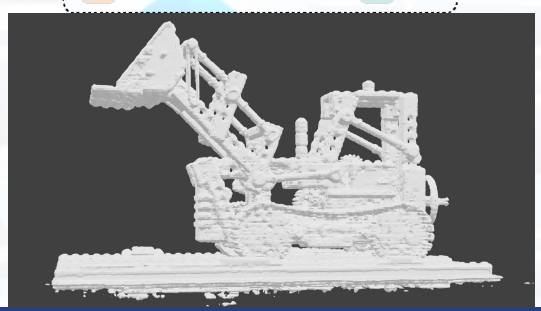
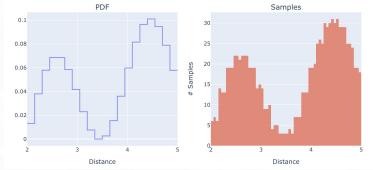
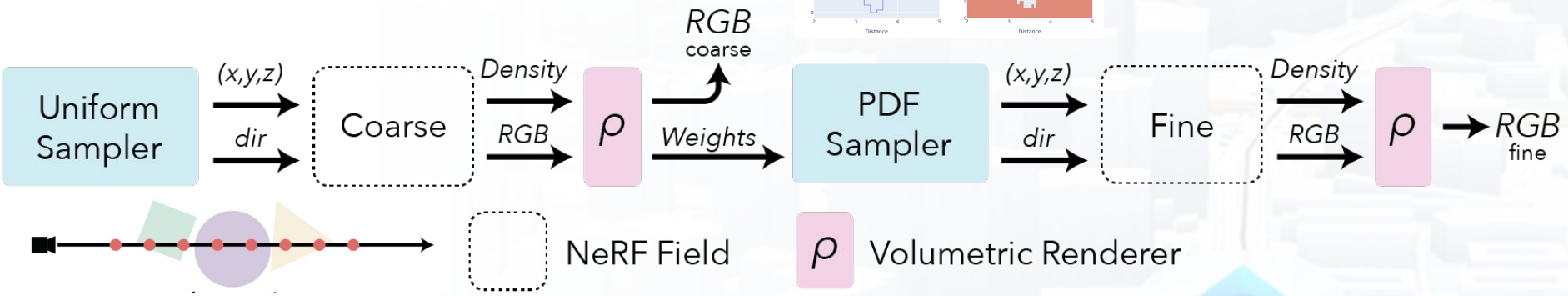


Figure is taken from nerfstudio document.

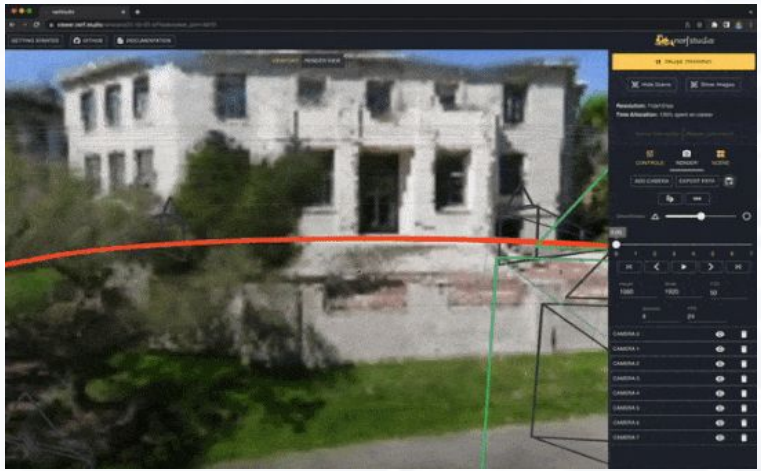
三维重建 | NeRF



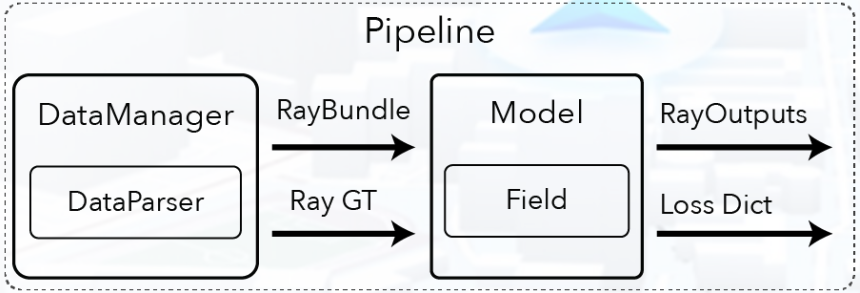


Coarse-to-fine

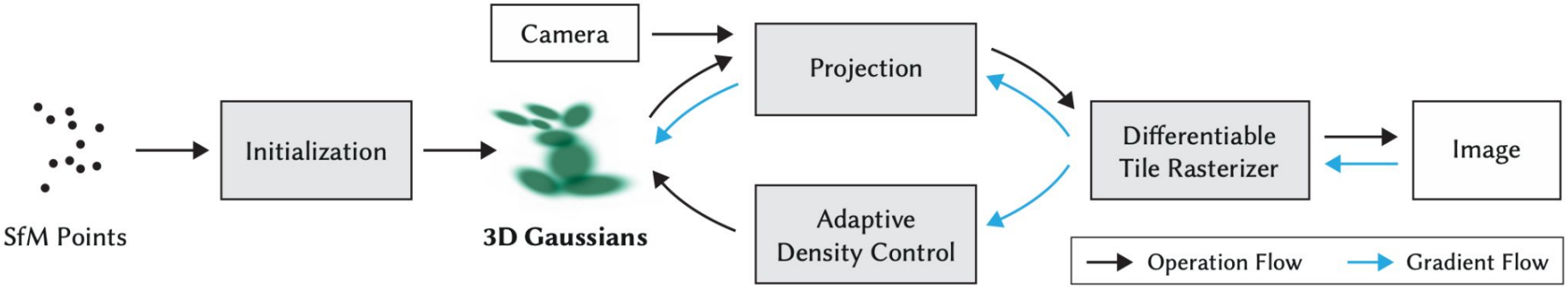
三维重建 | nerfstudio



- NeRF 界 mmcv/mmdet
- GitHub stars 8.7k+
- 自由浏览 Viewer



三维重建 | 3D Gaussian Splatting



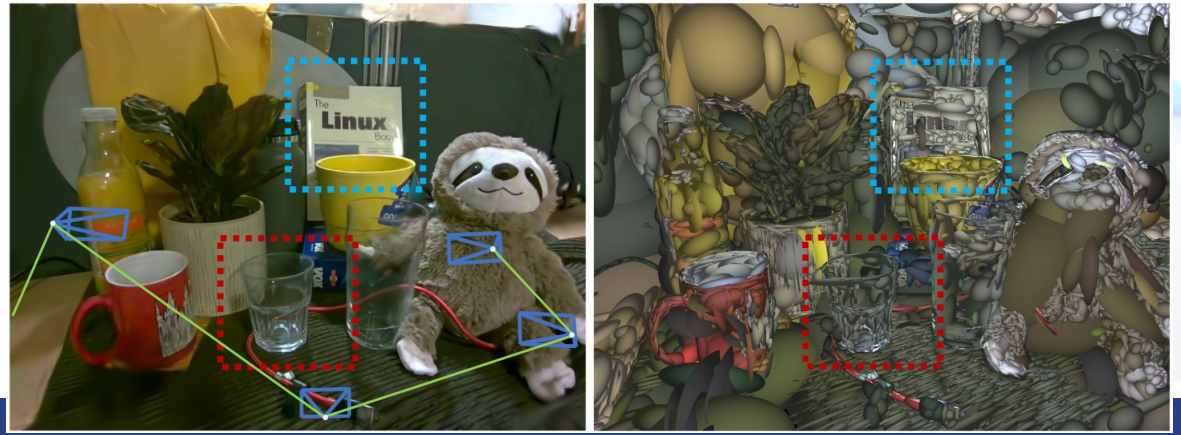
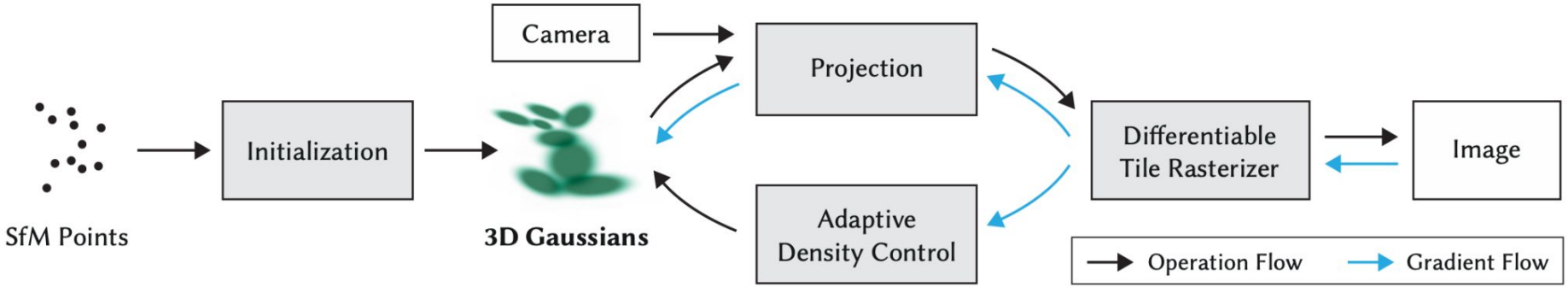


Figure is taken from 3DGS paper.

三维重建 | 3D Gaussian Splatting



- Gaussian Model: nn.Parameter with shape (n_pts, n_dim)
- n_dim: xyz, scale, rotation, opacity, …
- n_pts 如何变化？

三维重建 | 3D Gaussian Splatting

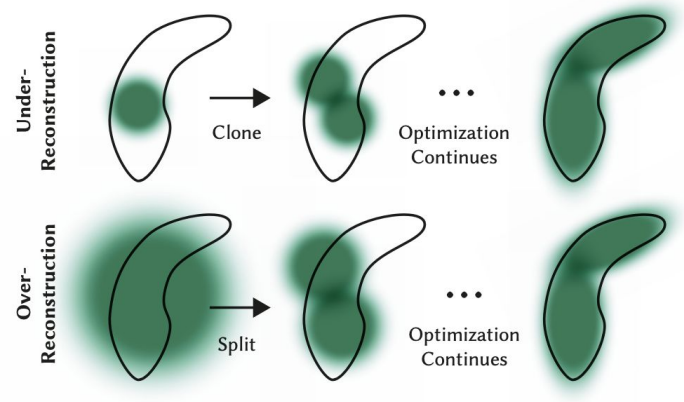
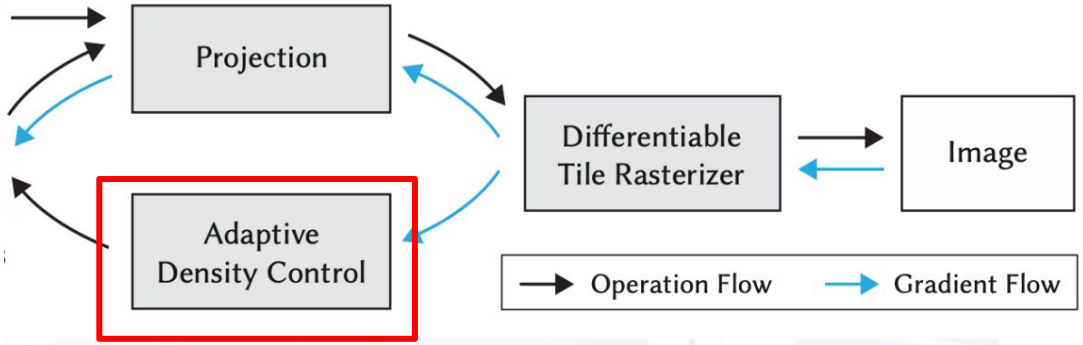


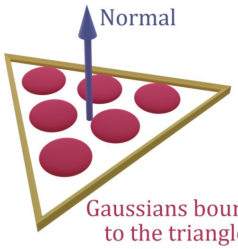
Fig. 4. Our adaptive Gaussian densification scheme. *Top row (under-reconstruction)*: When small-scale geometry (black outline) is insufficiently covered, we clone the respective Gaussian. *Bottom row (over-reconstruction)*: If small-scale geometry is represented by one large splat, we split it in two.



- n_dim: xyz, scale, rotation, opacity, …
- if grad_xyz > 0.1, clone or split
- if opacity < 0.1, delete
- reset opacity

三维重建 | 3D Gaussian Splatting

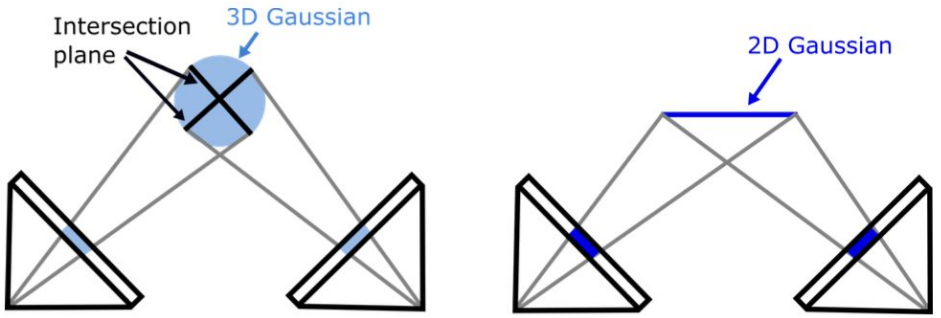
SuGaR, CVPR 2024



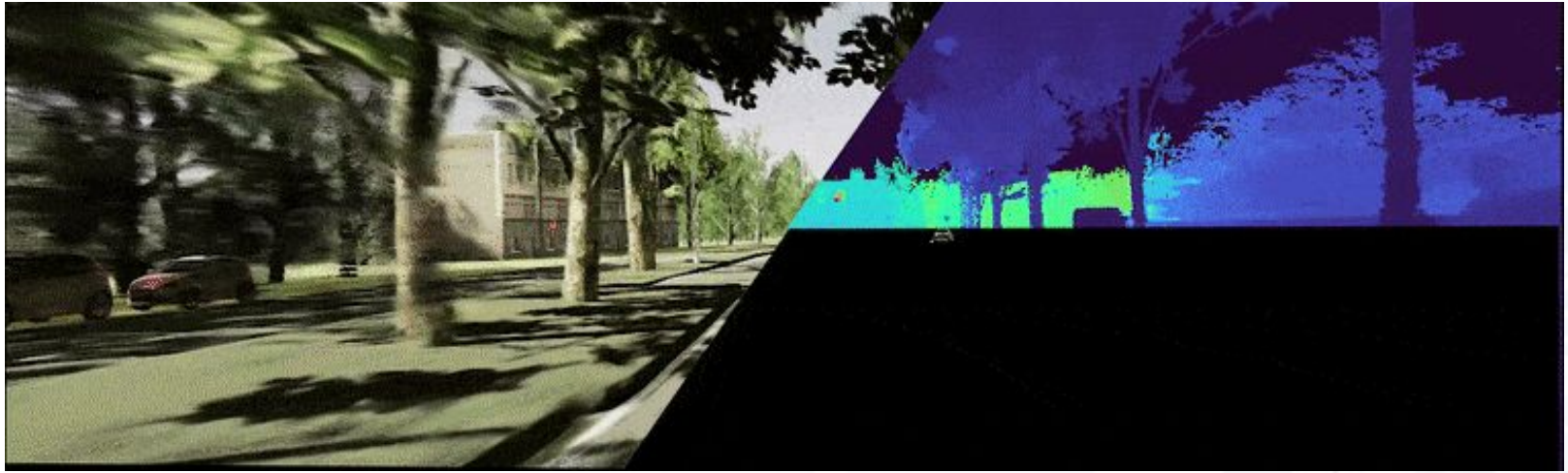
- Gaussians <—> geometry
- depth / normal / mesh
- regularization loss (normal)

三维重建 | 3D Gaussian Splatting

2D Gaussian, SIGGRAPH 2024

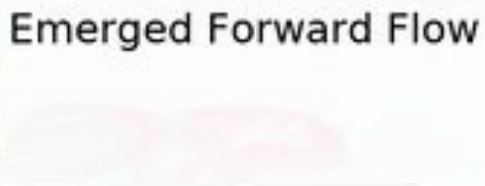


- 3D Gaussian → 2D Gaussian
- Gaussian Surfel
- regularization loss (depth, normal)



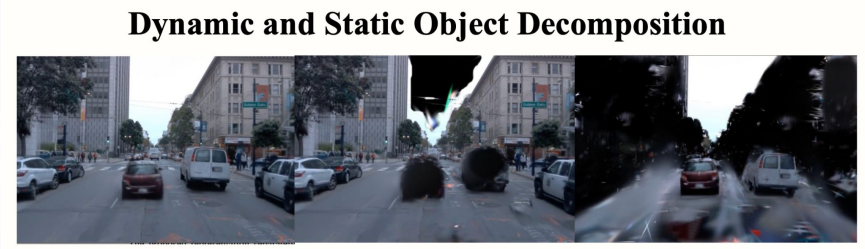
- 动静态/前后景分开建模：
- 2D mask，3D Tracking bbox
  - MARS，UniSim，NeuRAD
  - DrivingGaussian、StreetGaussian

自动驾驶中的三维重建



EmerNeRF

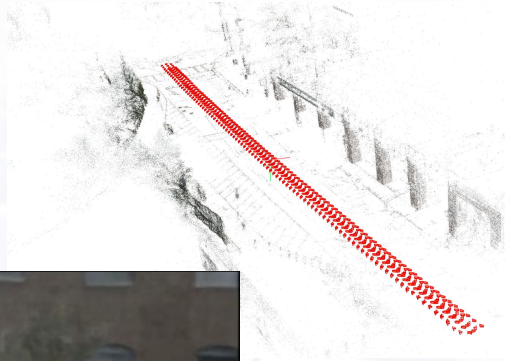
- 动静态/前后景分开建模：
- Self-Supervised: 3D Scene Flow
  - EmerNeRF
  - PVG、S3Gaussian



S3Gaussian

自动驾驶中的三维重建丨机遇与挑战

Figure is taken from NeuroNCAP website.



- 大幅度变换新视角能力差

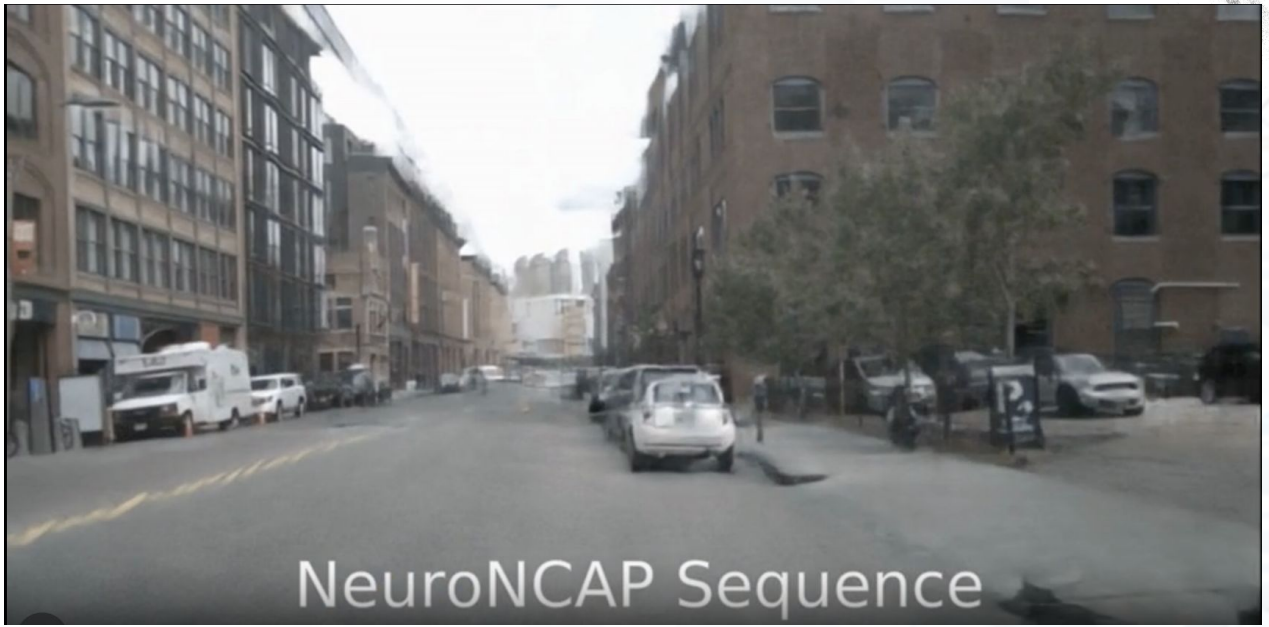


Table is taken from NeuRAD paper.

自动驾驶中的三维重建丨机遇与挑战

- 大幅度变换新视角能力差

		Ego shift			
		No shift	Lane 2m	Lane 3m	Vert. 1m
Panda FC	UniSim	-	74.7	97.5	-
	UniSim*	41.7	79.6	102.0	89.3
	NeuRAD	**25.0**	**72.3**	**93.9**	**76.3**
Panda 360	UniSim*	88.3	115.5	128.0	126.7
	NeuRAD	45.5	84.0	98.8	91.3
	NeuRAD w/ opt	**43.0**	**81.0**	**95.3**	**88.8**

自动驾驶中的三维重建丨机遇与挑战

- 动态物体放置

End