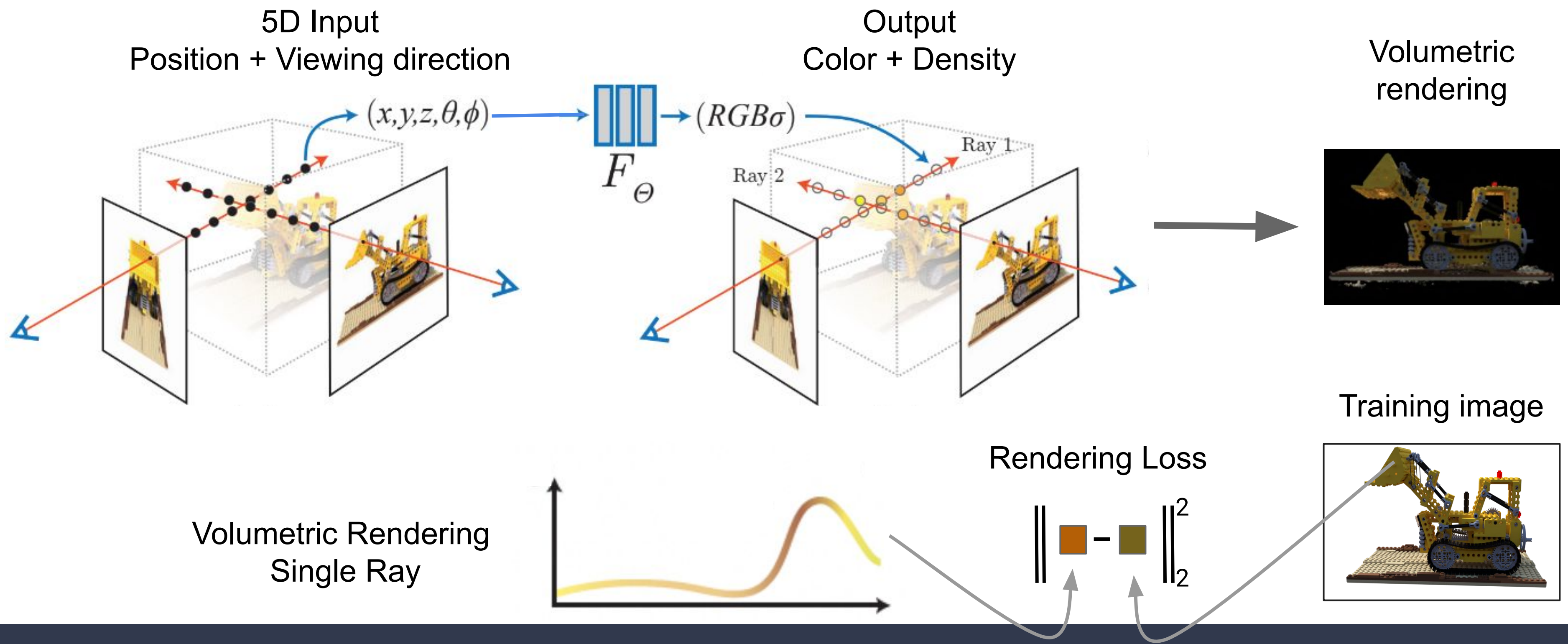
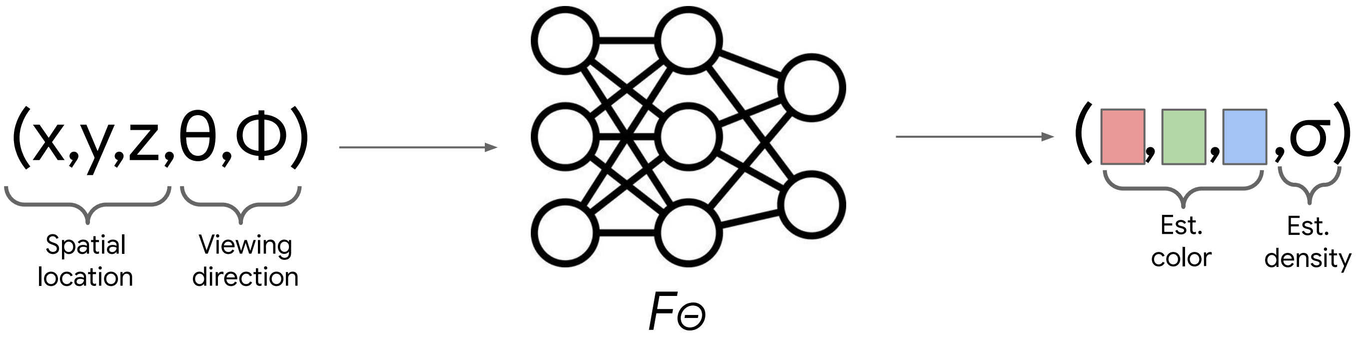
Neural Radiance Fields

ELLIS Summer School –
Large Scale AI

Recap - NeRFs

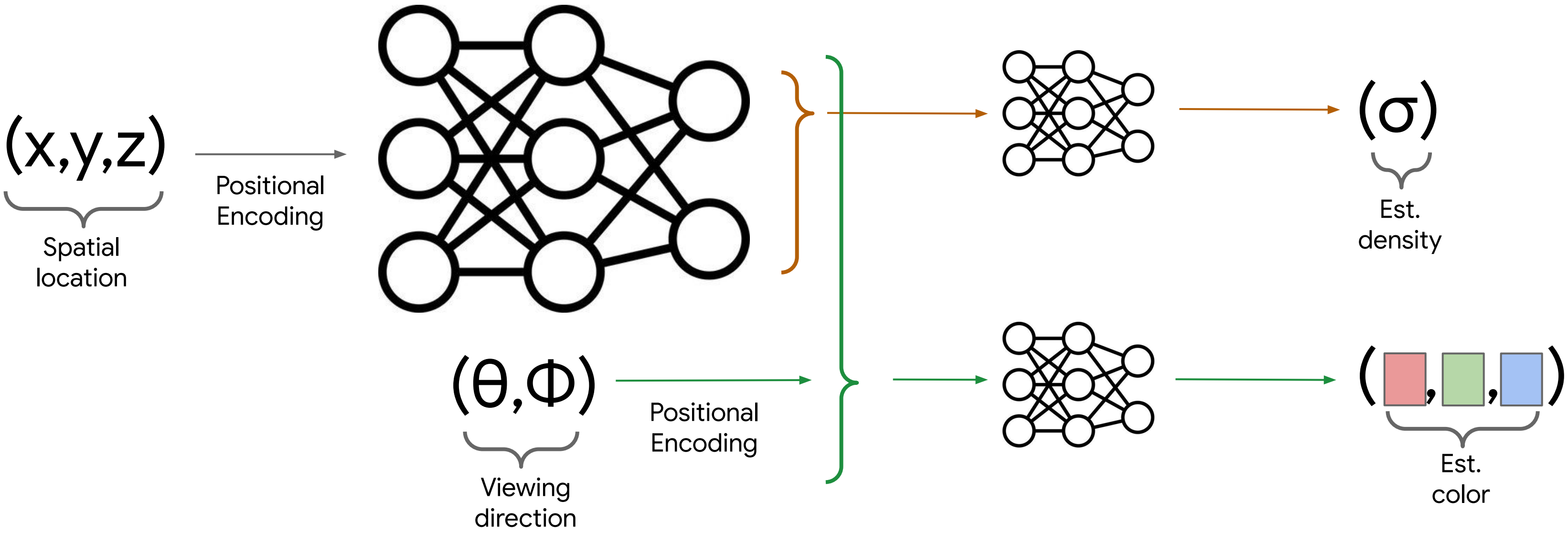
Training of NeRFs





- The network is a simple ReLU MLP that maps from location/view direction to color/density
- Density σ describes how solid/transparent a 3D point is (can model, e.g., fog)
- Conditioning on view direction allows for modeling view-dependent effects

One step further wrt before: learning density without 3D as input



- **Color and density are conditioned** on **3D input location**
  - While **color is conditioned** on **viewing direction** to model view-dependant artifacts such as lighting,
  - **density is not conditioned** on **it** as the object surface should not depend on the viewing direction
- **Positional encoding** (or other forms of encodings) are often employed to better deal with high frequency details
- Oftentimes, **multiple rounds of sampling** are employed to estimate color based on 3D locations near the surface

Libraries / Data

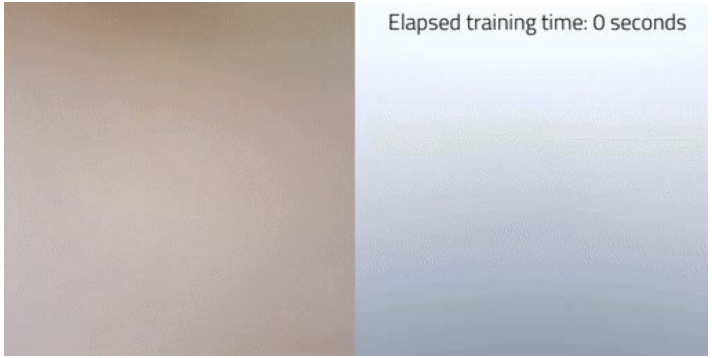
NerfAcc

NeRF-Factory

MultiNeRF

nerfstudio

Different Open-Source Libraries



K-Planes, CVPR 23

Temporal & Static Nerfs

Instruct Nerf2Nerf, ICCV 2023

3D Editing of NerFS with Text Prompts

Instant NGP, Siggraph 2022

Fast training(/inference) of NeRFs using trainable multi-level hash grids

Supported Models in NerfStudio

NeRFStudio - Supported models

Enables NeRFs to have a temporal dimension

NeRF together with 3D open-set semantic segmentation

Standard NeRF

Optimized training and rendering speed for NeRFs

Enables to edit NeRF with text prompts “Make it lego”

Unpublished work including several tricks such as pose refinement

NeRFStudio - Supported models

- Instant-NGP
- Instruct-NeRF2NeRF
- K-Planes
- LERF
- Mip-NeRF
- NeRF
- Nerfacto
- Nerfbusters
- NeRFPlayer
- Tetra-NeRF
- TensoRF
- Generfacto

Instant-NGP: Optimized training and rendering speed for NeRFs

Instruct-NeRF2NeRF: Enables to edit NeRF with text prompts “Make it lego”

K-Planes: Enables NeRFs to have a temporal dimension

LERF: NeRF together with 3D open-set semantic segmentation

NeRF: Standard NeRF

Nerfacto: Unpublished work including several tricks such as pose refinement

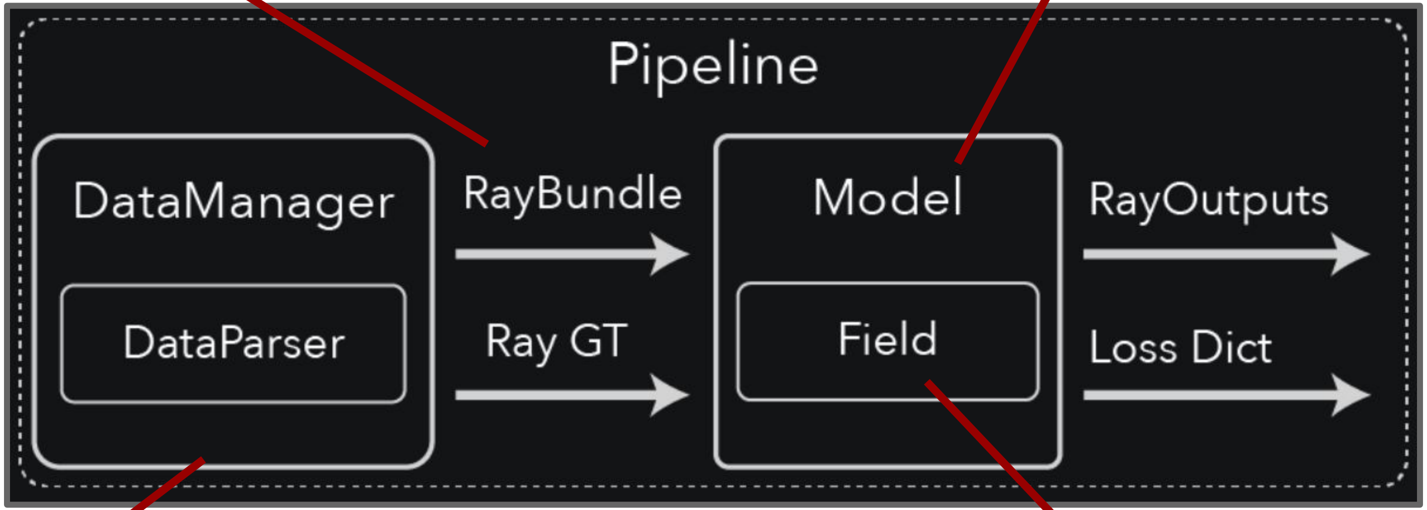
Supported Models in NerfStudio

RayBundle commonly includes

- Ray origin
- And ray direction

Defines the model, including:

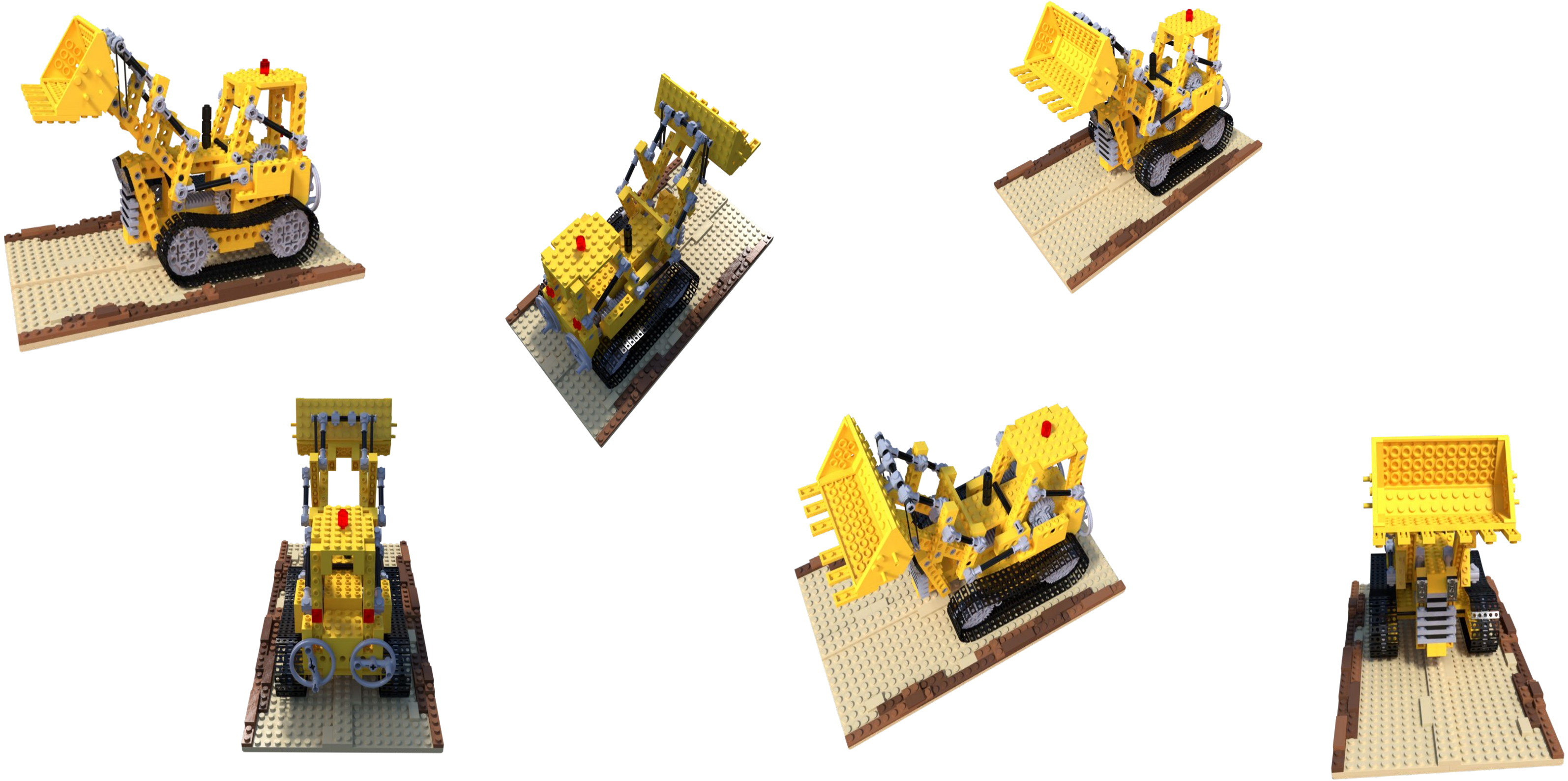
- Sampling of the points along the rays
- The chosen radiance fields and outputs
- The computed loss values



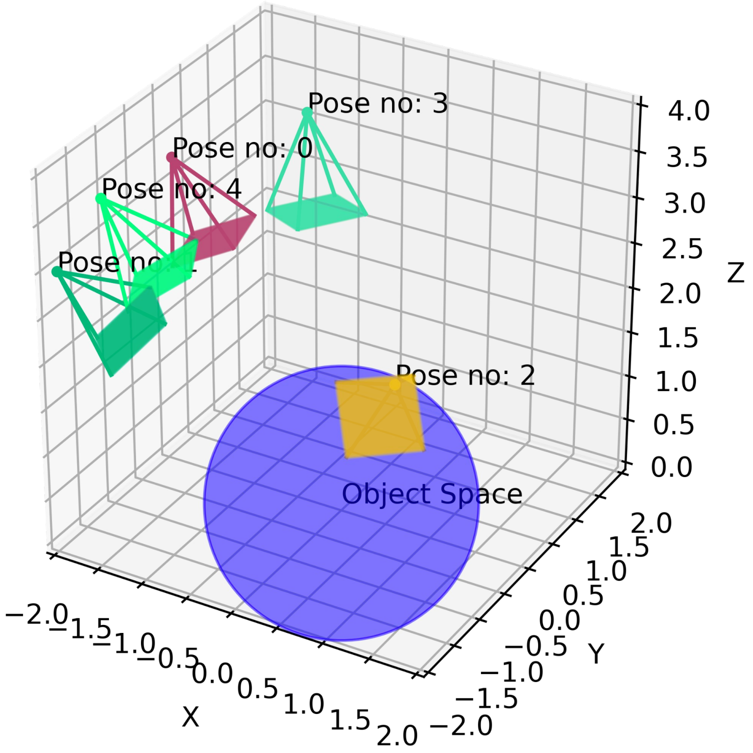
Parsing and loading of data.. Usually involves:

- RGB images
- Extrinsics and intrinsics camera parameters

Defines the underlying radiance field. Given 3D locations the fields predicts the Color and Density, etc.



Data Convention – Acquire/load Data



Also depth_file_path and mask_file_path are supported and can be provided here if needed for the method.

Camera extrinsics

For a transform matrix, the first 3 columns are the +X, +Y, and +Z defining the camera orientation, and the X, Y, Z values define the origin. The last row is to be compatible with homogeneous coordinates.

```
{
  // ...
  "frames": [
    {
      "file_path": "images/frame_00001.jpeg",
      "transform_matrix": [
        // [+X0 +Y0 +Z0 X]
        // [+X1 +Y1 +Z1 Y]
        // [+X2 +Y2 +Z2 Z]
        // [0.0 0.0 0.0 1]
        [1.0, 0.0, 0.0, 0.0],
        [0.0, 1.0, 0.0, 0.0],
        [0.0, 0.0, 1.0, 0.0],
        [0.0, 0.0, 0.0, 1.0]
      ]
      // Additional per-frame info
    }
  ]
}
```

Data Convention - Extrinsic Parameters

Camera intrinsics

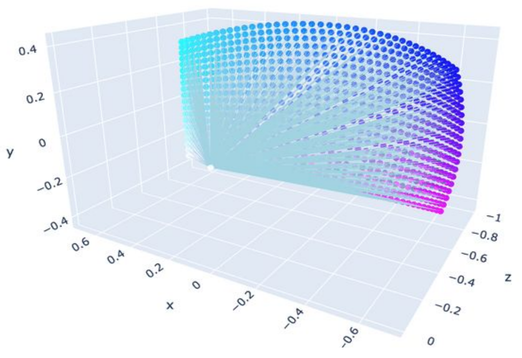
If all of the images share the same camera intrinsics, the values can be placed at the top of the file.

```
{
  "camera_model": "OPENCV_FISHEYE", // camera model type [OPENCV, OPENCV_FISHEYE]
  "fl_x": 1072.0, // focal length x
  "fl_y": 1068.0, // focal length y
  "cx": 1504.0, // principal point x
  "cy": 1000.0, // principal point y
  "w": 3008, // image width
  "h": 2000, // image height
  "k1": 0.0312, // first radial distorial parameter, used by [OPENCV, OPENCV_FISHEYE]
  "k2": 0.0051, // second radial distorial parameter, used by [OPENCV, OPENCV_FISHEYE]
  "k3": 0.0006, // third radial distorial parameter, used by [OPENCV_FISHEYE]
  "k4": 0.0001, // fourth radial distorial parameter, used by [OPENCV_FISHEYE]
  "p1": -6.47e-5, // first tangential distortion parameter, used by [OPENCV]
  "p2": -1.37e-7, // second tangential distortion parameter, used by [OPENCV]
  "frames": // ... per-frame intrinsics and extrinsics parameters
}
```

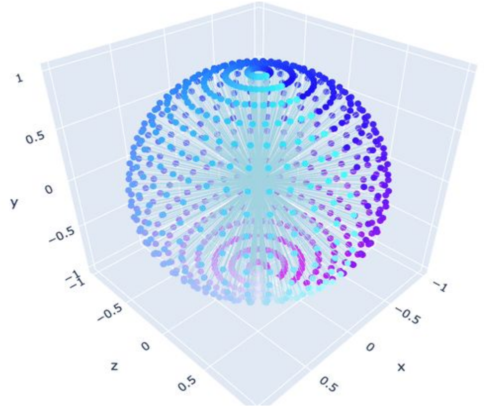
Per-frame intrinsics can also be defined in the `frames` field. If defined for a field (ie. `fl_x`), all images must have per-image intrinsics defined for that field. Per-frame `camera_model` is not supported.

```
{
  // ...
  "frames": [
    {
      "fl_x": 1234
    }
  ]
}
```

Perspective Camera Model



Spherical Camera Model



Data Convention - Intrinsic Parameters

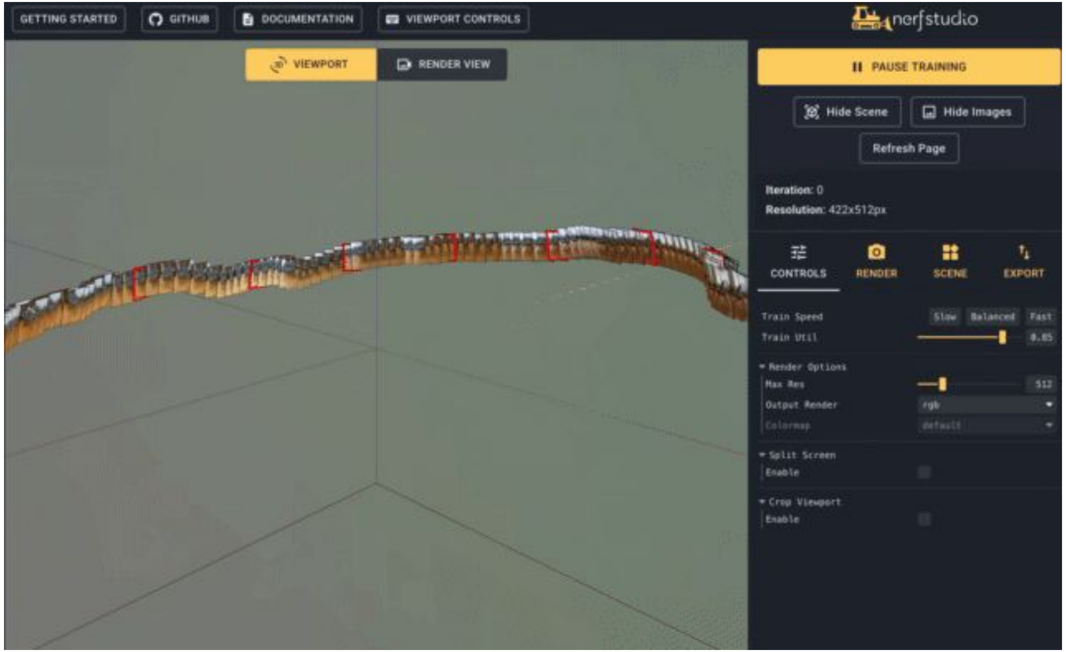
Data	Capture Device	Requirements	`ns-process-data` Speed
📷 Images	Any	COLMAP	🐢
📹 Video	Any	COLMAP	🐢
🌎 360 Data	Any	COLMAP	🐢
📱 Polycam	IOS with LiDAR	Polycam App	🐇
📱 KIRI Engine	IOS or Android	KIRI Engine App	🐇
📱 Record3D	IOS with LiDAR	Record3D app	🐇
🖥 Metashape	Any	Metashape	🐇
🖥 RealityCapture	Any	RealityCapture	🐇

Data Convention - How can we obtain these parameters

Let’s Train

```
# Activate conda environment
conda create --name nerfstudio -y python=3.8
conda activate nerfstudio
python -m pip install --upgrade pip

# Install torch
pip install torch==2.0.1+cu118 torchvision==0.15.2+cu118
--extra-index-url https://download.pytorch.org/whl/cu118

# Install cuda 11.8
conda install -c "nvidia/label/cuda-11.8.0" cuda-toolkit

# Install tiny-cuda-nn
pip install ninja
git+https://github.com/NVlabs/tiny-cuda-nn/#subdirectory=bindings/to
rch

# Install NeRF Studio
pip install git+https://github.com/nerfstudio-project/nerfstudio.git
```

Installing NeRF Studio



This is the chosen model*.

Similarly, own model with its field can be implemented and launch it the same way.

```
ns-train instant-ngp  \
  --viewer.websocket-port 7007 nerfstudio-data \
  --data $data \
  --downscale-factor 4
```

*https://github.com/nerfstudio-project/nerfstudio/blob/main/nerfstudio/configs/method_configs.py
https://github.com/nerfstudio-project/nerfstudio/blob/main/nerfstudio/models/

nerfstudio / nerfstudio / configs / method_configs.py

Code Blame 622 lines (591 loc) · 23 KB

```python
method_configs["instant-ngp"] = TrainerConfig(
    method_name="instant-ngp",
    steps_per_eval_batch=500,
    steps_per_save=2000,
    max_num_iterations=30000,
    mixed_precision=True,
    pipeline=DynamicBatchPipelineConfig(
        datamanager=VanillaDataManagerConfig(
            dataparser=NerfstudioDataParserConfig(),
            train_num_rays_per_batch=4096,
            eval_num_rays_per_batch=4096,
        ),
        model=InstantNGPModelConfig(eval_num_rays_per_chunk=8192),
    ),
    optimizers={
        "fields": {
            "optimizer": AdamOptimizerConfig(lr=1e-2, eps=1e-15),
            "scheduler": ExponentialDecaySchedulerConfig(lr_final=0.0001, max_steps=200000),
        }
    },
    viewer=ViewerConfig(num_rays_per_chunk=1 << 12),
    vis="viewer",
)
```

nerfstudio / nerfstudio / models / instant_ngp.py

Code Blame 273 lines (233 loc) · 10 KB

```python
class NGPModel(Model):
    """Instant NGP model

    Args:
        config: instant NGP configuration to instantiate model
    """

    config: InstantNGPModelConfig
    field: NerfactoField

    def __init__(self, config: InstantNGPModelConfig, **kwargs) -> None:
        super().__init__(config=config, **kwargs)

    def populate_modules(self):
        """Set the fields and modules."""
        super().populate_modules()

        if self.config.disable_scene_contraction:
            scene_contraction = None
        else:
            scene_contraction = SceneContraction(order=float("inf"))

        self.field = NerfactoField(
            aabb=self.scene_box.aabb,
            appearance_embedding_dim=0 if self.config.use_appearance_embedding else 32,
            num_images=self.num_train_data,
            log2_hashmap_size=self.config.log2_hashmap_size,
            max_res=self.config.max_res,
            spatial_distortion=scene_contraction,
        )
```

Extend With Own Model


```
ns-render camera-path \
  --load-config $config_filename \
  --camera-path-filename $camera_path_filename \
  --output-path renders/output.mp4
```

The file containing all the extrinsics and intrinsics parameters for the video to be rendered

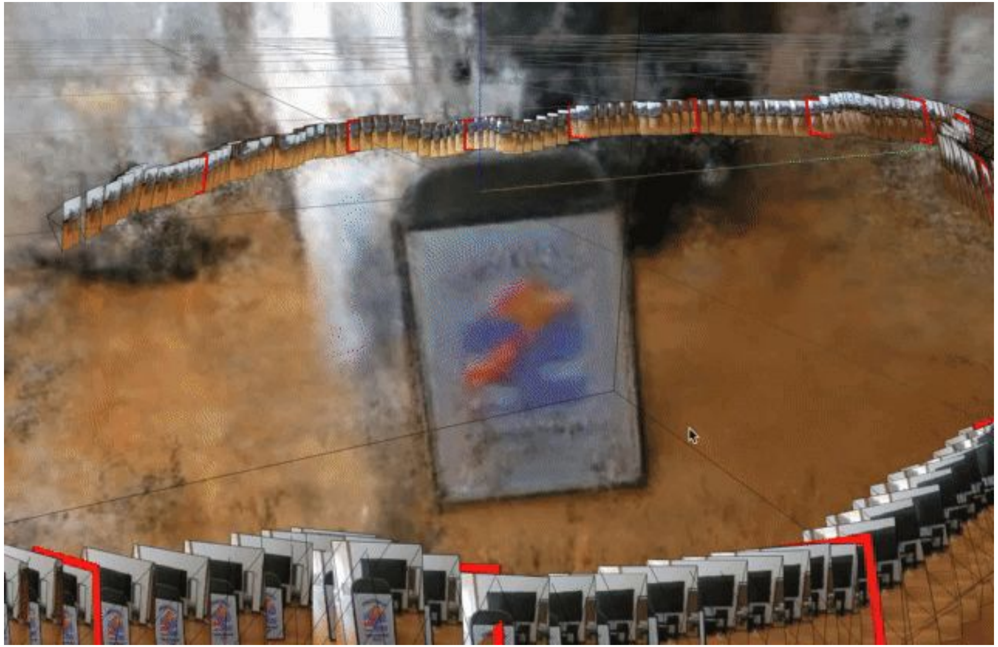
Rendering a Trajectory From a Trained Model with NeRF Studio


```
ns-export poisson \
  --load-config $config_filename \
  --output-dir $base_dir
```

3D Mesh Extraction From NerfStudio

Exercises

Pose Refining Neural Radiance Fields



```
ns-train instant-ngp  \
  --viewer.websocket-port 7007 nerfstudio-data \
  --data $data_noisy \
  --downscale-factor 4
```

Augmented the 3D rotation with noise of less than 3 degrees.

Exercise 1 - Train NeRF and Render

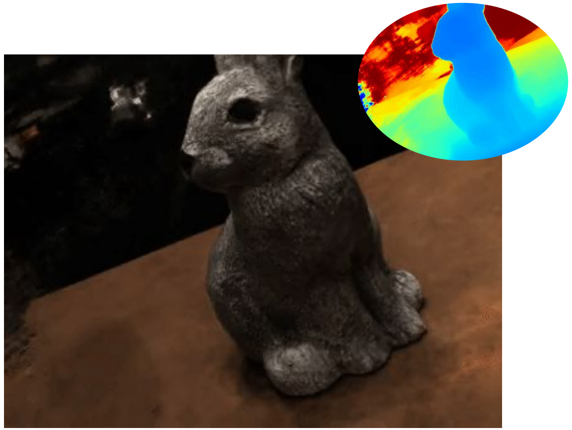
Train a NeRF

1.) Download the DTU dataset.

Link: https://roboimagedata.compute.dtu.dk/?page_id=36

2.) Train a NeRF with an instant NGP backbone on the images.

3.) Render novel trajectory.



Augment Camera Poses with Noise

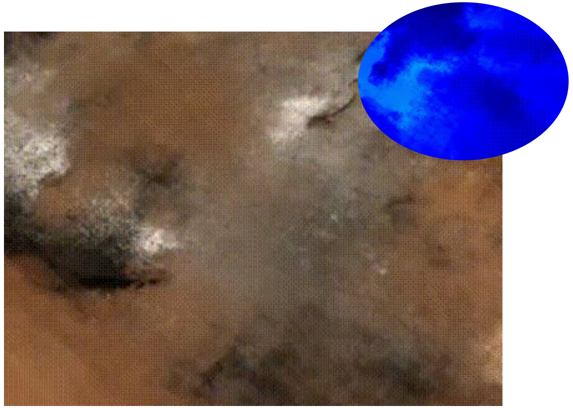
1.) Add noise to the camera rotations.

2.) Train a NeRF with an instant NGP backbone.

3.) Render novel trajectory.

4.) Keep increasing the noise and repeat until the reconstruction fails.

(In real life poses are often not super accurate, especially when coming from a handheld device)



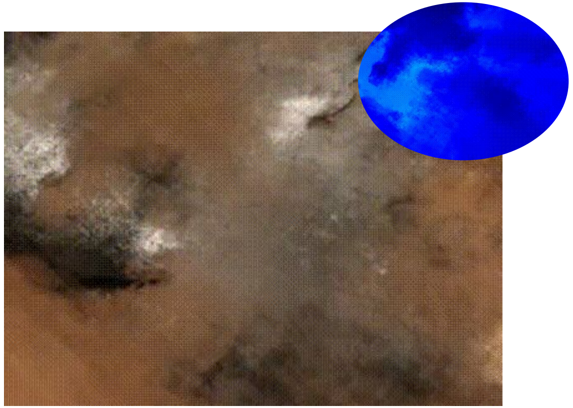
Exercise 1 – Train NeRF and Render

Dealing With Noise
1.) Add the camera poses as additional optimization target and optimize over reconstruction and poses.

- Try out different representations for the 3D rotation (quaternions, Zhou et al. CVPR 2019).
- Consider slowly increasing the expressiveness of the employed Nerf (BARF).

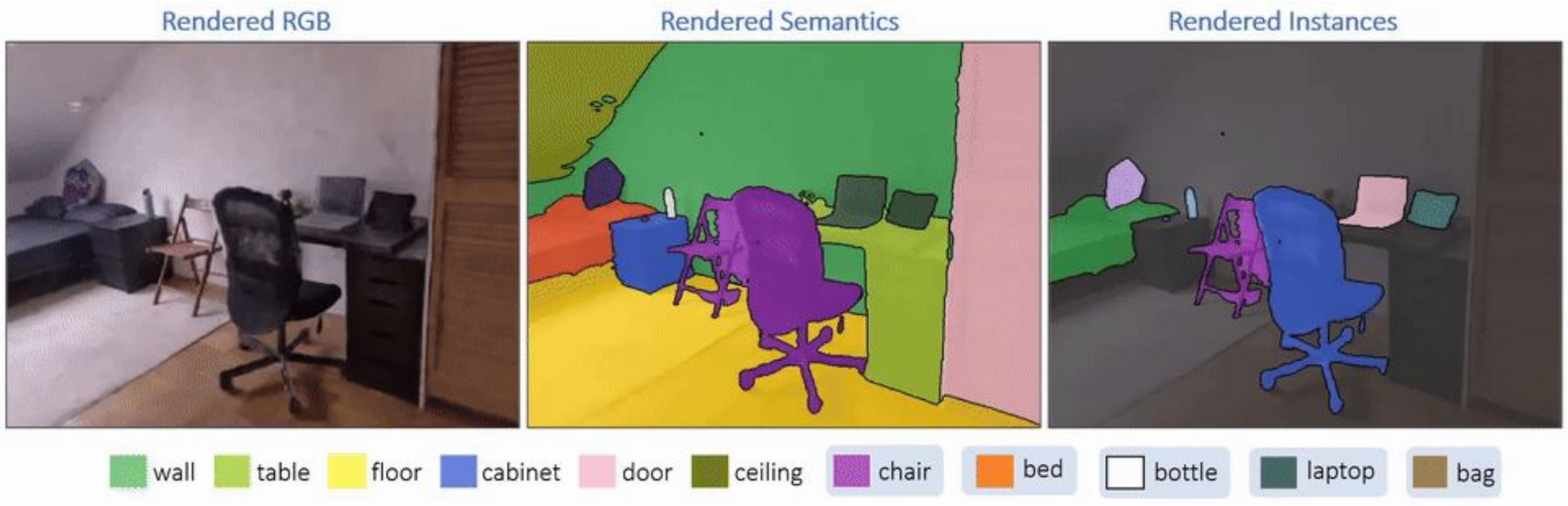
Open-End
Improve the noise handling to enhance accuracy and robustness:

- Bundle adjustment from BARF
- 2D correspondences and pseudo depth from SPARF
- Camera preconditioning from CamP
- Projected ray distance from SCNeRF
- …?



OpenSet 3D semantic segmentation

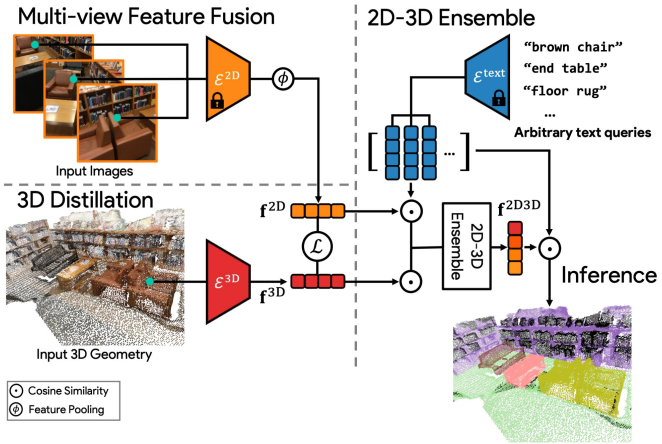
Existing methods for 3D scene understanding assume pre-defined set of object types ("closed-world" assumption).



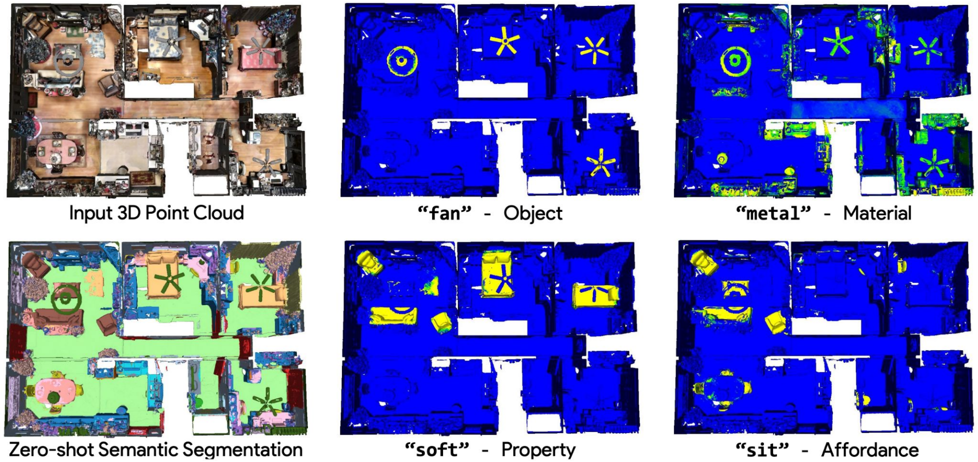
Panoptic Lifting for 3D Scene Understanding with Neural Fields
(CVPR 2023 Highlight)

The real world is, however, much more complex. Further, the data is usually not representing each class equally, leading to a significant drop in accuracy.

Hence, can we segment anything in our NeRF/3D mesh?



Model



Results

OpenScene: 3D Scene Understanding with Open Vocabularies

composite_0

Prompt - ("floor", "sign")

```
pip install git+https://github.com/kerrj/lerf

ns-train lerf-lite  \
  --viewer.websocket-port 7007 nerfstudio-data --data $data \
  --downscale-factor 4
```

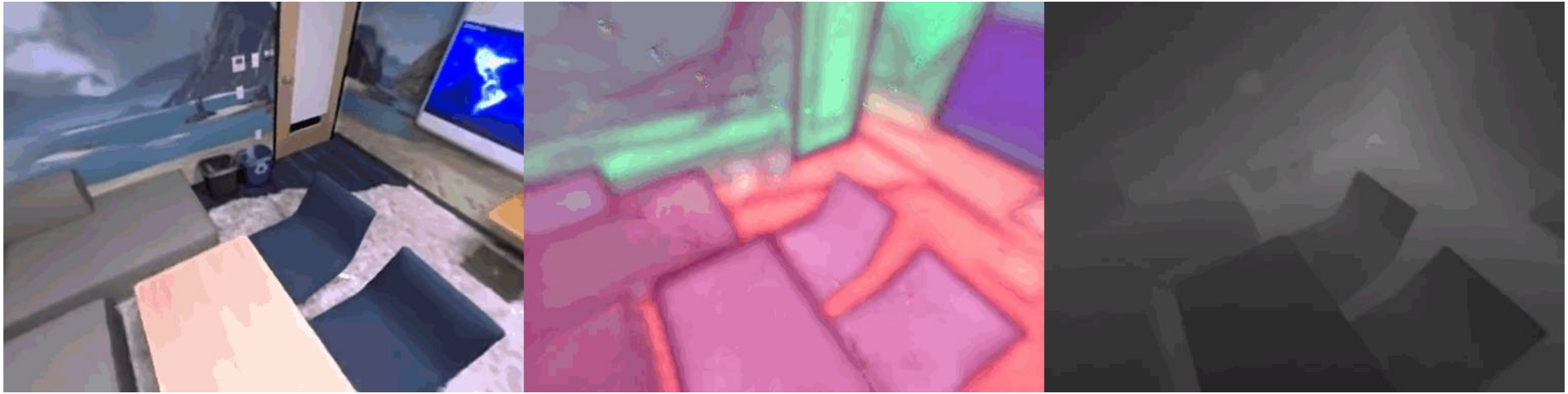
Segmenting with LeRF

Train a NeRF and Render

1.) Download the Replica dataset:

- https://github.com/cvg/nice-slam/blob/master/scripts/download_replica.sh
- https://github.com/facebookresearch/Replica-Dataset

2.) Train a NeRF with an instant NGP backbone on the images
3.) Render a novel trajectory.



Label the data with CLIP features

1.) Employ LSeg / OpenSeg to label each training image with CLIP-like pixel level features.
2.) Train again the NeRF but with additional branch which learns to render the LSeg features*.
3.) Render novel trajectory with Lseg features.

*https://github.com/nerfstudio-project/nerfstudio/blob/main/nerfstudio/fields/vanilla_nerf_field.py

Exercise 2 - Label The Input Image and Train Again

Segment Different Objects and Properties
1.) Render a view with its LSeg features and compute the correlation between LSeg and the CLIP encoding from a text prompt.
2.) Try out different objects as well as material properties such as glass.

Open End Question
How could the segmentation be improved:

- What about different object sizes?
- What about disagreement between different training views?
- …?

Exercise 2 - Run and Improve

Thanks / Questions?