

View Synthesis

Implementing NeRF in PyTorch

Task Description

"View synthesis" is a task which generating images of a 3D scene from a specific point of view.



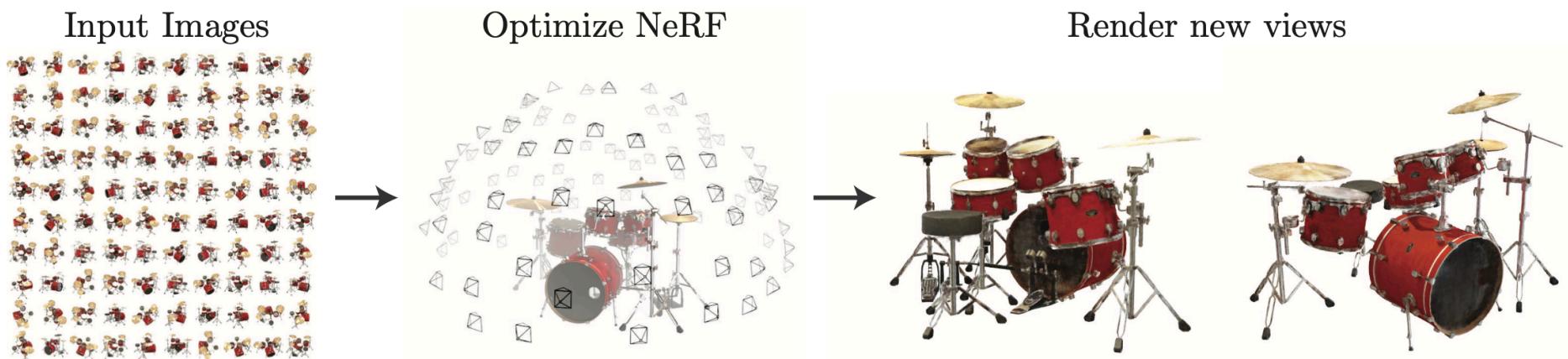
Solution Description (Cont.)

NeRF (Neural Radiance Field)

It can solve "View synthesis" by representing 3D scene using a neural network.

We can render the image \hat{Y} from the view point X by using the function F :

$$\hat{Y} = F(X)$$



Solution Description (Cont.)

Volume Rendering

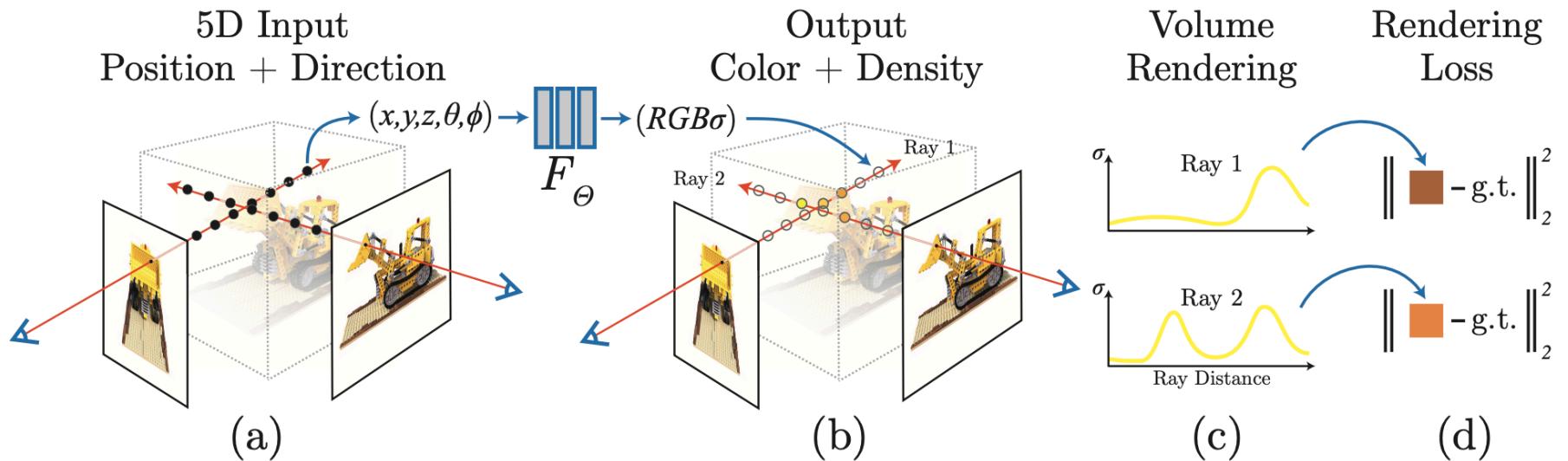
We render the 2D image \hat{Y} with height H and width W from the view point X using the 3D scene representation f :

1. The origin o and the direction $d(r = 1)$ of the view point X are given.
2. From the origin, we emit $H \times W$ rays through the image plane.
3. For each ray, sample S positions $p(r, i)$ along the direction $d(r)$.
4. Blend the color of the positions to approximate the color of the pixel $C(r)$.
5. Gather all the colors $C(r)$ to form the image \hat{Y} .

$$\begin{aligned} p(r, i) &= o + t_i d(r), \\ c_i, \sigma_i &= f(p(r, i), d(r)), \\ l(i) &= \exp(-\sigma_i(t_{i+1} - t_i)), \\ \hat{C}(r) &= \sum_{i=1}^S c_i (1 - l(i)) (\prod_{j=1}^{i-1} l(j)), \\ \hat{Y} &= \{\hat{C}(r) | r = 1, 2, \dots, H \times W\} \end{aligned}$$

Solution Description (Cont.)

Volume Rendering (Illustration)



Solution Description (Cont.)

Volumetric Representation Estimation

To estimate the 3D scene representation f , we can train a fully-connected neural network.

Since the volume rendering process is differentiable, we can optimize the neural network using Gradient Descent techniques.

The loss function is defined as the Mean Squared Error between the rendered image \hat{Y} and the ground truth image Y :

$$\mathcal{L} = \frac{\sum_{r=1}^{H \times W} (\hat{Y}_r - Y_r)^2}{H \times W}$$

Solution Conclusion

NeRF is composed of three parts:

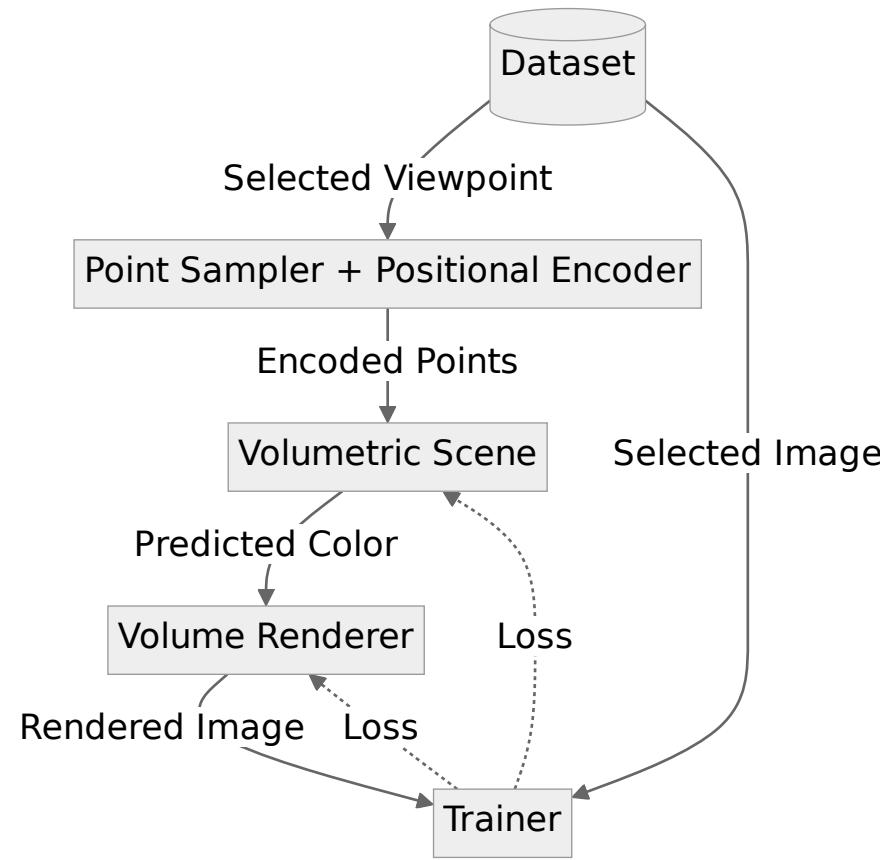
1. f : Volumetric representation estimation using a fully-connected neural network.
2. C : Volume rendering using the implicit scene representation.
3. \mathcal{L} : Loss function for training the volume rendering function C .

Modules

We have implemented the following modules using PyTorch v2.3:

1. **Point Sampler**: To sample points from batches of rays
2. **Positional Encoder**: To apply Fourier feature encoding for the input points
3. **Volumetric Scene**: To predict the color and density of the input points
4. **Volume Renderer**: To render the image from the sampled points and colors
5. **Trainer**: To optimize the renderer with the given images and viewpoints

Pipeline



Module Details

Positional Encoder

To explore the high-frequency features in the input points, we apply Fourier feature encoding.

Each coordinate value is encoded as follows:

$$\begin{aligned} & \text{Encode}_L(p) \\ &= \{\sin(2^0\pi p), \cos(2^0\pi p), \dots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)\} \\ &= \{\sin(2^0\pi p), \sin(\frac{\pi}{2} + 2^0\pi p), \dots, \sin(2^{L-1}\pi p), \sin(\frac{\pi}{2} + 2^{L-1}\pi p)\} \\ & \quad \text{where } p \in \mathbb{R}, L \in \mathbb{N}, \text{Encode}_L(p) \in \mathbb{R}^{2L} \end{aligned}$$

The raw and encoded coordinate values will be concatenated to form the network input.

The encoded dimensions are calculated as follows:

$$\text{Encoded Dimension} = \text{Input Dimension} \times (2L + 1)$$

Module Details

Volumetric Scene

The fully-connected neural network structure is as follows, and the skip connection is applied to 5th hidden layer:

Constants	Description	Value
I	Input Dimension	$3 (\text{Position}) + 3 (\text{Direction}) = 6$
E	Encoded Dimension	$I \times (2 \times 10 + 1) = 121$
H	Hidden Dimension	256
O	Output Dimension	$3 (\text{Color}) + 1 (\text{Density}) = 4$

Layer	Input Dim.	Output Dim.	Activation
0	E	H	ReLU
1	H	H	ReLU
2	H	H	ReLU
3	H	H	ReLU
4	H	H	ReLU
5	$H + E$	H	ReLU
6	H	H	ReLU
7	H	H	ReLU
8	H	O	Sigmoid (Color), ReLU (Density)

Experiment Details

Hyper-parameters

Constants	Description	Value
N	Number of Epochs	50000
S	Sample points per ray	100
B_r	Rays per batch	200
L	Encoding Factor	10
$\ Test\ $	Number of Testing Images	21
$\ Train\ $	Number of Training Images	85

References

1. View synthesis. (n.d.). In Wikipedia. Retrieved from https://en.wikipedia.org/wiki/View_synthesis
2. Neural radiance field. (n.d.). In Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Neural_radiance_field
3. Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., & Ng, R. (2020). NeRF: Neural radiance fields for image synthesis. arXiv preprint arXiv:2003.08934. Retrieved from <https://arxiv.org/pdf/2003.08934>
4. Tancik, M., Srinivasan, P. P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J. T., & Ng, R. (2020). Fourier features let networks learn high frequency functions in low dimensional domains. NeurIPS. Retrieved from <https://arxiv.org/pdf/2006.10739>

Module - Point Sampler

```
In [3]: from torch import Tensor
from torch.nn import Module
from torch.types import Device

class PointSampler(Module):
    def __init__(self,
                 focal: float,
                 height: int,
                 width: int,
                 points_per_ray: int,
                 device: Device,
                 ):
        """
        [4, 4] => [height, width, points_per_ray, 3 + 3 + 1]
        """

        super(PointSampler, self).__init__()

        import torch

        focal = float(focal)
        height = int(height)
        width = int(width)
        points_per_ray = max(int(points_per_ray), 1)

        self.directions = torch.stack(
            torch.meshgrid(
                (torch.arange(float(width), device=device) - width / 2.0) / focal,
                (-torch.arange(float(height), device=device) + height / 2.0) / focal,
                torch.tensor([-1.0, device=device]),
                indexing="xy",
            ),
            dim=-1,
        )
        self.points_per_ray = points_per_ray

    def forward(
        self,
        viewpoint: Tensor,
        distance_range: tuple[float, float],
        is_random: bool,
    ) -> Tensor:
        import torch

        viewpoint = torch.as_tensor(viewpoint)[:3]
        device = viewpoint.device
        distance_range = tuple(map(float, distance_range))
        is_random = bool(is_random)
        distance_max = max(distance_range)
        distance_min = min(distance_range)
        interval = (distance_max - distance_min) / self.points_per_ray

        directions = (self.directions * viewpoint[:, :3]).sum(dim=-1).unsqueeze(-2)
        origins = viewpoint[:, -1].expand_as(directions)
        distances = (
            torch.linspace(
```

```
        distance_min,
        distance_max,
        self.points_per_ray,
        device=device,
    )
    .repeat(
        (*origins.shape[:-2], 1),
    )
    .unsqueeze(-1)
)
if is_random:
    distances += (
        torch.rand(*origins.shape[:-2], self.points_per_ray, 1, device=device)
        * interval
    )
positions = origins + directions * distances
directions = directions.expand_as(positions)
return torch.cat([positions, directions, distances], dim=-1)
```

Module - Positional Encoder

```
In [4]: from torch import Tensor
from torch.nn import Module
from torch.types import Device

class PositionalEncoder(Module):
    def __init__(self, encoding_factor: int, device: Device):
        """
        [..., input_dimension] => [..., input_dimension * (2 * encoding_factor + 1)]
        """

        import torch

        super(PositionalEncoder, self).__init__()

        encoding_factor = max(int(encoding_factor), 0)

        freq_lvls = torch.arange(encoding_factor, device=device)
        self.freq = ((2**freq_lvls) * torch.pi).repeat_interleave(2).unsqueeze_(-1)
        sine_offsets = torch.tensor([0.0, torch.pi / 2], device=device)
        self.offsets = sine_offsets.repeat(encoding_factor).unsqueeze_(-1)

    def forward(self, inputs: Tensor) -> Tensor:
        import torch

        inputs = torch.as_tensor(inputs).unsqueeze(-2)

        features = (self.freq * inputs + self.offsets).sin()
        features = torch.cat([inputs, features], dim=-2)
        features = features.reshape(*inputs.shape[:-2], -1)
        return features

    def get_last_dimension(self, input_dimension: int) -> int:
        return int(input_dimension) * (self.freq.shape[0] + 1)
```

Module - Volumetric Scene

In [5]:

```
from torch.nn import Module

class VolumetricScene(Module):
    def __init__(self, encoding_factor: int, device: Device):
        """
        [..., 3 + 3] => [..., 4]
        """

        super(VolumetricScene, self).__init__()

        from torch.nn import Linear, ModuleList, ReLU, Sigmoid

        I = 3 + 3
        O = 4
        H = 256

        encoding_factor = max(int(encoding_factor), 0)
        self.encode = PositionalEncoder(encoding_factor, device=device)
        I = self.encode.get_last_dimension(I)

        self.layers = ModuleList(
            [
                Linear(I, H, device=device),
                ReLU(),
                Linear(H, H, device=device),
                ReLU(),
                Linear(H, H, device=device),
                ReLU(),
                Linear(H, H, device=device),
                ReLU(),
                Linear(H, H, device=device),
                ReLU(),
                Linear(H + I, H, device=device),
                ReLU(),
                Linear(H, H, device=device),
                ReLU(),
                Linear(H, H, device=device),
                ReLU(),
                Linear(H, O, device=device),
            ]
        )
        self.skip_indexs = {
            10,
        }
        self.output_rgb_activation = Sigmoid()
        self.output_alpha_activation = ReLU()

    def forward(self, inputs: Tensor) -> Tensor:
        import torch

        inputs = self.encode(inputs)
        outputs = inputs

        for index, layer in enumerate(self.layers):
            if index in self.skip_indexs:
                outputs = torch.cat([outputs, inputs], dim=-1)
            outputs = layer(outputs)
```

```
outputs = torch.cat(
    [
        self.output_rgb_activation(outputs[..., :3]),
        self.output_alpha_activation(outputs[..., 3:]),
    ],
    dim=-1,
)
return outputs
```

Module - Volume Renderer

```
In [6]: from torch import Tensor
from torch.nn import Module
from torch.types import Device

class VolumeRenderer(Module):
    def __init__(self,
                 focal: float,
                 height: int,
                 width: int,
                 points_per_ray: int,
                 encoding_factor: int,
                 device: Device,
                 ):
        """
        [4, 4] => [height, width, 3]
        """

        super(VolumeRenderer, self).__init__()

        self.sample = PointSampler(
            focal=focal,
            height=height,
            width=width,
            points_per_ray=points_per_ray,
            device=device,
        )
        self.predict = VolumetricScene(
            encoding_factor=encoding_factor,
            device=device,
        )

    def forward(
        self,
        viewpoint: Tensor,
        rays_per_batch: int,
        distance_range: tuple[float, float],
        is_random: bool,
    ) -> Tensor:
        import torch

        rays_per_batch = max(int(rays_per_batch), 1)

        points_per_batch = rays_per_batch * self.sample.points_per_ray
        points: Tensor = self.sample(viewpoint, distance_range, is_random)
        points, distances = points[..., :-1], points[..., -1]
        colors = torch.cat([
            self.predict(batch)
            for batch in TensorDataLoader(
                data=points.reshape(-1, points.shape[-1]),
                batch_size=points_per_batch,
            )
        ],
        dim=0,
    )
        colors = colors.reshape(*points.shape[:-1], -1)
```

```
rgb = colors[..., :3]
alpha = colors[..., 3]

intervals = torch.cat(
    [
        distances[..., 1:] - distances[..., :-1],
        torch.tensor([1e9], device=distances.device).expand_as(
            distances[..., :-1]
        ),
    ],
    dim=-1,
)
translucency = (-alpha * intervals).exp().unsqueeze(-1)
transmittance = (1.0 - translucency) * torch.cumprod(translucency + 1e-9, dim=-2)
rgb_planar = (rgb * transmittance).sum(dim=-2)

return rgb_planar

class TensorDataLoader:
    def __init__(self, data: Tensor, batch_size: int):
        from torch import as_tensor

        self.data = as_tensor(data)
        self.batch_size = max(int(batch_size), 1)

    def __iter__(self):
        return (
            self.data[i : i + self.batch_size]
            for i in range(0, self.data.shape[0], self.batch_size)
        )

    def __len__(self):
        return -(len(self.data) // self.batch_size)
```

Module - Trainer

```
In [7]: from dataclasses import dataclass
from torch import Tensor
from torch.types import Device
```

```
@dataclass
class Trainer:
    test_dataset: "ViewSynthesisDataset"
    train_dataset: "ViewSynthesisDataset"

    @staticmethod
    def from_dataset(
        dataset: "ViewSynthesisDataset",
        train_ratio: float,
        device: Device,
    ):
        train_data_count = max(int(round(dataset.count * train_ratio)), 1)
        test_data_count = dataset.count - train_data_count
        data_split_index = -test_data_count
        focal = dataset.focal
        height = dataset.height
        images = dataset.images
        viewpoints = dataset.viewpoints
        width = dataset.width

        return Trainer(
            test_dataset=ViewSynthesisDataset(
                count=test_data_count,
                focal=focal,
                height=height,
                images=images[data_split_index:],
                viewpoints=viewpoints[data_split_index:],
                width=width,
            ).set_device(device),
            train_dataset=ViewSynthesisDataset(
                count=train_data_count,
                focal=focal,
                height=height,
                images=images[:data_split_index],
                viewpoints=viewpoints[:data_split_index],
                width=width,
            ).set_device(device),
        )

    def train(
        self,
        render: VolumeRenderer,
        epochs: int,
        rays_per_batch: int,
        distance_range: tuple[float, float],
        show_progress: bool,
    ) -> None:
        import torch
        from torch.nn import MSELoss
        from torch.optim import Adam
        from tqdm import tqdm

        EPOCHS_PER_DEMO = 250
```

```

criterion = MSELoss()
optimizer = Adam(render.parameters(), lr=5e-4)
progress = tqdm(
    disable=not show_progress,
    desc="Fitting the renderer to {self.train_dataset.count}x images and viewpoints",
    colour="green",
    dynamic_ncols=True,
    total=epochs,
)

with progress:
    for epoch in range(epochs):
        true_image, viewpoint = self.train_dataset.get_image_and_viewpoint()
        optimizer.zero_grad()
        rendered_image: Tensor = render(
            viewpoint=viewpoint,
            rays_per_batch=rays_per_batch,
            distance_range=distance_range,
            is_random=True,
        )
        loss = criterion(rendered_image, true_image)
        loss.backward()
        optimizer.step()

        if show_progress and epoch % EPOCHS_PER_DEMO == 0:
            with torch.no_grad():
                true_image_demo, viewpoint_demo = (
                    self.train_dataset.get_image_and_viewpoint(0)
                )
                rendered_image_demo = render(
                    viewpoint=viewpoint_demo,
                    rays_per_batch=rays_per_batch,
                    distance_range=distance_range,
                    is_random=False,
                )
                Trainer.display(
                    torch.cat([true_image_demo, rendered_image_demo], dim=1)
                )

        progress.update()

def test(
    self,
    render: VolumeRenderer,
    rays_per_batch: int,
    distance_range: tuple[float, float],
) -> None:
    from math import log10
    import torch
    from torch.nn.functional import mse_loss

    with torch.no_grad():
        for index in range(self.test_dataset.count):
            true_image, viewpoint = self.test_dataset.get_image_and_viewpoint(index)
            rendered_image = render(
                viewpoint=viewpoint,
                rays_per_batch=rays_per_batch,
                distance_range=distance_range,
                is_random=False,
            )
            quality_mse = mse_loss(rendered_image, true_image).item()
            quality_psnr = -10 * log10(quality_mse)
            display(
                dict(
                    test_index=index,

```

```

        quality=dict(mse=quality_mse, psnr=quality_psnr),
    )
)

Trainer.display(torch.cat([true_image, rendered_image], dim=1))

@staticmethod
def display(image: Tensor):
    from IPython.display import display
    from PIL import Image
    from torch import uint8

    return display(
        Image.fromarray((image * 255).round().type(uint8).numpy(force=True))
    )

@dataclass
class ViewSynthesisDataset:
    count: int
    focal: float
    height: int
    images: Tensor
    viewpoints: Tensor
    width: int

    def __post_init__(self) -> None:
        if self.images.shape[0] != self.viewpoints.shape[0]:
            raise ValueError("The number of images and viewpoints must be the same")

    @staticmethod
    def from_numpy(url: str) -> "ViewSynthesisDataset":
        from httpx import get
        from io import BytesIO
        from numpy import load
        import torch

        try:
            file = open(url, "rb")
        except:
            file = BytesIO(
                get(url, follow_redirects=True, timeout=60).raise_for_status().content
            )

        with file as file_entered:
            arrays = load(file_entered)
            focal = float(arrays["focal"])
            images = torch.as_tensor(arrays["images"])
            viewpoints = torch.as_tensor(arrays["poses"])

        return ViewSynthesisDataset(
            count=images.shape[0],
            focal=focal,
            height=images.shape[1],
            images=images,
            viewpoints=viewpoints,
            width=images.shape[2],
        )

    def get_image_and_viewpoint(self, index: int | None = None) -> tuple[Tensor, Tensor]:
        from random import randint

        if index is not None:
            index = int(index)
        else:
            index = randint(0, self.count - 1)

```

```
    return self.images[index], self.viewpoints[index]

def set_device(self, device: Device) -> "ViewSynthesisDataset":
    self.images = self.images.to(device)
    self.viewpoints = self.viewpoints.to(device)
    return self

def __repr__(self) -> str:
    repr = f"{self.__class__.__name__}("
    for name, value in self.__dict__.items():
        if isinstance(value, Tensor):
            value = f"Tensor(shape={tuple(value.shape)}, dtype={value.dtype})"
        elif type(value) is float:
            value = f"{value:.7f}"
        repr += f"\n  {name}={value},"
    repr += "\n"
    return repr
```

Experiment

In [8]:

```
def main() -> None:
    import torch

    EPOCHS = 50000
    RAYS_PER_BATCH = 200
    POINTS_PER_RAY = 100
    ENCODING_FACTOR = 10
    DISTANCE_RANGE = (2.0, 6.0)

    # The original dataset is retrieved from
    # http://cseweb.ucsd.edu/~viscomp/projects/LF/papers/ECCV20/nerf/tiny_nerf_data.npz
    # The alternative dataset can be retrieved from
    # https://raw.githubusercontent.com/AsherJingkongChen/nerf/main/tiny_nerf_data.npz
    dataset = ViewSynthesisDataset.from_numpy("tiny_nerf_data.npz")
    device = (
        "cuda:1"
        if torch.cuda.is_available()
        else ("mps" if torch.backends.mps.is_available() else "cpu")
    )
    render = VolumeRenderer(
        focal=dataset.focal,
        height=dataset.height,
        width=dataset.width,
        points_per_ray=POINTS_PER_RAY,
        encoding_factor=ENCODING_FACTOR,
        device=device,
    )
    trainer = Trainer.from_dataset(dataset=dataset, train_ratio=0.8, device=device)

    display(dict(trainer=trainer, render=render))

    trainer.train(
        render=render,
        epochs=EPOCHS,
        rays_per_batch=RAYS_PER_BATCH,
        distance_range=DISTANCE_RANGE,
        show_progress=True,
    )
    trainer.test(
        render=render,
        rays_per_batch=RAYS_PER_BATCH,
        distance_range=DISTANCE_RANGE,
    )
    torch.save(render.state_dict(), "VolumeRenderer.pth")
```

In [9]:

```
if __name__ == "__main__":
    main()
```

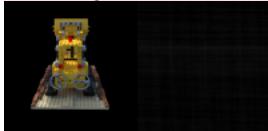
```

{'trainer': Trainer(test_dataset=ViewSynthesisDataset(
    count=21,
    focal=138.8888789,
    height=100,
    images=Tensor(shape=(21, 100, 100, 3), dtype=torch.float32),
    viewpoints=Tensor(shape=(21, 4, 4), dtype=torch.float32),
    width=100,
), train_dataset=ViewSynthesisDataset(
    count=85,
    focal=138.8888789,
    height=100,
    images=Tensor(shape=(85, 100, 100, 3), dtype=torch.float32),
    viewpoints=Tensor(shape=(85, 4, 4), dtype=torch.float32),
    width=100,
)),
'render': VolumeRenderer(
    (sample): PointSampler()
    (predict): VolumetricScene(
        (encode): PositionalEncoder()
        (layers): ModuleList(
            (0): Linear(in_features=126, out_features=256, bias=True)
            (1): ReLU()
            (2): Linear(in_features=256, out_features=256, bias=True)
            (3): ReLU()
            (4): Linear(in_features=256, out_features=256, bias=True)
            (5): ReLU()
            (6): Linear(in_features=256, out_features=256, bias=True)
            (7): ReLU()
            (8): Linear(in_features=256, out_features=256, bias=True)
            (9): ReLU()
            (10): Linear(in_features=382, out_features=256, bias=True)
            (11): ReLU()
            (12): Linear(in_features=256, out_features=256, bias=True)
            (13): ReLU()
            (14): Linear(in_features=256, out_features=256, bias=True)
            (15): ReLU()
            (16): Linear(in_features=256, out_features=4, bias=True)
        )
        (output_rgb_activation): Sigmoid()
        (output_alpha_activation): ReLU()
    )
)
}

```

Fitting the renderer to 85x images and viewpoints: 0% |

| 0/50000 [00:00<?, ?it/s]



Fitting the renderer to 85x images and viewpoints: 0% |

| 250/50000 [02:54<10:02:46, 1.38it/s]



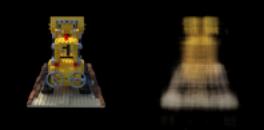
Fitting the renderer to 85x images and viewpoints: 1% |

| 500/50000 [05:56<9:57:11, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 2%|

| 750/50000 [08:57<9:54:55, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 2%|

| 1000/50000 [11:58<9:52:33, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 2%|

| 1250/50000 [15:00<9:48:10, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 3%|

| 1500/50000 [18:01<9:42:37, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 4%|

| 1750/50000 [21:02<9:36:56, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 4%|

| 2000/50000 [24:03<9:39:01, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 4%|

| 2250/50000 [27:03<9:35:40, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 5%|

| 2500/50000 [30:04<9:30:26, 1.39it/s]



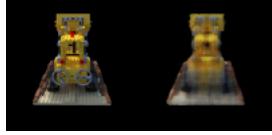
Fitting the renderer to 85x images and viewpoints: 6%|

| 2750/50000 [33:05<9:32:41, 1.38it/s]



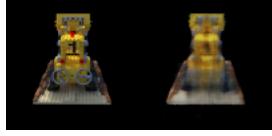
Fitting the renderer to 85x images and viewpoints: 6%|

| 3000/50000 [36:08<9:31:36, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 6%|

| 3250/50000 [39:10<9:28:07, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 7%|

| 3500/50000 [42:11<9:23:17, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 8%|

| 3750/50000 [45:13<9:20:12, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 8%|

| 4000/50000 [48:15<9:17:19, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 8%|

| 4250/50000 [51:16<9:14:59, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 9%|

| 4500/50000 [54:18<9:10:25, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 10%|

| 4750/50000 [57:19<9:07:43, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 10%|██████

| 5000/50000 [1:00:21<9:03:11, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 10%|██████

| 5250/50000 [1:03:22<9:02:01, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 11%|██████

| 5500/50000 [1:06:24<8:56:53, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 12%|██████

| 5750/50000 [1:09:25<8:54:14, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 12%|██████

| 6000/50000 [1:12:26<8:49:12, 1.39it/s]



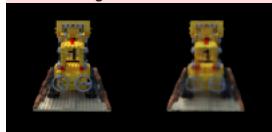
Fitting the renderer to 85x images and viewpoints: 12%|██████

| 6250/50000 [1:15:28<8:47:32, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 13%|██████

| 6500/50000 [1:18:29<8:46:30, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 14%|██████

| 6750/50000 [1:21:30<8:44:31, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 14% [██████]

| 7000/50000 [1:24:32<8:42:02, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 14% [██████]

| 7250/50000 [1:27:33<8:35:40, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 15% [██████]

| 7500/50000 [1:30:35<8:33:12, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 16% [██████]

| 7750/50000 [1:33:36<8:29:39, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 16% [██████]

| 8000/50000 [1:36:38<8:25:21, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 16% [██████]

| 8250/50000 [1:39:39<8:25:18, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 17% [██████]

| 8500/50000 [1:42:40<8:20:34, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 18% [██████]

| 8750/50000 [1:45:41<8:15:52, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 18% | |

| 9000/50000 [1:48:43<8:11:52, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 18% | |

| 9250/50000 [1:51:44<8:11:51, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 19% | |

| 9500/50000 [1:54:45<8:08:56, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 20% | |

| 9750/50000 [1:57:47<8:08:02, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 20% | |

| 10000/50000 [2:00:48<8:02:10, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 20% | |

| 10250/50000 [2:03:49<7:59:49, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 21% | |

| 10500/50000 [2:06:50<7:57:55, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 22% | |

| 10750/50000 [2:09:52<7:54:31, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 22% | | 11000/50000 [2:12:53<7:52:32, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 22% | | 11250/50000 [2:15:54<7:48:32, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 23% | | 11500/50000 [2:18:56<7:45:07, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 24% | | 11750/50000 [2:21:57<7:43:51, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 24% | | 12000/50000 [2:24:58<7:38:06, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 24% | | 12250/50000 [2:27:59<7:33:00, 1.39it/s]



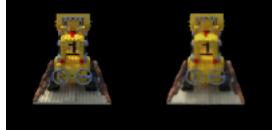
Fitting the renderer to 85x images and viewpoints: 25% | | 12500/50000 [2:31:00<7:33:48, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 26% | | 12750/50000 [2:34:02<7:30:25, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 26% | | 13000/50000 [2:37:03<7:28:48, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 26% | | 13250/50000 [2:40:04<7:22:09, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 27% | | 13500/50000 [2:43:05<7:21:38, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 28% | | 13750/50000 [2:46:07<7:18:05, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 28% | | 14000/50000 [2:49:08<7:15:51, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 28% | | 14250/50000 [2:52:09<7:10:31, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 29% | | 14500/50000 [2:55:10<7:08:20, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 30% | | 14750/50000 [2:58:11<7:04:34, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 30% | | 15000/50000 [3:01:13<7:04:13, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 30% | | 15250/50000 [3:04:14<6:59:37, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 31% | | 15500/50000 [3:07:15<6:53:51, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 32% | | 15750/50000 [3:10:16<6:54:39, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 32% | | 16000/50000 [3:13:17<6:48:01, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 32% | | 16250/50000 [3:16:18<6:44:55, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 33% | | 16500/50000 [3:19:19<6:45:24, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 34% | | 16750/50000 [3:22:21<6:42:48, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 34% |  | 17000/50000 [3:25:22<6:38:46, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 34% |  | 17250/50000 [3:28:23<6:36:01, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 35% |  | 17500/50000 [3:31:24<6:32:28, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 36% |  | 17750/50000 [3:34:25<6:30:22, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 36% |  | 18000/50000 [3:37:27<6:27:31, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 36% |  | 18250/50000 [3:40:28<6:22:55, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 37% |  | 18500/50000 [3:43:29<6:18:12, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 38% |  | 18750/50000 [3:46:31<6:18:54, 1.37it/s]



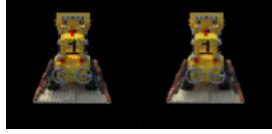
Fitting the renderer to 85x images and viewpoints: 38% |  19000/50000 [3:49:32<6:14:55, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 38% |  19250/50000 [3:52:33<6:12:00, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 39% |  19500/50000 [3:55:34<6:07:03, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 40% |  19750/50000 [3:58:35<6:04:30, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 40% |  20000/50000 [4:01:37<6:02:09, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 40% |  20250/50000 [4:04:38<5:59:01, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 41% |  20500/50000 [4:07:39<5:56:14, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 42% |  20750/50000 [4:10:40<5:53:06, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 42% | 21000/50000 [4:13:41<5:47:52, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 42% | 21250/50000 [4:16:43<5:46:00, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 43% | 21500/50000 [4:19:44<5:45:20, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 44% | 21750/50000 [4:22:45<5:42:25, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 44% | 22000/50000 [4:25:47<5:37:27, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 44% | 22250/50000 [4:28:48<5:34:44, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 45% | 22500/50000 [4:31:49<5:31:52, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 46% | 22750/50000 [4:34:50<5:29:51, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 46% | 23000/50000 [4:37:52<5:25:20, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 46% | 23250/50000 [4:40:53<5:23:30, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 47% | 23500/50000 [4:43:54<5:20:29, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 48% | 23750/50000 [4:46:55<5:15:47, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 48% | 24000/50000 [4:49:56<5:14:31, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 48% | 24250/50000 [4:52:58<5:08:04, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 49% | 24500/50000 [4:55:59<5:06:33, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 50% | 24750/50000 [4:59:00<5:05:54, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 50% | 25000/50000 [5:02:01<5:00:42, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 50% | 25250/50000 [5:05:03<4:59:09, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 51% | 25500/50000 [5:08:04<4:55:35, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 52% | 25750/50000 [5:11:05<4:52:09, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 52% | 26000/50000 [5:14:06<4:50:59, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 52% | 26250/50000 [5:17:08<4:45:36, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 53% | 26500/50000 [5:20:09<4:44:33, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 54% | 26750/50000 [5:23:10<4:40:57, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 54% | 27000/50000 [5:26:11<4:36:20, 1.39it/s]



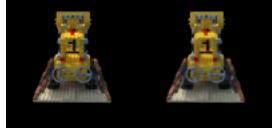
Fitting the renderer to 85x images and viewpoints: 55% | 27250/50000 [5:29:12<4:33:38, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 55% | 27500/50000 [5:32:14<4:30:50, 1.38it/s]



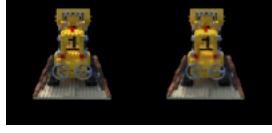
Fitting the renderer to 85x images and viewpoints: 56% | 27750/50000 [5:35:15<4:28:25, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 56% | 28000/50000 [5:38:17<4:26:01, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 56% | 28250/50000 [5:41:18<4:23:47, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 57% | 28500/50000 [5:44:20<4:19:53, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 57% | 28750/50000 [5:47:22<4:16:56, 1.38it/s]



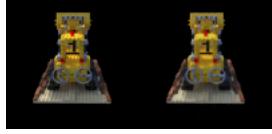
Fitting the renderer to 85x images and viewpoints: 58% | 29000/50000 [5:50:23<4:14:22, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 58% | 29250/50000 [5:53:25<4:10:22, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 59% | 29500/50000 [5:56:27<4:08:23, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 60% | 29750/50000 [5:59:28<4:05:20, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 60% | 30000/50000 [6:02:30<4:01:35, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 60% | 30250/50000 [6:05:32<3:58:59, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 61% | 30500/50000 [6:08:33<3:55:02, 1.38it/s]



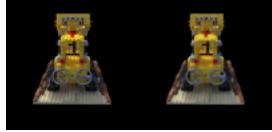
Fitting the renderer to 85x images and viewpoints: 62% | 30750/50000 [6:11:35<3:53:04, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 62% | | 31000/50000 [6:14:37<3:49:54, 1.38it/s]



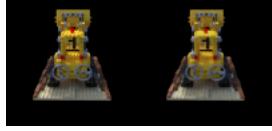
Fitting the renderer to 85x images and viewpoints: 62% | | 31250/50000 [6:17:38<3:46:53, 1.38it/s]



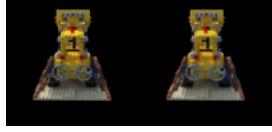
Fitting the renderer to 85x images and viewpoints: 63% | | 31500/50000 [6:20:40<3:43:55, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 64% | | 31750/50000 [6:23:42<3:40:35, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 64% | | 32000/50000 [6:26:44<3:38:24, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 64% | | 32250/50000 [6:29:45<3:35:02, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 65% | | 32500/50000 [6:32:47<3:32:53, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 66% | | 32750/50000 [6:35:49<3:28:45, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 66% |  | 33000/50000 [6:38:51<3:25:20, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 66% |  | 33250/50000 [6:41:52<3:23:06, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 67% |  | 33500/50000 [6:44:54<3:18:21, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 68% |  | 33750/50000 [6:47:56<3:16:39, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 68% |  | 34000/50000 [6:50:58<3:12:55, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 68% |  | 34250/50000 [6:53:59<3:11:12, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 69% |  | 34500/50000 [6:57:01<3:07:49, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 70% |  | 34750/50000 [7:00:03<3:03:30, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 70% | 35000/50000 [7:03:05<3:01:32, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 70% | 35250/50000 [7:06:06<2:58:51, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 71% | 35500/50000 [7:09:08<2:55:23, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 72% | 35750/50000 [7:12:10<2:52:22, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 72% | 36000/50000 [7:15:12<2:49:25, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 72% | 36250/50000 [7:18:13<2:46:42, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 73% | 36500/50000 [7:21:15<2:42:47, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 74% | 36750/50000 [7:24:17<2:39:27, 1.38it/s]



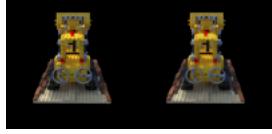
Fitting the renderer to 85x images and viewpoints: 74% | 37000/50000 [7:27:19<2:37:15, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 74% | 37250/50000 [7:30:21<2:34:40, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 75% | 37500/50000 [7:33:23<2:30:47, 1.38it/s]



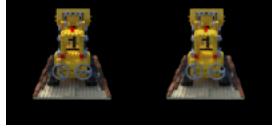
Fitting the renderer to 85x images and viewpoints: 76% | 37750/50000 [7:36:24<2:28:12, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 76% | 38000/50000 [7:39:26<2:25:54, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 76% | 38250/50000 [7:42:28<2:22:27, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 77% | 38500/50000 [7:45:30<2:18:45, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 78% | 38750/50000 [7:48:32<2:16:37, 1.37it/s]



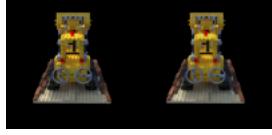
Fitting the renderer to 85x images and viewpoints: 78% | 39000/50000 [7:51:33<2:13:00, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 78% | 39250/50000 [7:54:35<2:10:45, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 79% | 39500/50000 [7:57:37<2:06:50, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 80% | 39750/50000 [8:00:39<2:04:10, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 80% | 40000/50000 [8:03:41<2:01:38, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 80% | 40250/50000 [8:06:43<1:57:58, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 81% | 40500/50000 [8:09:45<1:55:10, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 82% | 40750/50000 [8:12:47<1:52:01, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 82% | 41000/50000 [8:15:49<1:48:35, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 82% | 41250/50000 [8:18:50<1:45:49, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 83% | 41500/50000 [8:21:52<1:42:50, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 84% | 41750/50000 [8:24:54<1:40:02, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 84% | 42000/50000 [8:27:56<1:37:05, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 84% | 42250/50000 [8:30:58<1:33:58, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 85% | 42500/50000 [8:34:00<1:30:58, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 86% | 42750/50000 [8:37:02<1:26:40, 1.39it/s]



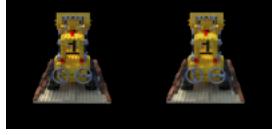
Fitting the renderer to 85x images and viewpoints: 86% | 43000/50000 [8:40:03<1:24:15, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 86% | 43250/50000 [8:43:05<1:21:45, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 87% | 43500/50000 [8:46:07<1:18:43, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 88% | 43750/50000 [8:49:09<1:15:45, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 88% | 44000/50000 [8:52:11<1:12:44, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 88% | 44250/50000 [8:55:13<1:09:33, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 89% | 44500/50000 [8:58:15<1:06:34, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 90% | 44750/50000 [9:01:16<1:03:41, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 90% | 45000/50000 [9:04:18<1:00:41, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 90% | 45250/50000 [9:07:20<57:26, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 91% | 45500/50000 [9:10:22<54:29, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 92% | 45750/50000 [9:13:24<51:21, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 92% | 46000/50000 [9:16:26<48:28, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 92% | 46250/50000 [9:19:27<45:04, 1.39it/s]



Fitting the renderer to 85x images and viewpoints: 93% | 46500/50000 [9:22:29<42:23, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 94% | 46750/50000 [9:25:31<39:22, 1.38it/s]



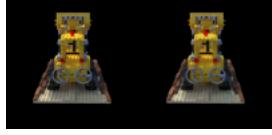
Fitting the renderer to 85x images and viewpoints: 94% | 47000/50000 [9:28:33<36:20, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 94% | 47250/50000 [9:31:34<33:15, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 95% | 47500/50000 [9:34:36<30:18, 1.37it/s]



Fitting the renderer to 85x images and viewpoints: 96% | 47750/50000 [9:37:38<27:14, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 96% | 48000/50000 [9:40:40<24:11, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 96% | 48250/50000 [9:43:42<21:11, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 97% | 48500/50000 [9:46:44<18:07, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 98% | 48750/50000 [9:49:46<15:05, 1.38it/s]



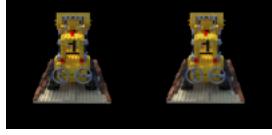
Fitting the renderer to 85x images and viewpoints: 98% | 49000/50000 [9:52:47<12:05, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 98% | 49250/50000 [9:55:49<09:04, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 99% | 49500/50000 [9:58:51<06:03, 1.38it/s]



Fitting the renderer to 85x images and viewpoints: 100% | 49750/50000 [10:01:53<03:01, 1.37it/s]

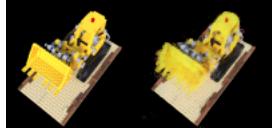


Fitting the renderer to 85x images and viewpoints: 100% | 50000/50000 [10:04:55<00:00, 1.38it/s]

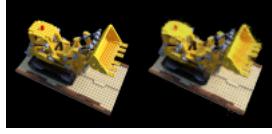
```
{'test_index': 0,  
 'quality': {'mse': 0.001204087631776929, 'psnr': 29.193419045962816}}
```



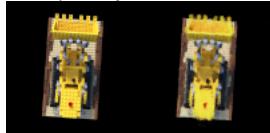
```
{'test_index': 1,  
 'quality': {'mse': 0.004533842206001282, 'psnr': 23.435335989548555}}
```



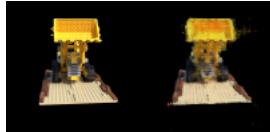
```
{'test_index': 2,  
 'quality': {'mse': 0.002511141588911414, 'psnr': 26.001287992148665}}
```



```
{'test_index': 3,  
 'quality': {'mse': 0.005727969575673342, 'psnr': 22.419992974656118}}
```



```
{'test_index': 4,  
 'quality': {'mse': 0.0035275353584438562, 'psnr': 24.52528624377273}}
```



```
{'test_index': 5,  
 'quality': {'mse': 0.0010996309574693441, 'psnr': 29.587530421380844}}
```



```
{'test_index': 6,  
 'quality': {'mse': 0.002335385186597705, 'psnr': 26.316414788590656}}
```



```
{'test_index': 7,  
 'quality': {'mse': 0.0035134812351316214, 'psnr': 24.54262361608918}}
```



```
{'test_index': 8,  
 'quality': {'mse': 0.0016367126954719424, 'psnr': 27.860275488848508}}
```



```
{'test_index': 9,  
 'quality': {'mse': 0.000952104979660362, 'psnr': 30.213151634079523}}
```



```
{'test_index': 10,  
 'quality': {'mse': 0.007672819774597883, 'psnr': 21.15045002739822}}
```



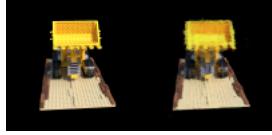
```
{'test_index': 11,  
 'quality': {'mse': 0.0009475569240748882, 'psnr': 30.233946905218737}}
```



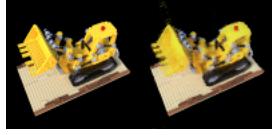
```
{'test_index': 12,  
 'quality': {'mse': 0.0008089240873232484, 'psnr': 30.92092232409381}}
```



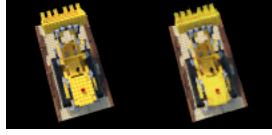
```
{'test_index': 13,  
 'quality': {'mse': 0.0020833706948906183, 'psnr': 26.812334490047046}}
```



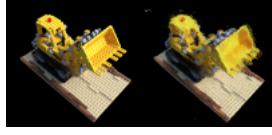
```
{'test_index': 14,  
 'quality': {'mse': 0.002727762097492814, 'psnr': 25.6419350947432}}
```



```
{'test_index': 15,  
 'quality': {'mse': 0.002334021497517824, 'psnr': 26.31895148200516}}
```



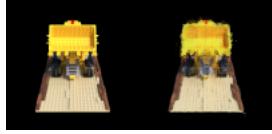
```
{'test_index': 16,  
 'quality': {'mse': 0.004051529336720705, 'psnr': 23.923810120623617}}
```



```
{'test_index': 17,  
 'quality': {'mse': 0.0009359606774523854, 'psnr': 30.287423969073846}}
```



```
{'test_index': 18,  
 'quality': {'mse': 0.0026484045665711164, 'psnr': 25.7701567194805}}
```



```
{'test_index': 19,  
 'quality': {'mse': 0.0009441359434276819, 'psnr': 30.249654683821994}}
```



```
{'test_index': 20,  
 'quality': {'mse': 0.002496316097676754, 'psnr': 26.02700422681586}}
```

