CS180 Final Project: Neural Radiance Field

Part 1: Fit a Neural Field to a 2D Image

This part involves training a neural field to reconstruct a 2D image by mapping pixel coordinates to RGB color values using a neural network.

Detailed Algorithm Explanation

1. Positional Encoding (PE)

- Neural networks often struggle to fit low-dimensional inputs such as raw pixel coordinates (x, y). To address this, **Positional Encoding (PE)** is applied, mapping the coordinates to a higher-dimensional space.
- The encoding is computed as:

$$\text{PE}(x) = \left[x, \sin(2^0 \pi x), \cos(2^0 \pi x), \ldots, \sin(2^L \pi x), \cos(2^L \pi x)\right]$$

  - $L = 10$, representing the number of frequency bands (a hyperparameter).
  - After encoding, the 2D input expands to 42 dimensions (original 2 coordinates + 20 sine and 20 cosine terms), capturing both low- and high-frequency information.

2. Neural Network (MLP)

- The neural network architecture:
  - **Input Layer**: Accepts the positional-encoded pixel coordinates (42-dimensional for each pixel).
  - **Hidden Layers**: Three fully connected layers, each with 256 neurons and ReLU activation.
  - **Output Layer**: A linear layer that outputs 3 values (RGB), followed by a Sigmoid activation to constrain the output range to $[0, 1]$
- This structure enables the network to learn a complex mapping from coordinates to color values.

---

3. Training Process

- Loss Function
  - The network minimizes the **Mean Squared Error (MSE)** between the predicted and ground-truth pixel colors:

$$\text{Loss} = \frac{1}{N} \sum_{i=1}^{N} \|\hat{C}(x_i) - C(x_i)\|^2$$

  - Where:
    - N: Number of pixels in a batch.
    - $\hat{C}(x_i)$: Predicted color for pixel iii.
    - $C(x_i)$: Ground-truth color for pixel iii.

- Optimizer
  - The Adam optimizer is used with a learning rate of 1×10^{-2}.
- Batch Sampling
  - Since the image contains a large number of pixels, a batch of $10,000$ randomly sampled pixels is used in each iteration for efficiency.
- Training Iterations
  - The model is trained for $3,000$ iterations. PSNR is calculated periodically (every 25 iterations) to monitor reconstruction quality.

---

4. Evaluation Metrics and Validation

- PSNR (Peak Signal-to-Noise Ratio)
  - This metric evaluates the quality of the reconstructed image:

$$\text{PSNR} = 20 \cdot \log_{10} \left(\frac{1.0}{\sqrt{\text{MSE}}} \right)$$

  - A higher PSNR value indicates better reconstruction quality.
- Visualization
  - Predicted images are saved periodically to show the progress of reconstruction.

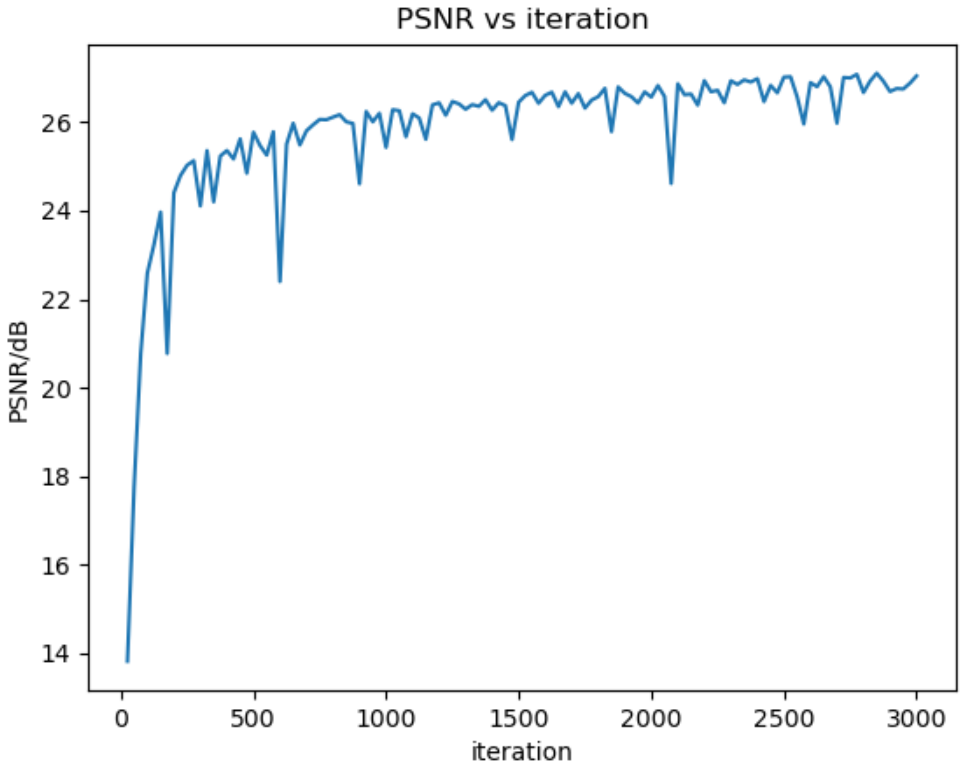
Deliverables

1. Model Architecture

- **Input Dimension**: 42 (including positional encoding).
- **Hidden Layers**: Three layers with 256 neurons each.
- **Output Dimension**: 3 (RGB).

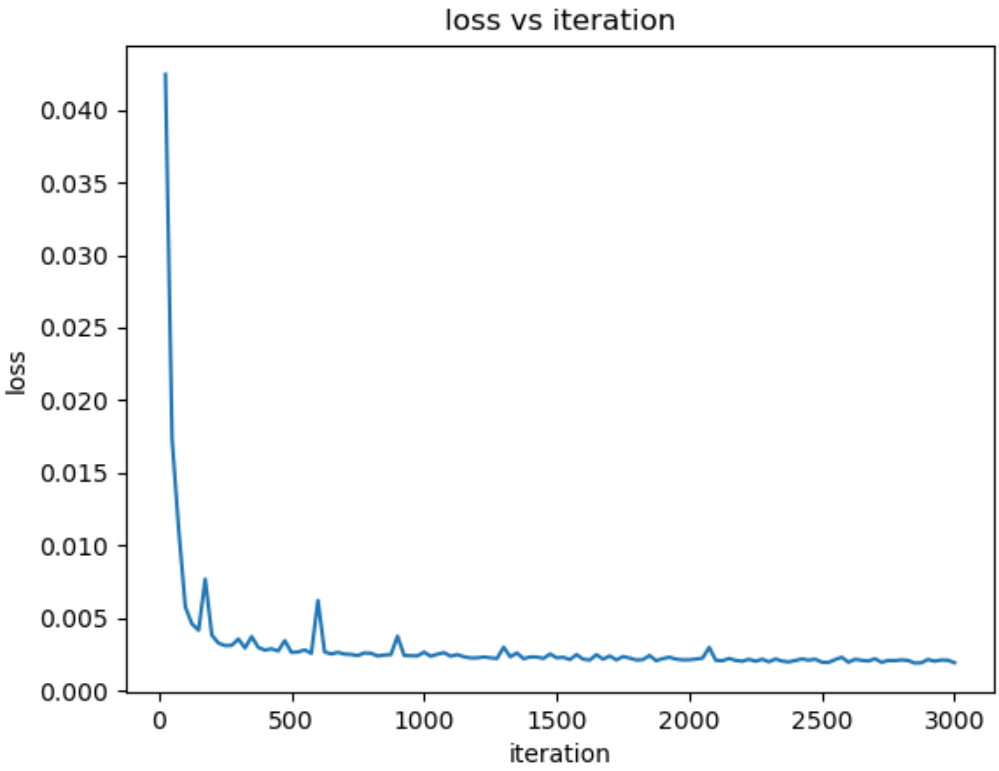
2. PSNR Curve

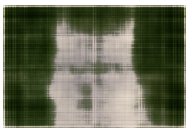
- Track PSNR at regular intervals during training.
- Generate a plot of **PSNR vs. Iterations**, showing the improvement in reconstruction quality over time.



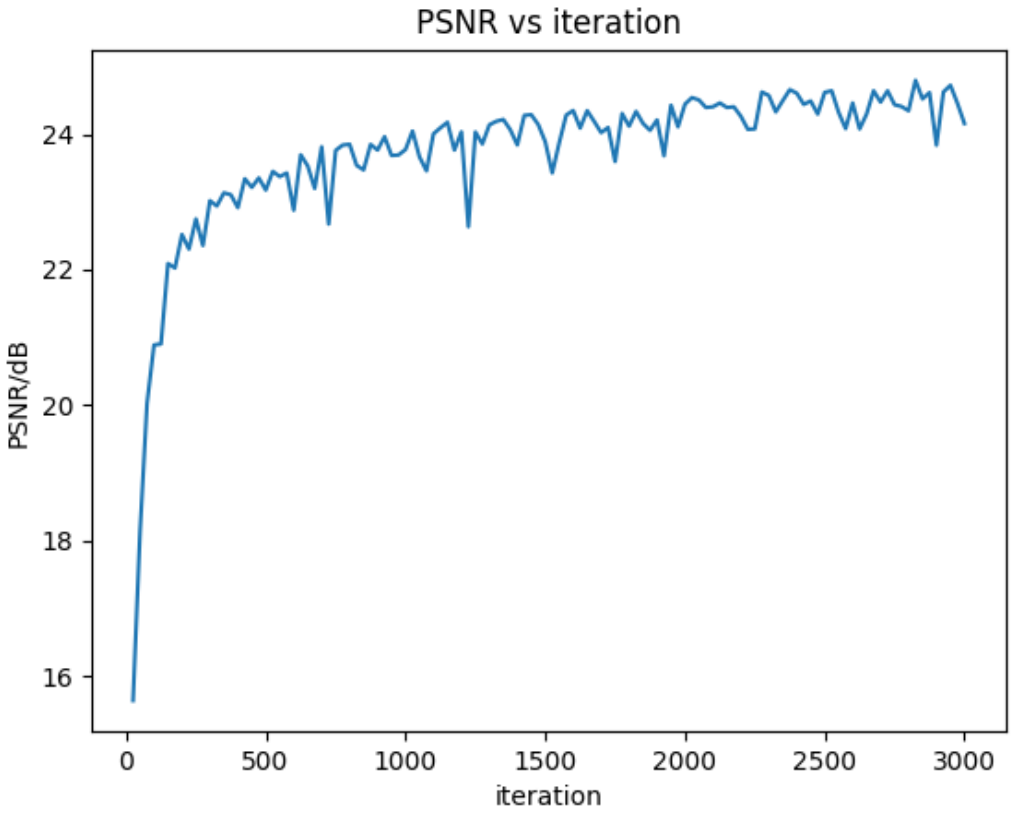
3. Training Visualization

- Save intermediate predictions to visualize the gradual reconstruction of the image.
- Save the final reconstructed image and compare it with the ground truth.

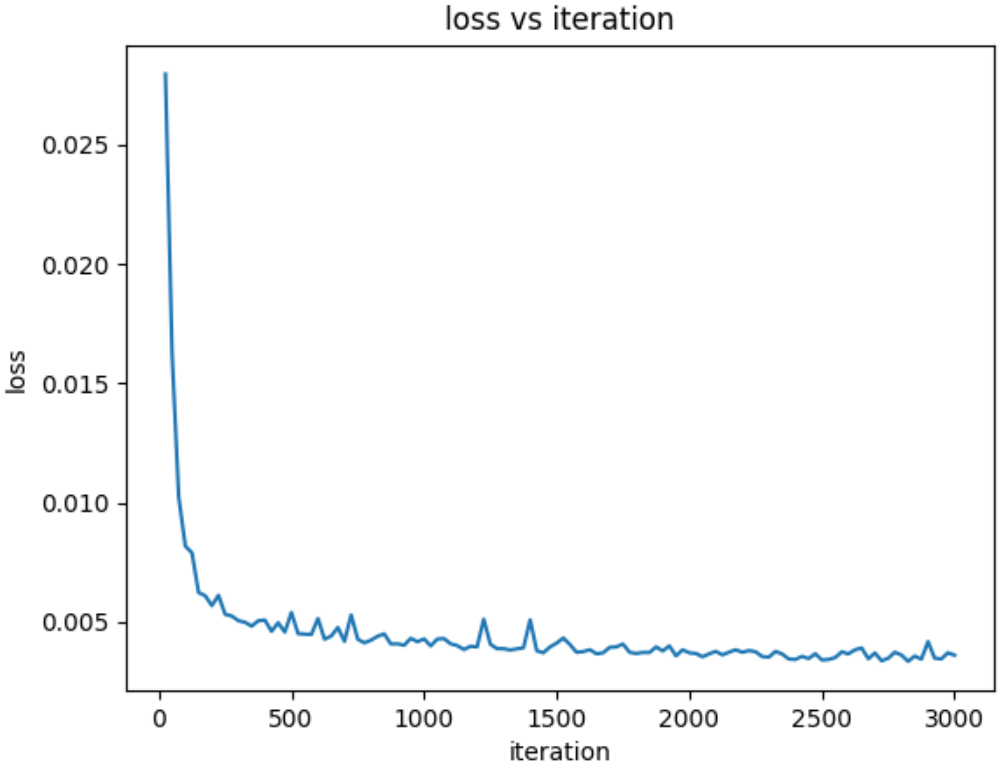


origin	**25**	**50**	**75**	**100**
				
500	1000	2000	3000	final
				

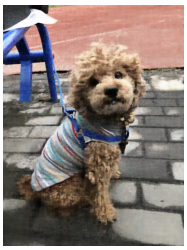
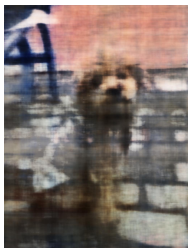
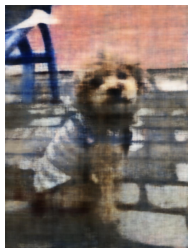
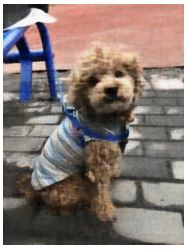
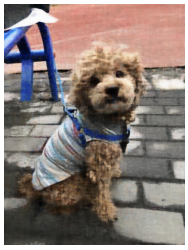
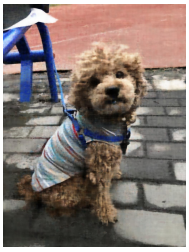
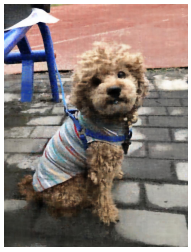
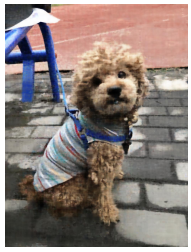
- another example:



- loss curve:



- process of training:

origin	**25**	**50**	**75**	**100**
				
500	1000	2000	3000	final
				

- gif

Part 2: Fit a Neural Radiance Field from Multi-view Images

1. Data Preparation

The `part2` function loads training and validation datasets, which include images, camera poses (`c2ws`), and camera intrinsics (focal length). These datasets are essential for training the NeRF model.

2. Ray Sampling

The `RaysData` class is responsible for generating rays corresponding to each pixel in the images. For each pixel, it computes the ray origin and direction using camera intrinsics and extrinsics. This process involves:

- **Pixel to Camera Coordinates**: Transforming pixel coordinates to camera coordinates using the intrinsic matrix K.
- **Camera to World Coordinates**: Transforming camera coordinates to world coordinates using the camera pose matrix $\mathrm{c2w}$.

This results in rays defined by their origins and directions in the world coordinate system.

3. Positional Encoding

To capture high-frequency details, the `positional_encoding` function applies a positional encoding to the input 3D coordinates and viewing directions. This encoding is defined as:

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

where ppp represents the input coordinates or directions, and LLL is the number of frequency bands. This encoding allows the neural network to model fine details in the scene.

4. Neural Network Architecture

The `NeRFNetwork` class defines the neural network architecture, which consists of:

- **Geometry Network**: Processes the positional encodings of 3D coordinates to predict the volume density σ\sigmaσ.
- **View Direction Encoding**: Encodes the viewing direction to capture view-dependent effects.
- **Color Network**: Combines features from the geometry network and the encoded viewing direction to predict the RGB color.

This design enables the network to represent complex scene geometry and appearance.

5. Volume Rendering

The `volume_rendering` function synthesizes images by integrating color and density values along each ray. For a ray $r(t) = \mathbf{o} + t\mathbf{d}$, the color $C(r)$ is computed as:

$$C(r) = \int_{t_n}^{t_f} T(t)\sigma(r(t))\mathbf{c}(r(t), \mathbf{d})\, dt$$

where:

- $T(t) = \exp\left(-\int_{t_n}^{t} \sigma(r(s))\, ds\right)$ is the accumulated transmittance, representing the probability that the ray travels from tnt_ntn to ttt without hitting any particles.
- $\sigma(r(t))$ is the volume density at point r(t)r(t)r(t).
- $\mathbf{c}(r(t), \mathbf{d}$ is the color at point $r(t)$ for viewing direction $\mathbf{d}$.

In practice, this integral is approximated using numerical quadrature by sampling points along the ray.

6. Training

The `train_nerf` function trains the NeRF model by minimizing the difference between the rendered images and the ground truth images. The training process involves:

- **Ray Sampling**: Randomly sampling rays from the dataset.
- **Point Sampling**: Sampling points along each ray within a predefined near and far plane.
- **Prediction**: Using the neural network to predict color and density for each sampled point.
- **Rendering**: Applying the volume rendering procedure to obtain the final pixel color.

- **Loss Computation**: Calculating the loss between the rendered pixel colors and the ground truth pixel colors.
- **Backpropagation**: Updating the network parameters using gradient descent.

This iterative process enables the model to learn a continuous representation of the scene.

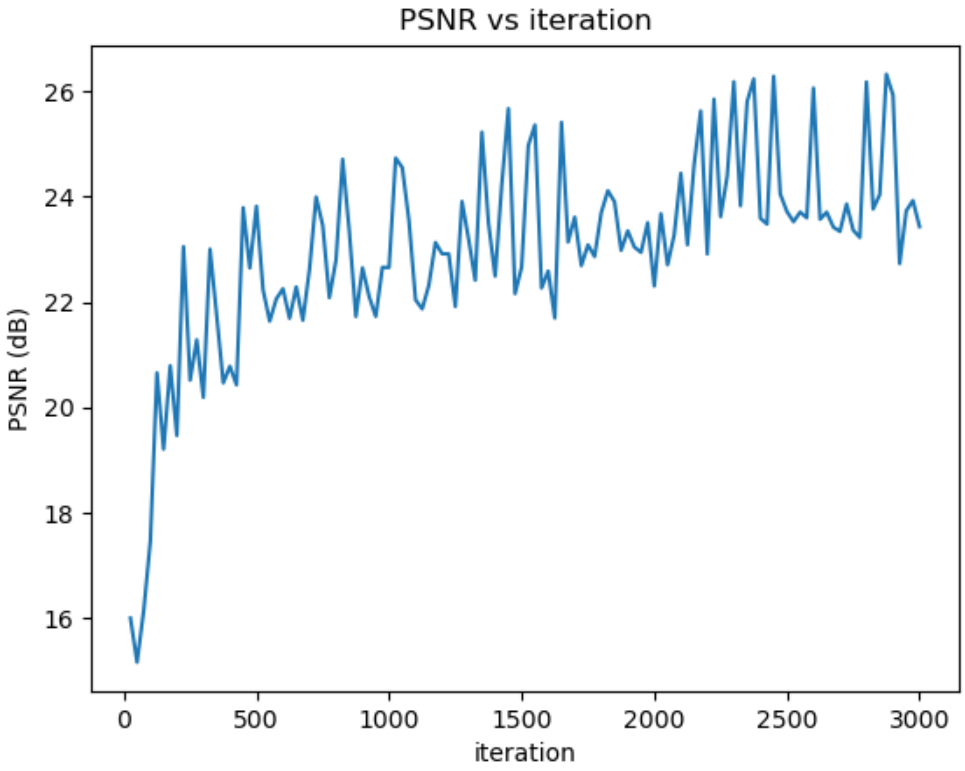
7. Rendering Novel Views

After training, the model can render novel views by specifying new camera poses. For each pixel in the desired view:

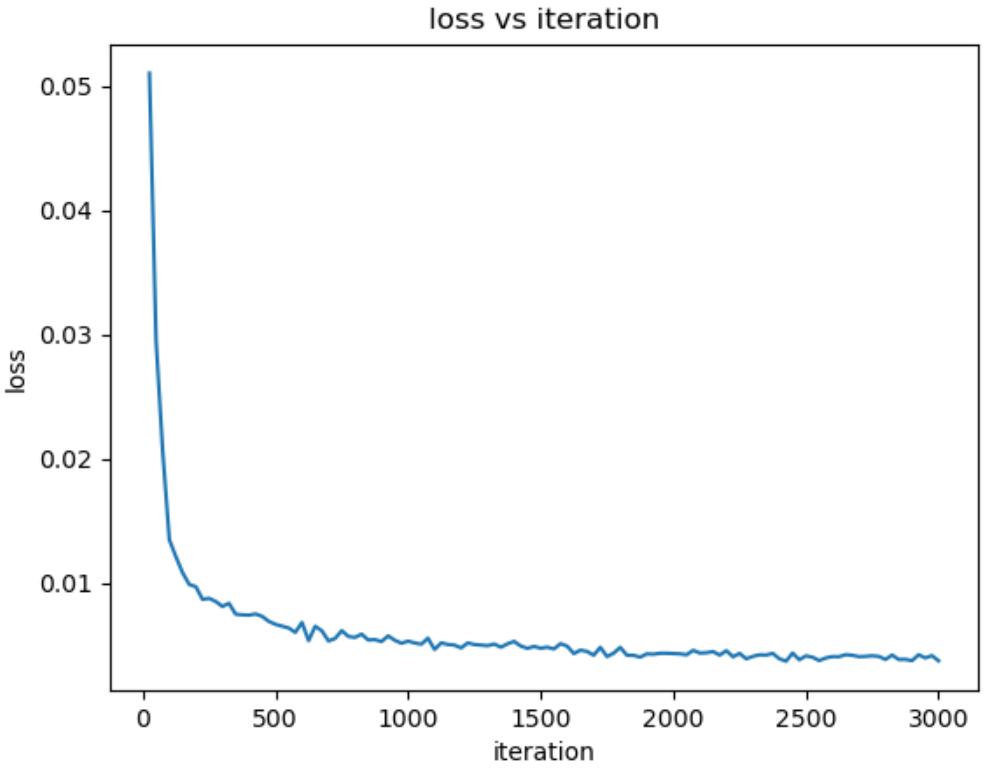
- **Ray Generation**: Compute the ray origin and direction based on the new camera pose.
- **Point Sampling**: Sample points along the ray.
- **Prediction and Rendering**: Use the network to predict color and density for each point and apply volume rendering to compute the final pixel color.

8. Deliverables

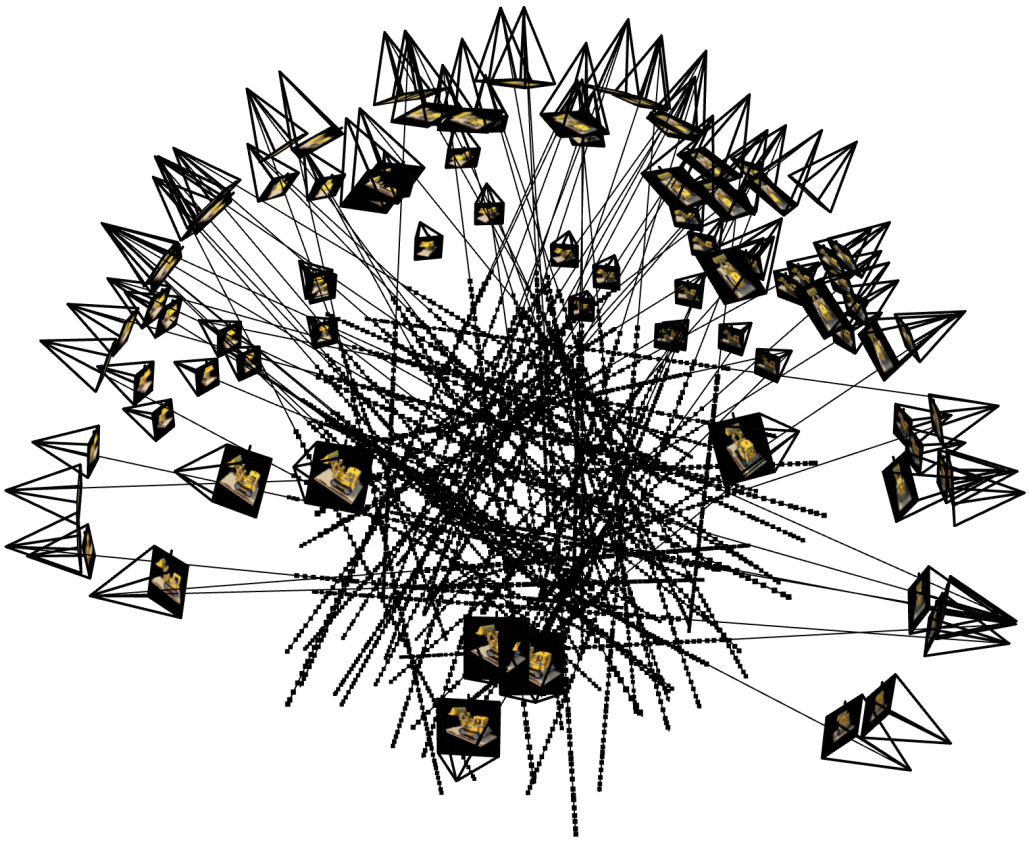
- PSNR



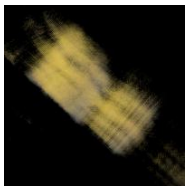
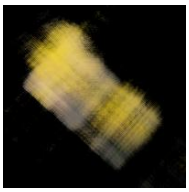
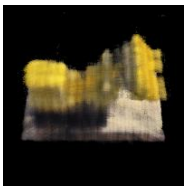
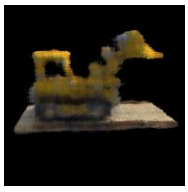
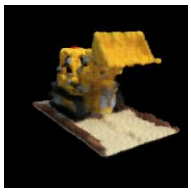
- loss curve:



- camera



- training process

25	50	75	100	125
				
250	500	1000	2000	3000
				

- render gif

Bells & Whistles

1. Data Preparation and Ray Generation

- Data Loading
  - Training and test datasets are loaded with camera intrinsics (`focal`), extrinsics (`c2ws`), and image data.
  - Intrinsic and extrinsic matrices are used to map pixel coordinates to rays.
- Ray Calculation
  - The `RaysData` class computes ray origins and directions for each pixel using:

$$\mathbf{o} = \mathbf{c2w}_{\text{translation}}, \quad \mathbf{d} = \text{normalize}(\mathbf{c2w}_{\text{rotation}} \cdot \mathbf{x}\text{camera})$$

where xcamera\mathbf{x}_{\text{camera}}xcamera is the 3D point derived from pixel coordinates and intrinsic matrix.

---

2. NeRF Model

- **Positional Encoding**:
  - Encodes high-frequency details using:

    $$\gamma(p) = \left[\sin(2^k \pi p), \cos(2^k \pi p)\right], \quad k = 0, \ldots, L-1$$

  - This enables the model to represent detailed variations in the scene.
- **Network Architecture**:
  - **Geometry Module**: Predicts density σ\sigmaσ based on 3D coordinates.
  - **View-Dependent Color Module**: Combines the geometry features with the view direction to predict RGB values.
  - The architecture is designed to model the scene's 3D geometry and appearance effectively.

---

3. Volume Rendering of Depth

- **Depth Rendering**:
  - For each ray, depths are composited instead of colors using:

    $$\text{Depth}(\mathbf{r}) = \sum_i T_i \left(1 - \exp(-\sigma_i \delta_i)\right) z_i$$

    where:
    - $T_i = \prod_{j=1}^{i-1} \exp(-\sigma_j \delta_j)$ (transmittance)
    - σ_i is the density at the i^{th} point.
    - z_i is the sampled depth.
- **Implementation**:
  - Depths are computed for each point along rays using `volume_rendering_depth`.
  - The output depth maps correspond to the expected depth of the scene for each pixel.

---

4. Depth Map Video Generation

- **Rendering for Test Poses**:
  - For each test camera pose:
    1. Generate rays for every pixel.
    2. Sample points along the rays within a predefined near-far range.
    3. Predict density σ and RGB values using the trained NeRF model.
    4. Composite depth values using the `volume_rendering_depth` function.
- **Output**:
  - Save depth maps for each camera view as images.
  - Use the `create_animation` utility to combine these depth maps into a video.

---

5. Training

- **Optimization**:
  - Train NeRF using a loss function comparing predicted RGB values to ground truth pixel colors.
  - While training focuses on RGB, the same density predictions are leveraged for depth rendering.
- **Validation**:

- Validate using held-out views and monitor PSNR metrics for color predictions to gauge convergence.

---

6. Novel View Depth Rendering

- The trained model generalizes to unseen camera poses.
- Depth maps for these novel views are computed by following the same depth compositing pipeline.