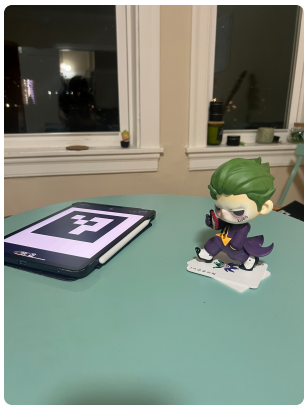
Project 4: Neural Radiance Field

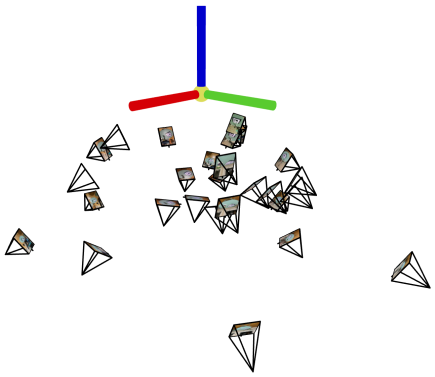
Part 0: Calibrating Your Camera and Capturing a 3D Scan

After calibrating my camera, I captured a 3D scan of my Pop Mart figure. Below are the estimates of the camera's extrinsic parameters.

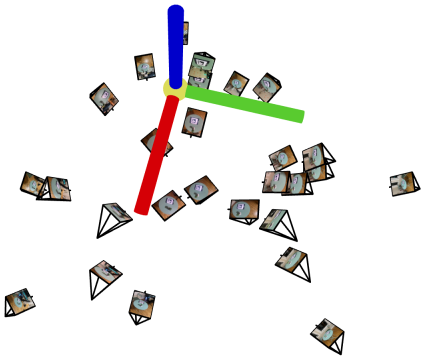
Example Image



Horizontal View



Vertical View



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

Architecture:

- Layers: 4
- Width: 256 (tested 32 and 256)
- Positional Encoding: 2D positional encoding with max frequency L (tested with L=0 and L=4)
- Input Dimension: 2 + 4L
- Output: 3 Channels (RGB), using Sigmoid to output values in [0,1]

Training Hyperparameters:

- Optimizer: Adam
- Learning Rate: 0.01
- Loss Function : Mean Squared Error (MSE)
- Batch Size: 10,000 pixels per iteration
- Iterations : 2,000

We train a coordinate-based MLP with 4 hidden layers, ReLU activations, and a Sigmoid RGB output. The input is 2D pixel coordinates encoded with sinusoidal positional encoding of dimension 2+4L. We optimize with Adam with a learning rate of 0.01, MSE loss, batch size 10,000 pixel samples, for 2,000 iterations per setting.

Fox Image Training Progression:

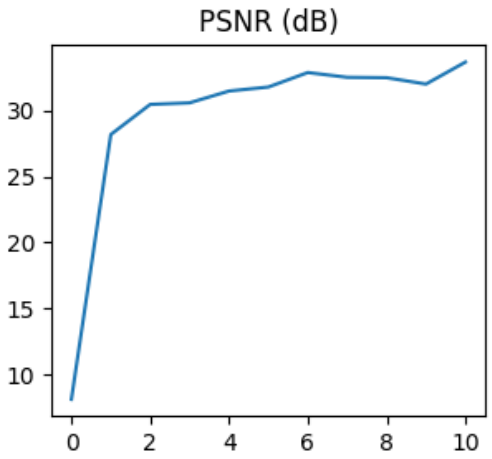
Below are the results of my training progression on the provided test image with different hyperparameters.

Original Image

Iterations

Final Results

PSNR Curve



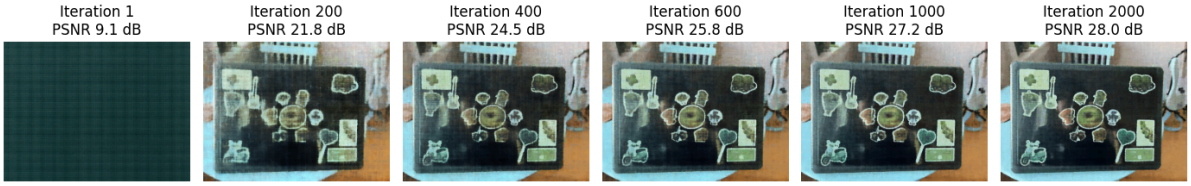
Laptop Image Training Progression:

Now, I use an image of my laptop to further test my model with different hyperparameters. My results from the test image and this laptop image illustrate the importance of positional encoding in capturing high-frequency details. The PSNR curve for the laptop image grew slower compared to the fox image, corroborating with the fact that the laptop image is more complex and requires more iterations to converge.

Original Image

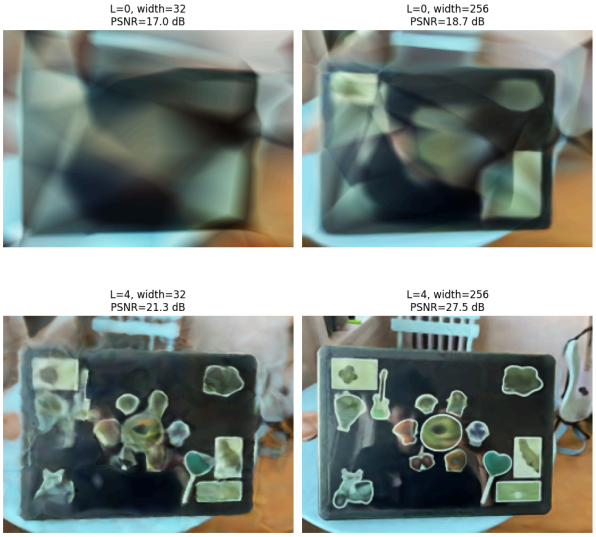
Original Image

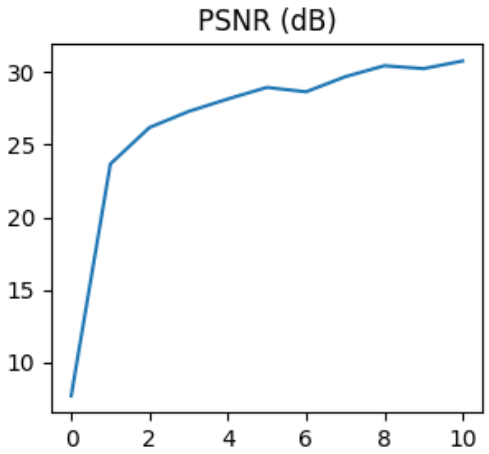
Iterations



Final Results

PSNR Curve





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

2.1: Create Rays from Cameras

For 2.1, I first implemented a transform(T, x) function to convert normalized pixel coordinates in an image to 3D coordinates. To convert 2D pixel coordinates back to 3D camera coordinates, I implemented pixel_to_camera(K, uv, s), which uses the camera extrinsic matrix to calculate the transformations. Lastly, I defined pixel_to_ray(K, c2w, uv), which combines the previous two functions to compute the rays for each pixel in the image by calculating the origin and direction of the rays in world coordinates.

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

2.2: Sampling

For 2.2, I expanded on my functions from 2.1 to randomly sample a fixed number of rays per training iteration. My process for this function was to load all images and their corresponding poses (c2w) and intrinsics (K). Then, I computed the rays for each pixel in every image using pixel_to_ray(K, c2w, uv). Finally, I randomly selected a fixed number of rays from the entire set of rays across all images to return a batch containing ray origins, directions, and RGB values.

2.3: Putting the Dataloading All Together

Here are the visualizations of the work I did from the previous 2 parts using the lego dataset.

All Cameras and Rays

Rays from One Camera