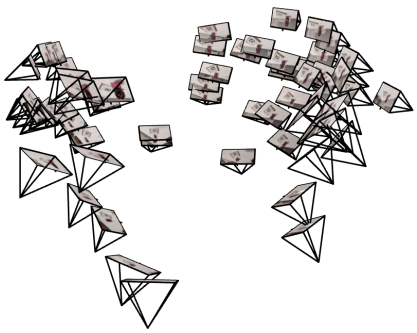
Neural Radiance Fields (NeRF) Project

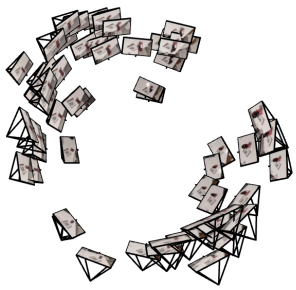
CS180/280A Computer Vision Assignment

Part 0: Camera Calibration and 3D Scanning

In this part, we calibrated a camera using a checkerboard pattern and captured a 3D scan of an object. The visualization below shows the camera frustums captured during the scanning process.



Camera Frustums Visualization - View 1



Camera Frustums Visualization - View 2

Technical Details: We used OpenCV to calibrate the camera, obtaining intrinsic parameters including focal length and principal point. The extrinsic parameters (camera positions and orientations) were computed from the captured images of the object. The visualization shows how multiple camera views are used to capture a 3D scene from different angles.

Part 1: Neural Field Fitting to 2D Image

Model Architecture

We implemented a neural field model with the following architecture:

- **Layers:** 4 hidden layers
- **Width:** 256 neurons per layer
- **Positional Encoding:** 10 frequency bands for coordinates
- **Learning Rate:** 5e-4 with Adam optimizer

- **Activation:** ReLU for hidden layers, Sigmoid for output

Training Progression

Iteration 100

Iteration 200

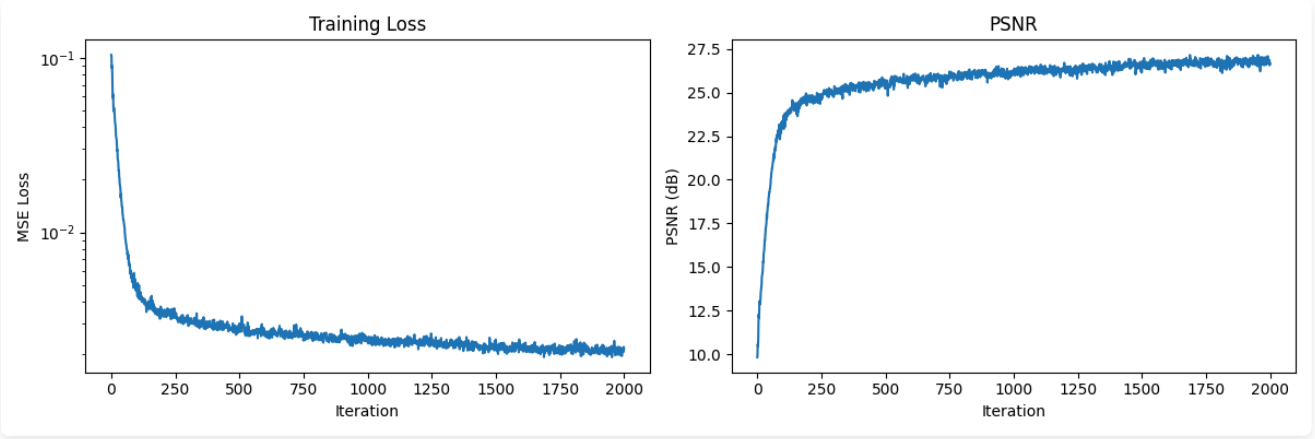
Iteration 500

Iteration 1000

Iteration 2000

Original Image

Training History



PSNR vs Training Iterations

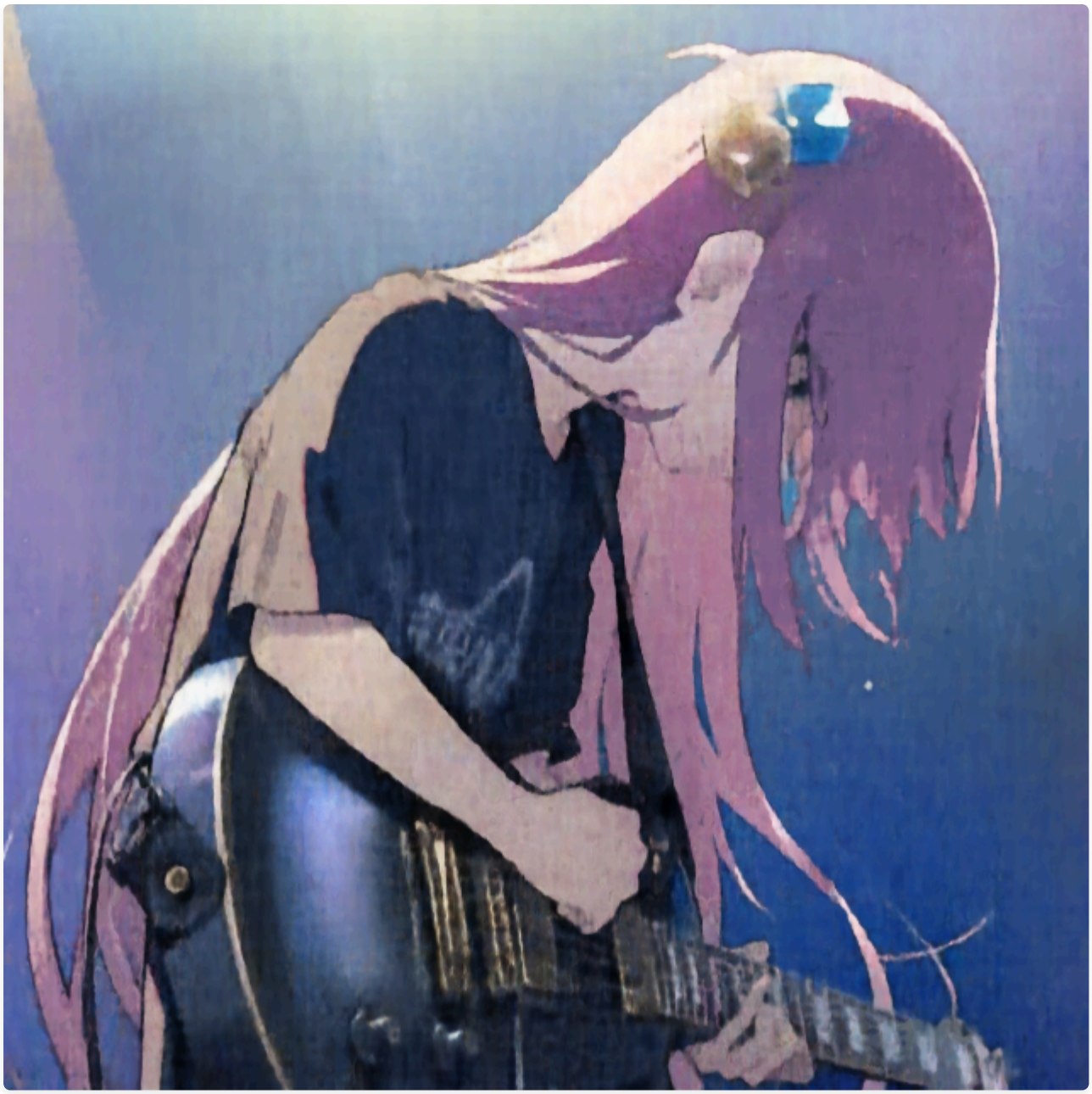
Training Progression

Iteration 100

Iteration 200

Iteration 500

Iteration 1000



Iteration 2000

Original Image

Hyperparameter Comparison

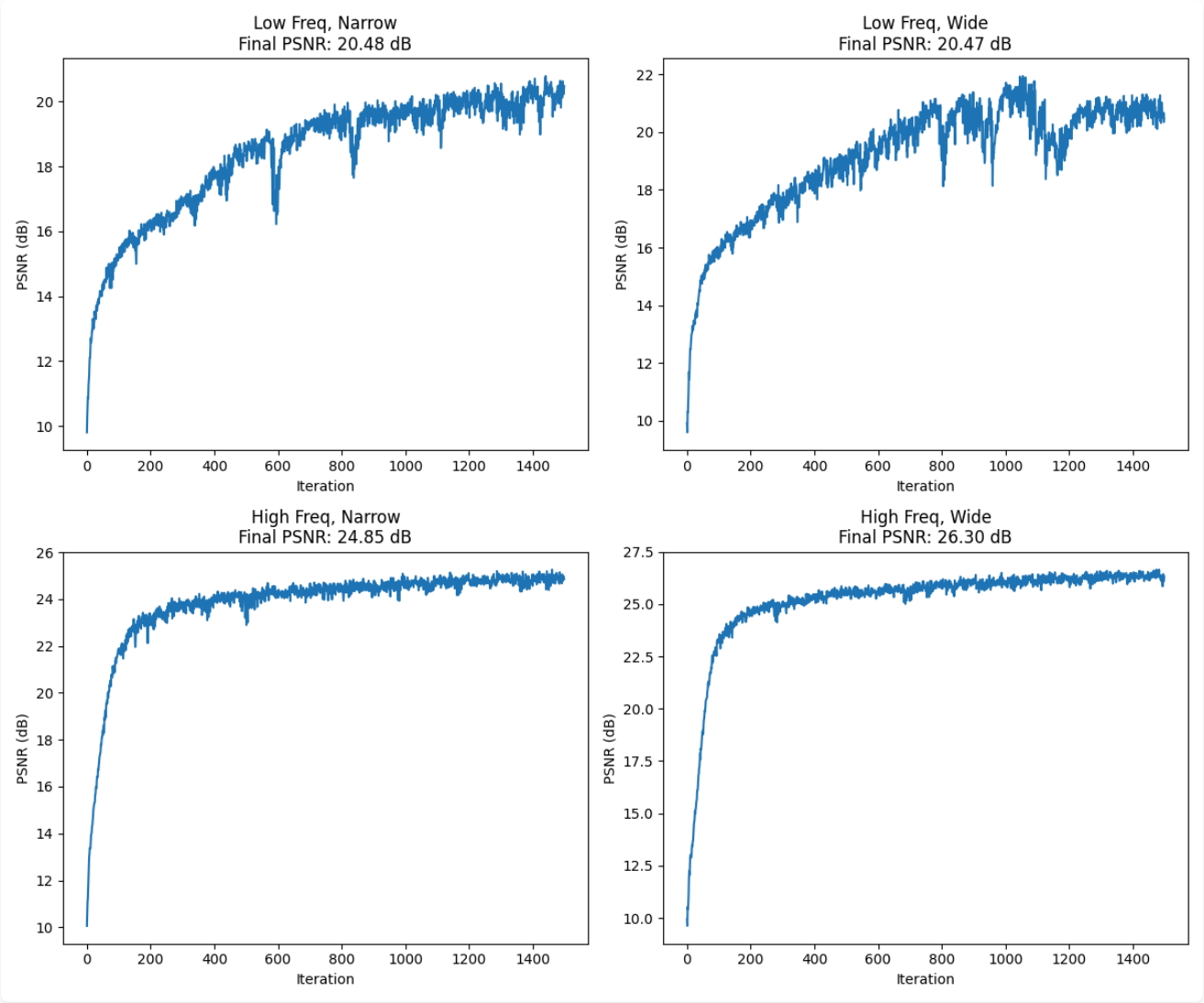
Freq: 3, Width: 64

Freq: 3, Width: 256

Freq: 10, Width: 64

Freq: 10, Width: 256

Training PSNR Curve



PSNR vs Training Iterations

Part 2: Neural Radiance Field from Multi-view Images

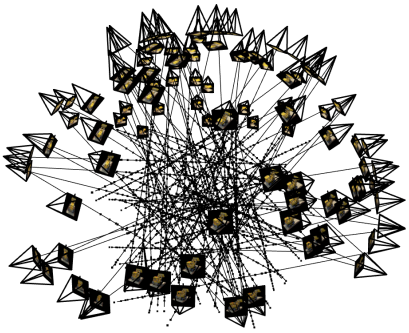
Implementation Overview

Our implementation consists of several key components:

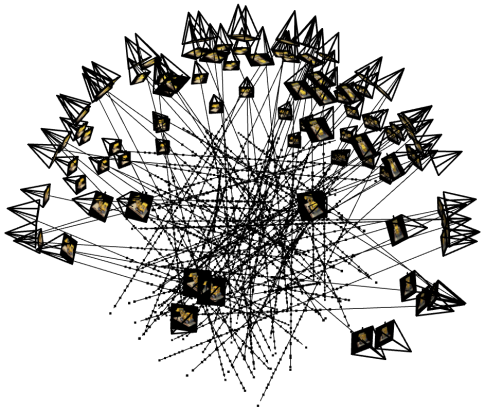
1. **Ray Generation:** Convert pixel coordinates to rays in 3D space using camera parameters
2. **Sampling:** Sample points along rays with hierarchical sampling strategy
3. **NeRF Network:** 8-layer MLP with positional encoding for coordinate and view direction

4. **Volume Rendering:** Implement volume rendering equation to compute pixel colors

Rays and Samples Visualization

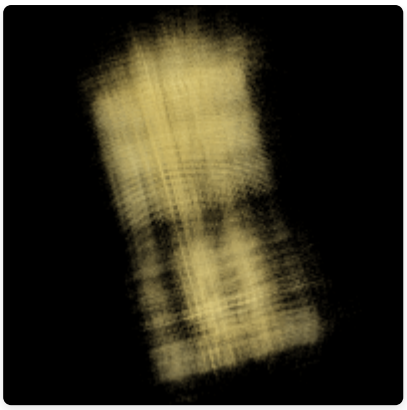


Rays and Samples Visualization (Up to 100 rays)

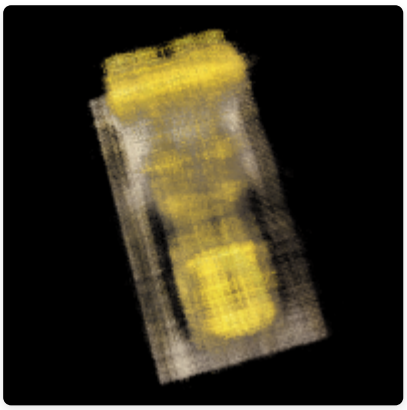


Camera Frustums with Rays

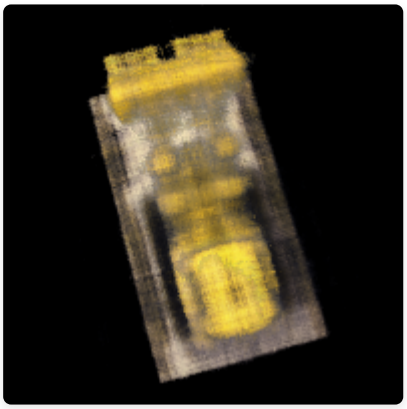
Training Progression



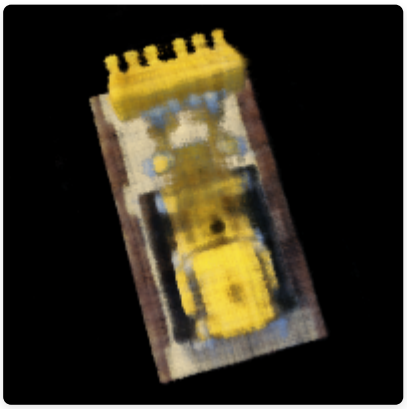
Epoch 100



Epoch 300

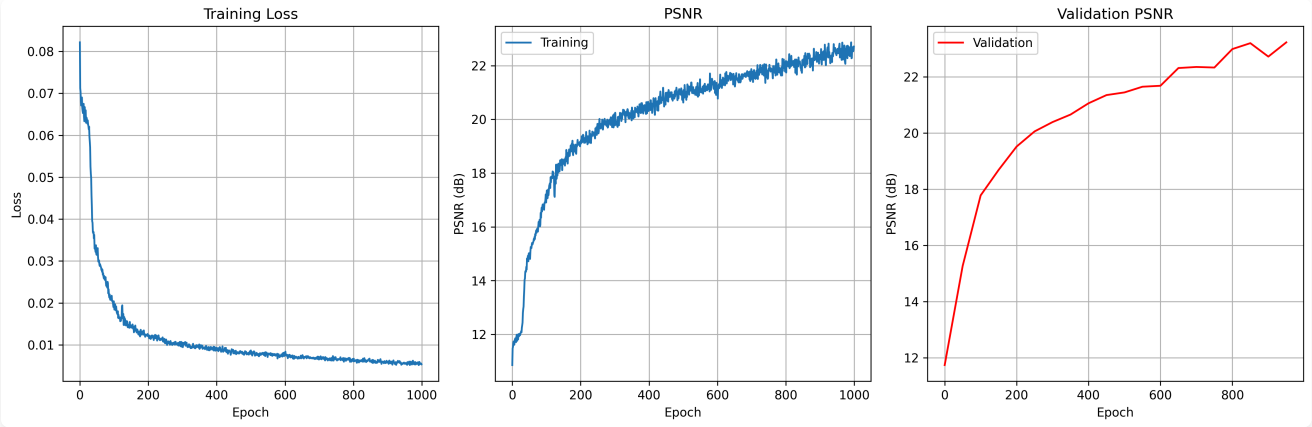


Epoch 500



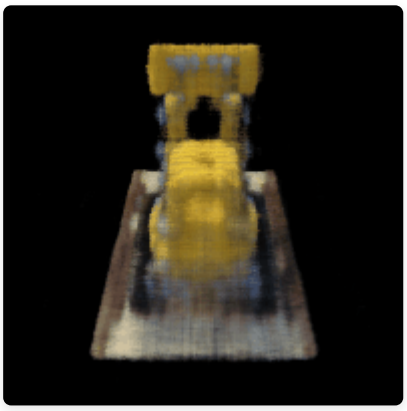
Epoch 1000

Validation PSNR Curve



Training and Validation PSNR

Spherical Rendering Video



Lego Scene Novel View Synthesis

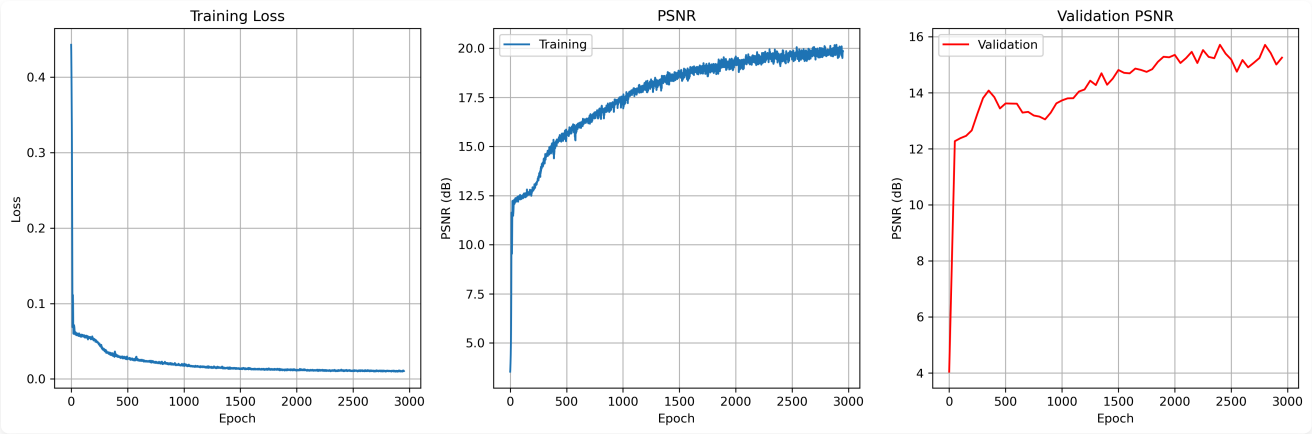
Part 2.6: Training with Custom Data

Implementation Details

> To train on our custom data, we made several modifications to handle large images and memory constraints:
>
> - **Image Resizing:** Resized images to 400px height to manage memory usage
> - **Memory Optimization:** Implemented chunked processing and CPU-GPU memory management
> - **Hyperparameters:**

- Near plane: 0.02, Far plane: 0.5 (adjusted for real-world scale)
- Batch size: 10000 rays per iteration
- Samples per ray: 64
- Learning rate: 5e-4

Training Loss Curve



Training Loss and PSNR for Custom Data

Intermediate Renders

Epoch 500

Epoch 100

Epoch 1500

Epoch 3000

Novel View Synthesis

Novel Views of Custom Object

Novel Views of Custom Object

Novel Views of Custom Object

Technical Challenges and Solutions

During the implementation, we encountered several challenges:

1. **Memory Management:** High-resolution images caused CUDA out-of-memory errors. We solved this by:
    - Resizing images to manageable dimensions
    - Implementing chunked processing for ray sampling
    - Moving non-essential data to CPU memory
2. **Ray Generation:** Converting pixel coordinates to 3D rays required careful handling of coordinate systems and camera parameters
3. **Training Stability:** Ensuring stable training required careful tuning of learning rates and batch sizes