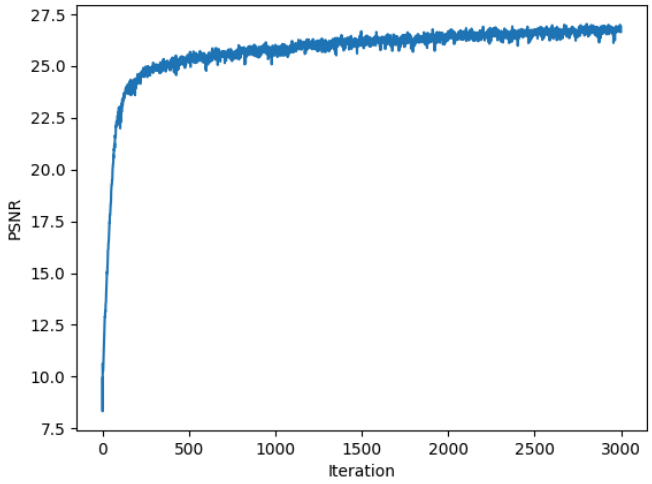
Neural Radiance Field

Anders Museth

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

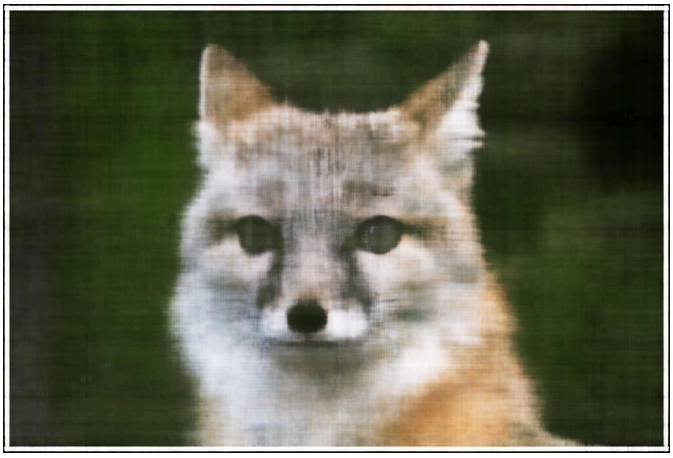
As a prelude to studying 3D neural radiance fields (NeRF), I first implemented a NeRF in 2D. This required training a neural network to map input coordinates u and v to the corresponding RGB values in an image. During training, I compared the predicted RGB values with those in the actual image to compute the Mean Squared Error (MSE) loss. Based on this, I then performed backpropagation to refine the network's performance. The architecture of the network incorporates a Sinusoidal Positional Encoding with L = 3. This is followed by three sequential blocks, each consisting of a 256-unit linear layer paired with a ReLU activation function. The sequence concludes with a three-dimensional linear layer, which employs a sigmoid activation function. I utilized the Adam optimizer for the training process, setting the learning rate at 0.01. The training involved is using 10,000 randomly selected UV coordinates and was conducted over 3,000 iterations.



PSNR for the fox image

2nd iteration of training

75th iteration of training

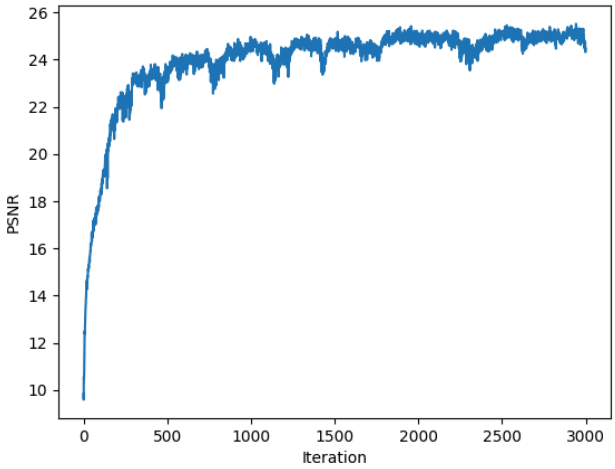


300th iteration of training

1500th iteration of training

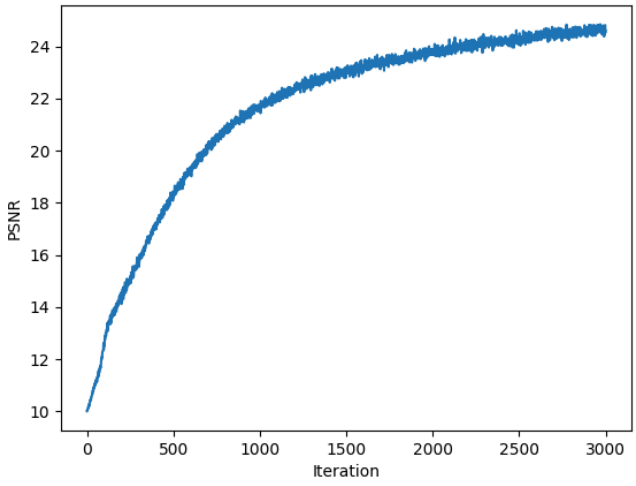
3000th iteration of training

Subsequently, I experimented with adjusting the hyperparameters to observe their impact on the quality and accuracy of the predicted images. Specifically, I tested the effects of reducing the Sinusoidal Positional Encoding value (L) to 4 and lowering the learning rate.



PSNR for the fox image with L =4

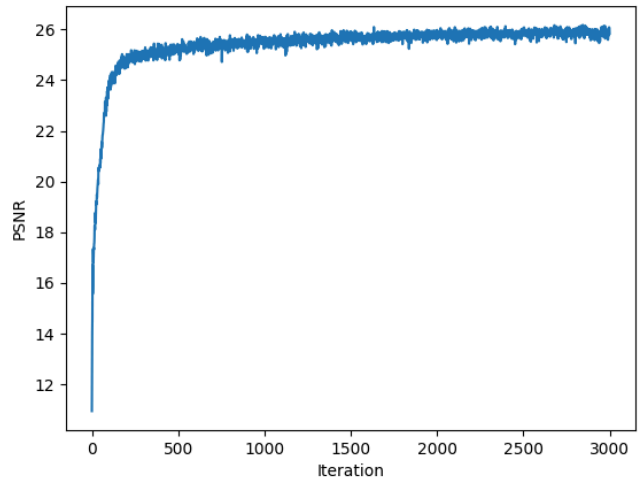
3000th iteration of training with L =4



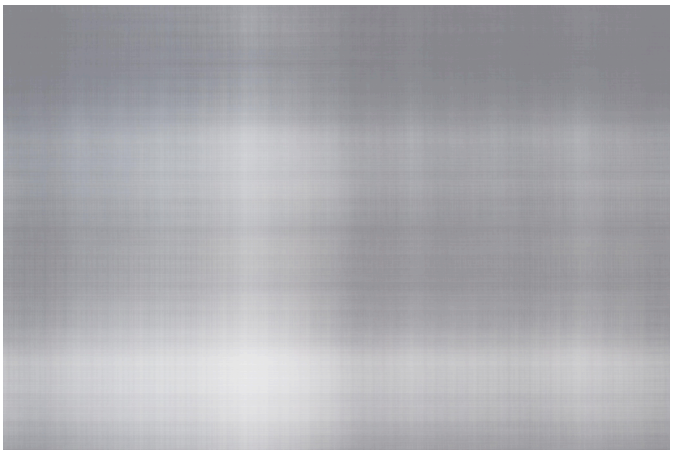
PSNR for the fox image with learning rate = 0.0001

3000th iteration of training with learning rate = 0.0001

The images above clearly demonstrate that when the Sinusoidal Positional Encoding value (L) is set to 4, there is insufficient variation in the positional encodings to capture finer details. Additionally, a lower learning rate implies that longer training durations are necessary to further reduce the loss and improve image quality.



PSNR for the dog image


10th iteration of training


1500th iteration of training


3000th iteration of training

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

Now, I am progressing towards implementing Neural Radiance Fields (NeRFs) in 3D. The initial step in this process involved developing essential helper functions for volume rendering and projecting rays from our images.

Part 2.1: Create Rays from Cameras

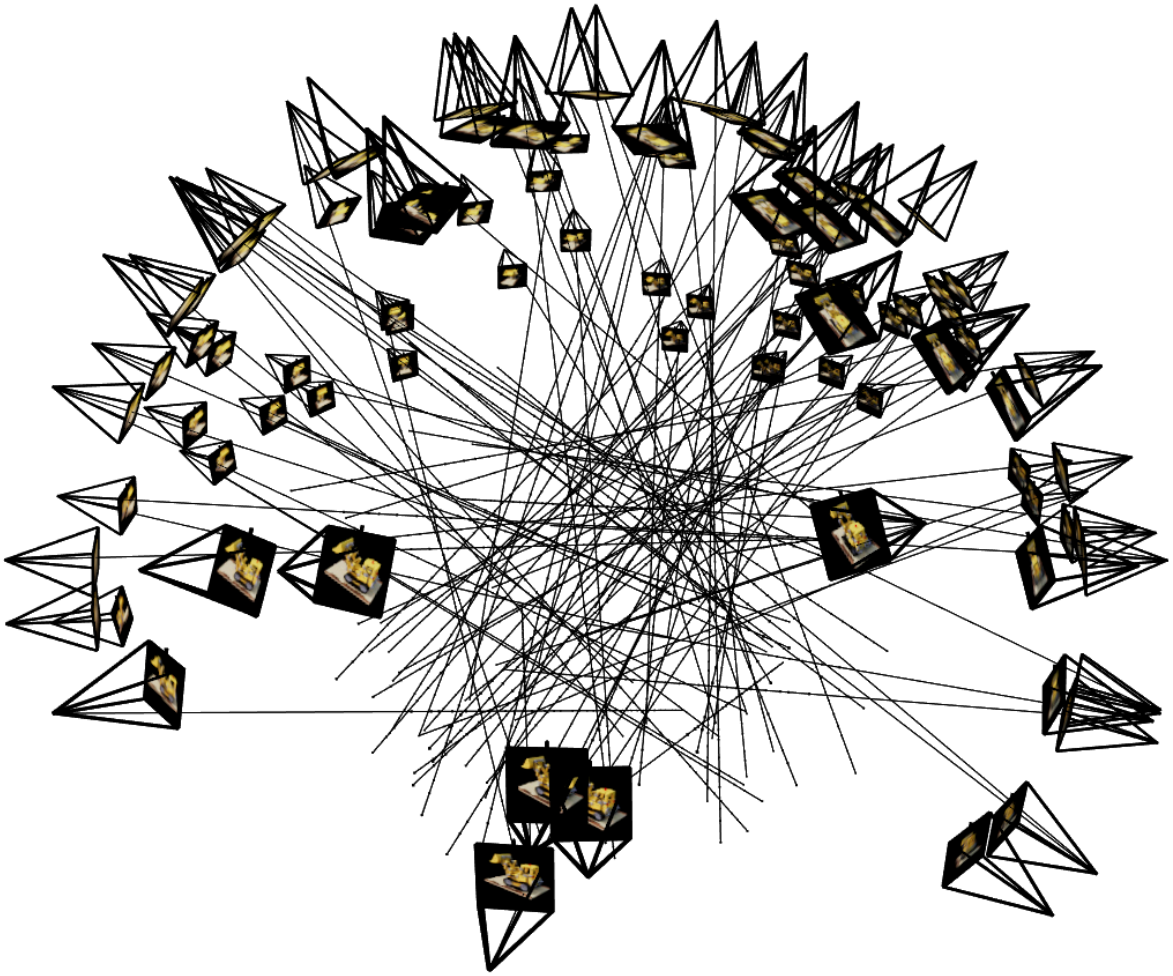
The first helper function I developed was a basic matrix transformation function. It is designed to accept a batch of points along with a transformation matrix and outputs the transformed batch of points. Subsequently, I developed an additional function designed to transform a point from the pixel coordinate system back to the camera coordinate system. This is achieved by taking a pixel index, an intrinsic matrix, and a specified depth along the optical axis. The inverse of the intrinsic matrix is used to transform the image indices (UV coordinates), which are then scaled by the depth. The final helper function I created takes a camera-to-world matrix, an intrinsic matrix, and UV coordinates to calculate the camera's ray origin point and the direction of those rays relative to their respective UVs. This is accomplished by transforming the camera's position using the inverse of the camera-to-world matrix's transpose, effectively determining the camera's origin in the scene. Then, the function determines the ray direction for each UV. This is achieved by employing the previously developed helper function to identify the corresponding 3D points of the UVs. These points are then transformed into world coordinates, allowing the calculation of rays that pass through both these points and the camera's origin.

Part 2.2: Sampling

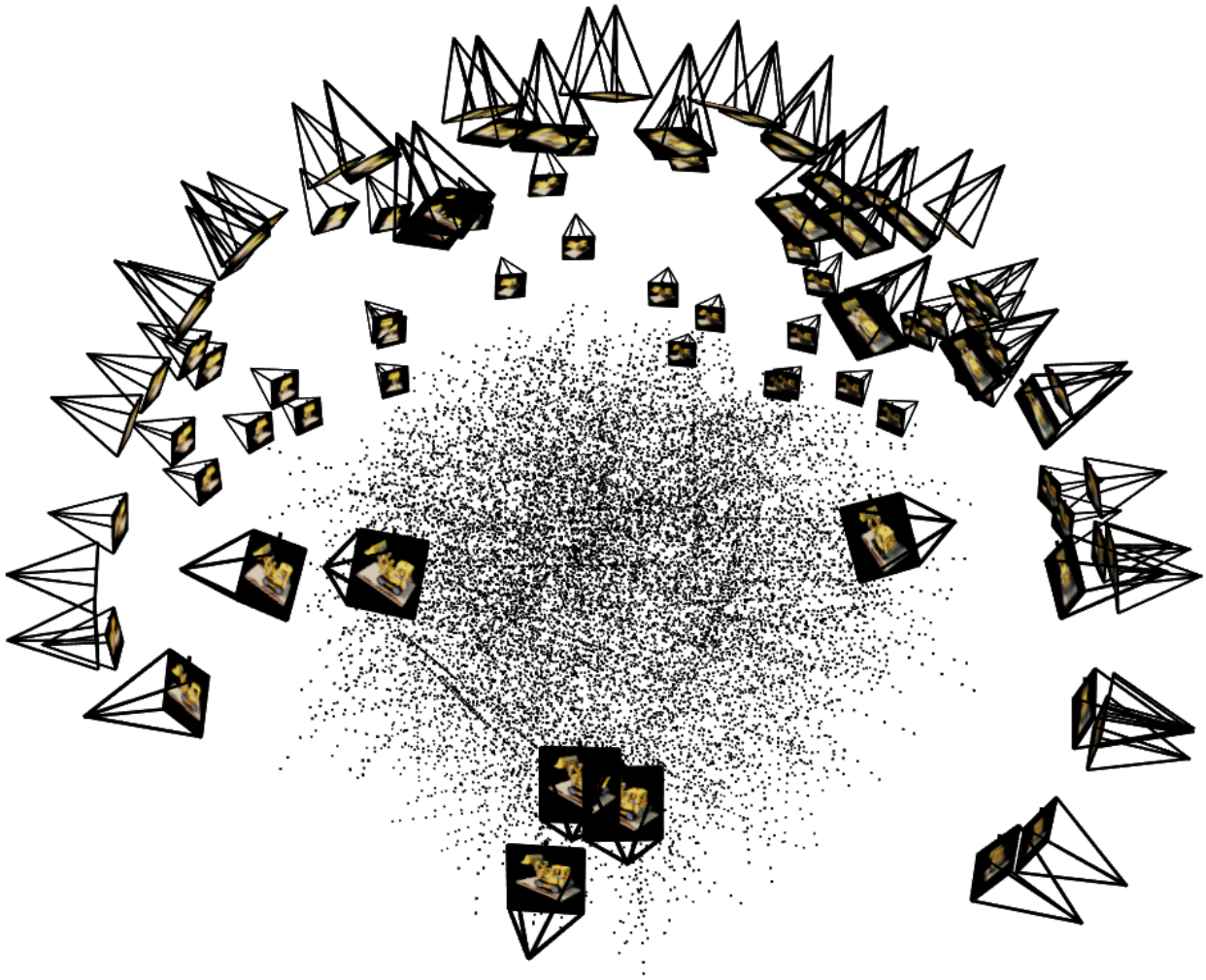
The subsequent step involves sampling numerous rays by selecting random UVs from various multi-view images. After sampling these rays, I then sample 64 points along each ray. These points are uniformly spaced along the ray, with the exception of during training, where the step size between points is slightly perturbed.

Part 2.3: Putting the Dataloading All Together

When all these components are integrated, the result is a representation of what a single random sample might look like during training:



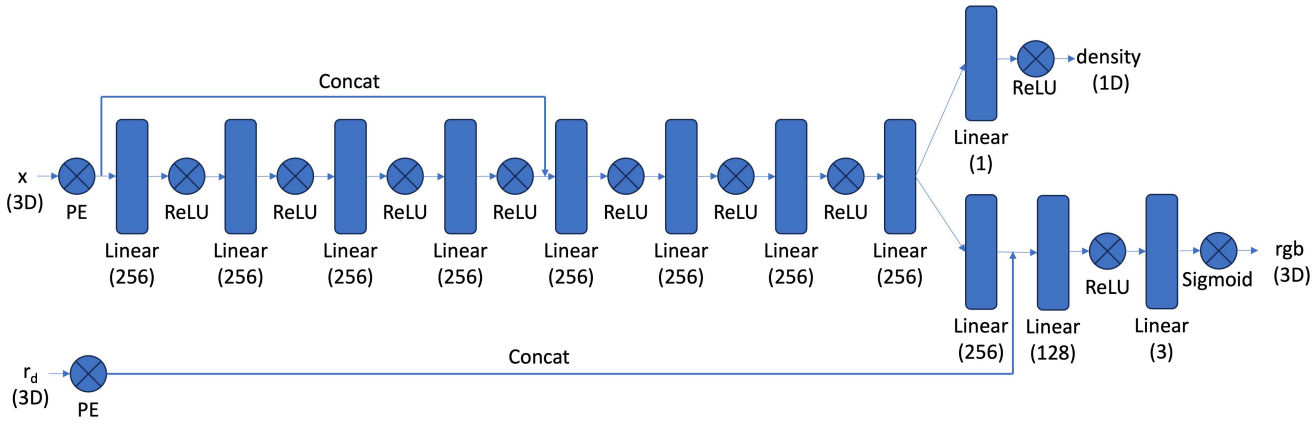
One ray shot from each image with 64 point samples along the ray



Points sampled from rays above

Part 2.4: Neural Radiance Field

The following step involves constructing the neural network for the NeRF. I employed the architecture recommended by the project guidelines, which is illustrated below:



The PE for x is L = 10 and the PE for r_d is L =4

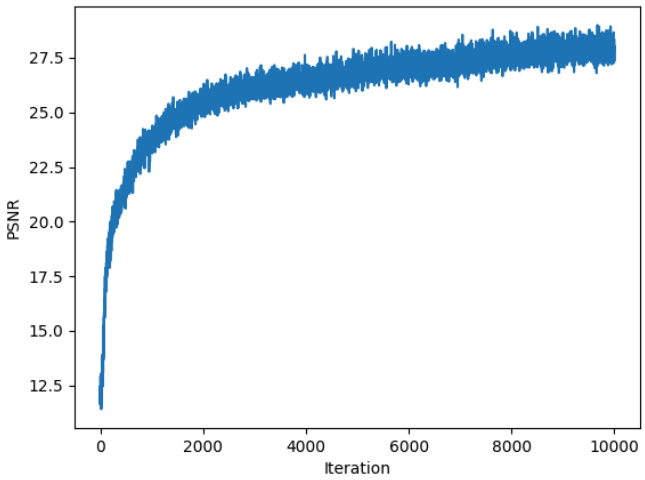
Part 2.5: Volume Rendering

The final preparatory step before training the NeRF is to implement volume rendering. This process involves obtaining the network's predicted density for each sampled point, along with the RGB value of each sample. We then calculate a weighted sum of these components to approximate the color of a specific pixel in an image based on the neural network's prediction. This calculation is performed using the following equation:

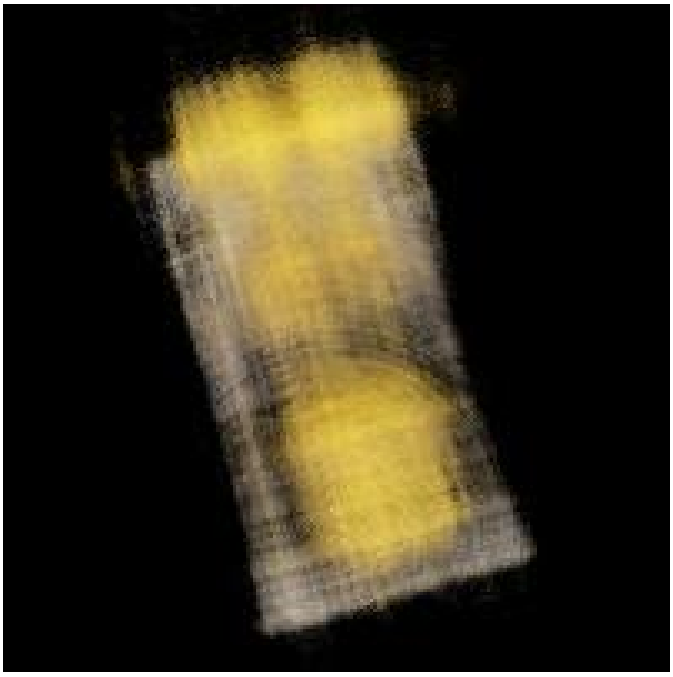
$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \left(1 - \exp\left(-\sigma_i \delta_i\right)\right) \mathbf{c}_i, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$$

In this context, σ represents the predicted density, c is the predicted RGB color, and δ denotes the step size between the points. Additionally, T represents the probability that the ray terminates at a given point along its path. I trained the NeRF by processing 10,000 random rays at each of the 10,000 iterations. The training involved computing the Mean Squared Error (MSE) loss between the predicted RGB values of the samples and their actual values. Subsequently, I performed backpropagation using the Adam optimizer, with a learning rate set at 5×10^−4.

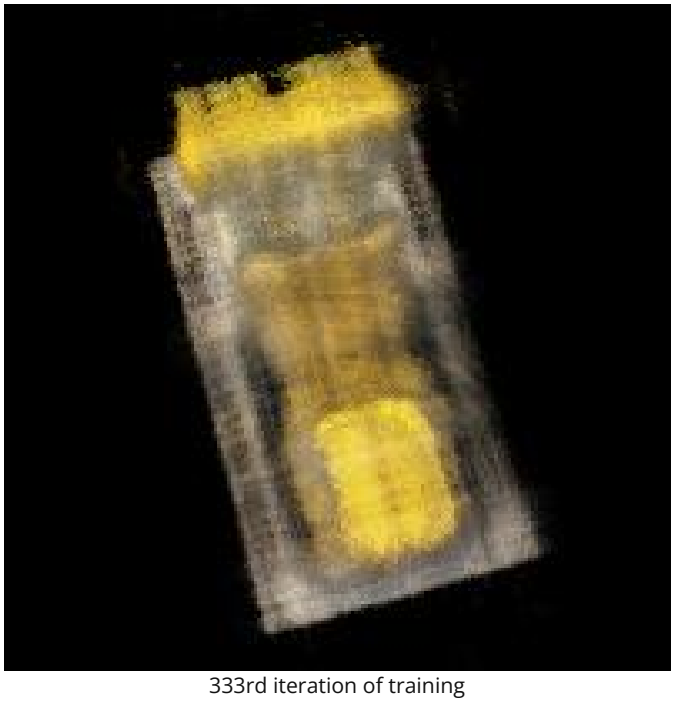
Results



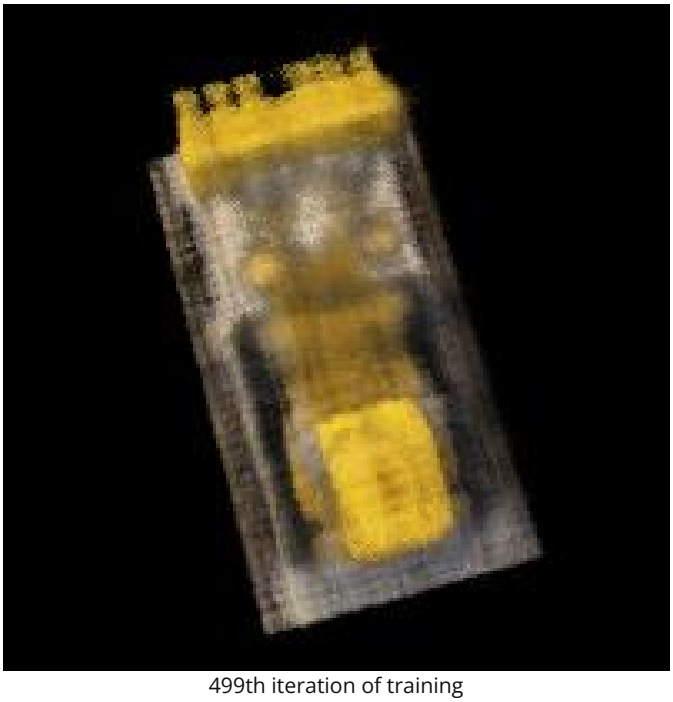
PSNR over 10k iterations with 10k samples



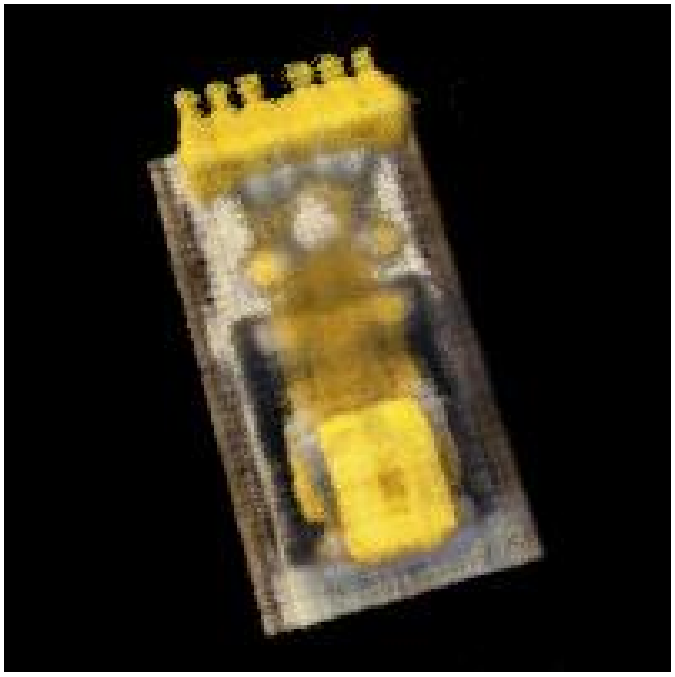
167th iteration of training



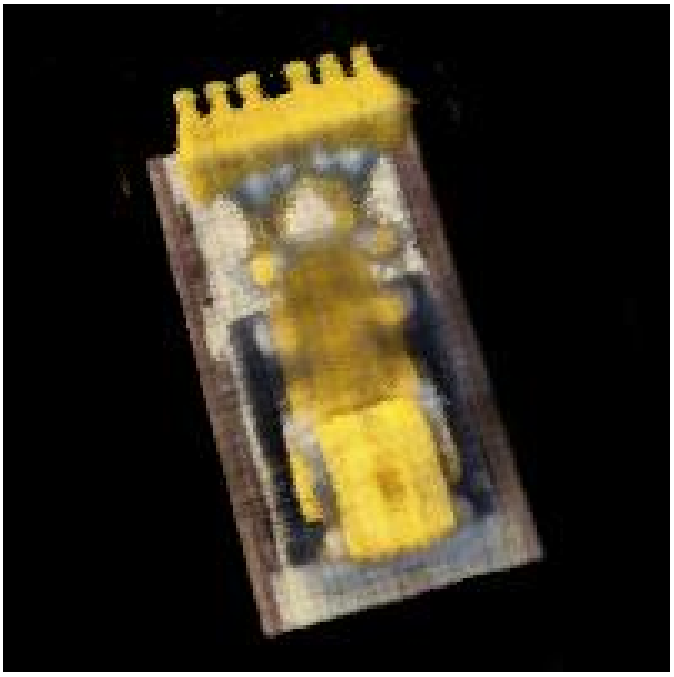
333rd iteration of training



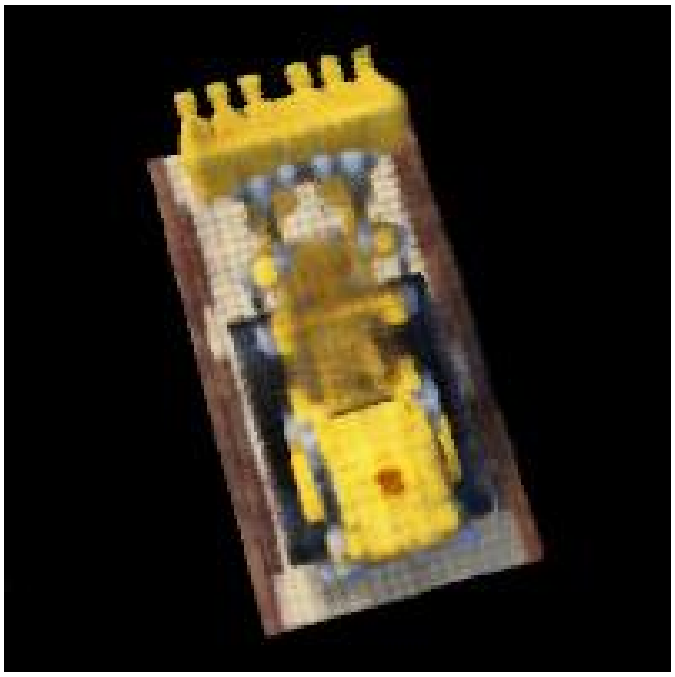
499th iteration of training



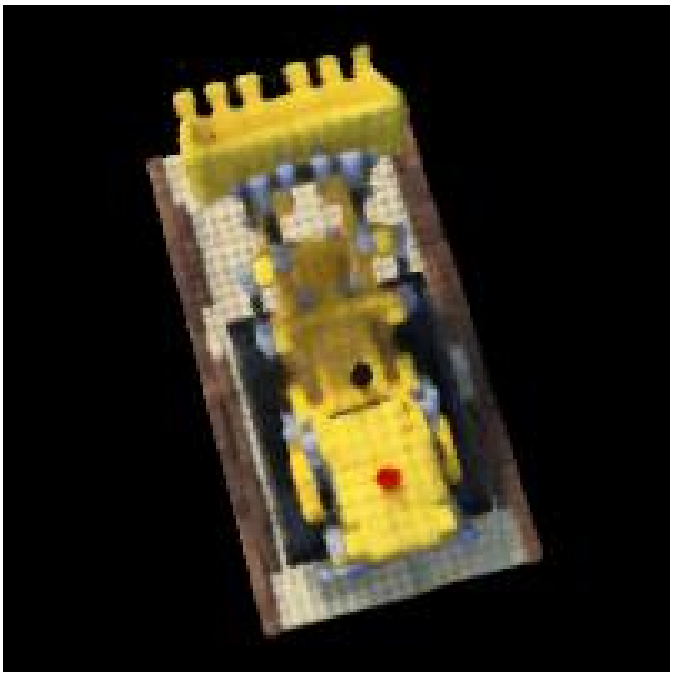
665th iteration of training



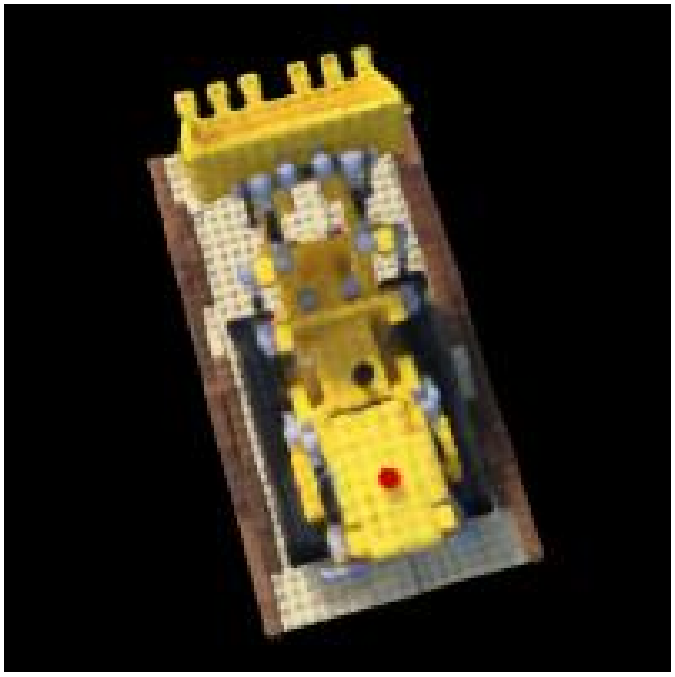
831st iteration of training



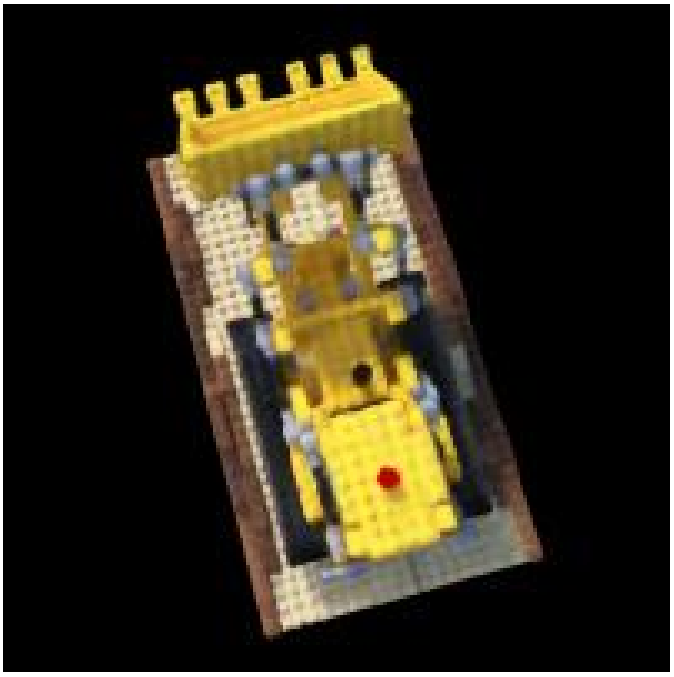
1667th iteration of training



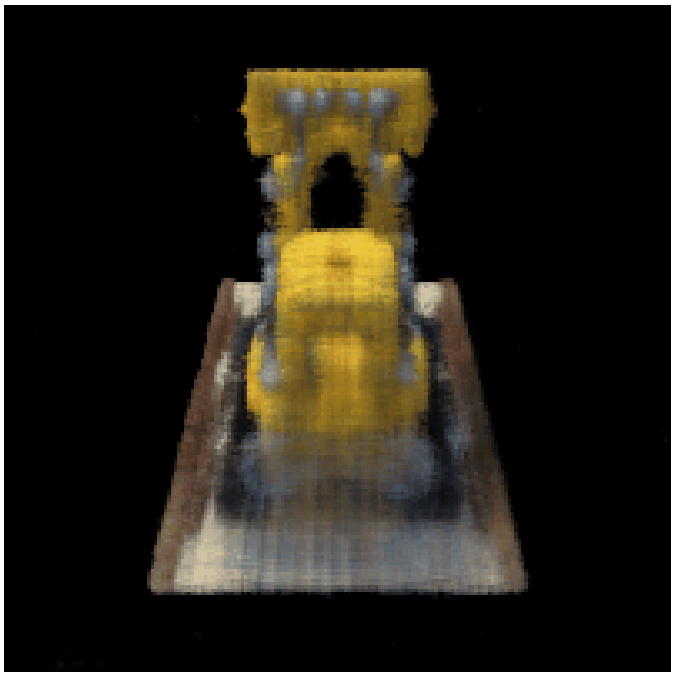
3333rd iteration of training



8831st iteration of training



10000th iteration of training



1000 iterations of training gif

10000 iterations of training gif