

CS83 Assignment 4: Neural Radiance Fields (NeRF) Implementation from Scratch

Mithun Rameshkumar

June 1, 2026

1 Introduction

This report documents the implementation of Neural Radiance Fields (NeRF) from scratch for novel view synthesis on the synthetic Blender `lego` scene. NeRF, introduced by Mildenhall et al. [1], represents a 3D scene implicitly as a continuous function parameterized by a multilayer perceptron (MLP): given a 3D position and a viewing direction, the network predicts a color and a volume density. Images are synthesized by querying the MLP at points sampled along camera rays and compositing the results with a differentiable volume rendering integral. Because the entire pipeline is differentiable, the scene representation is learned end-to-end by minimizing the photometric error against a set of posed input images.

The implementation covers the full NeRF pipeline:

- **Positional encoding** – a sinusoidal mapping of the low-dimensional inputs into a higher-dimensional space, enabling the MLP to represent high-frequency detail.
- **NeRF MLP** – an 8-input-channel coordinate network with a skip connection and a view-dependent color branch.
- **Ray generation and stratified sampling** – casting rays from camera intrinsics/extrinsics and sampling points along them.
- **Volume rendering** – converting raw density and color predictions into pixel colors, depth, and disparity.
- **Hierarchical sampling** – an importance-sampling stage that concentrates a second set of samples where the coarse network predicts mass.
- **End-to-end system** – the complete render/train loop that ties the components together.

The final model is trained on the 100 `lego` training views and evaluated on the 25 held-out test views, reporting Peak Signal-to-Noise Ratio (PSNR) against the ground-truth images.

2 Methodology

2.1 Architecture Overview

The full pipeline maps a posed camera to a rendered image in four stages:

1. For each pixel, a ray is generated in world space from the camera intrinsics K and the camera-to-world matrix.
2. A set of 3D points is sampled along each ray between the near and far bounds.
3. Each point (and the ray’s viewing direction) is positionally encoded and passed through the MLP, which outputs an RGB color and a density σ .
4. The per-point colors and densities are composited along the ray with the volume rendering integral to produce the final pixel color.

2.2 Part 1: Positional Encoding

A plain MLP is biased toward learning low-frequency functions, which produces over-smoothed renderings. To counteract this, each scalar input coordinate p is lifted into a higher-dimensional space via a bank of sinusoids at geometrically increasing frequencies:

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

The original input is retained alongside the encoding (`include_input`). Spatial positions use $L = 10$ frequency bands and viewing directions use $L = 4$. With the 3 raw coordinates plus $2L$ sinusoidal terms per coordinate, the encoded position has dimension $3 + 3 \cdot 2 \cdot 10 = 63$ and the encoded direction $3 + 3 \cdot 2 \cdot 4 = 27$. The encoder is implemented as a list of callables built once at construction time; a special index ($i = -1$) returns an identity mapping for ablation.

2.3 Part 2: NeRF MLP

The network follows the architecture of the original paper (Fig. 7 in [1]). The position branch is a stack of fully-connected layers of width $W = 256$ with ReLU activations. The 63-dimensional encoded position is concatenated back into the network at the skip layer (index 4), so that layer accepts $W + \text{input_ch}$ inputs rather than W . Density and color are produced by separate heads:

- **Density** (σ) is read directly from the position-branch features via `alpha_linear` ($W \rightarrow 1$). Because it does not depend on the viewing direction, the density of a point is view-invariant, as physical opacity should be.
- **Color** is produced by concatenating a W -dimensional feature vector with the encoded viewing direction, passing it through a half-width ($W/2 = 128$) layer, and projecting to 3 channels (`rgb_linear`). This view dependence lets the model represent specular and lighting effects.

Raw outputs are emitted without final activations; the sigmoid (color) and ReLU/softplus (density) are applied downstream in the volume renderer.

2.4 Part 3: Ray Generation and Sampling

For an image of size $H \times W$ and intrinsic matrix K , the camera-space direction of the ray through pixel (i, j) is

$$d_{\text{cam}} = \left(\frac{i - c_x}{f_x}, -\frac{j - c_y}{f_y}, -1 \right),$$

following NeRF’s convention that the camera looks down the $-z$ axis with y pointing up. The direction is rotated into world space by the 3×3 rotation block of the camera-to-world matrix, and all rays from a view share the camera origin (its translation column). Both PyTorch (`get_rays`) and NumPy (`get_rays_np`) versions are implemented; the latter is used to precompute rays for the whole dataset.

Along each ray, `N_samples` points are drawn by partitioning $[t_n, t_f]$ into evenly spaced bins and, during training, jittering one stratified sample uniformly within each bin (`perturb`). An NDC reparameterization (`ndc_rays`) is also implemented for forward-facing scenes, though the synthetic `lego` scene uses metric bounds $[2, 6]$ rather than NDC.

2.5 Part 4: Volume Rendering

Given raw network outputs $(\mathbf{c}_i, \sigma_i)$ at samples with depths z_i , the renderer (`raw2outputs`) implements the discretized volume rendering integral:

$$\alpha_i = 1 - \exp(-\sigma_i \delta_i), \quad T_i = \prod_{k < i} (1 - \alpha_k), \quad w_i = T_i \alpha_i,$$

where $\delta_i = z_{i+1} - z_i$ is the inter-sample distance (scaled by the ray direction norm so distances are metric). The final pixel color is the weighted sum $\hat{C} = \sum_i w_i \mathbf{c}_i$ with $\mathbf{c}_i = \text{sigmoid}(\text{raw rgb})$. The accumulated opacity $\sum_i w_i$, the expected depth $\sum_i w_i z_i$, and the disparity (inverse depth) are computed from the same weights. When a white background is assumed (as in the Blender renders), $(1 - \sum_i w_i)$ is added to the color so that fully-transparent rays composite to white.

2.6 Part 5: Hierarchical Sampling

Uniform sampling wastes capacity in empty space. NeRF therefore uses a two-network, coarse-to-fine scheme. The coarse network is evaluated at the `N_samples` stratified points; its rendering weights w_i define a piecewise-constant probability density along the ray. The function `sample_pdf` draws `N_importance` additional samples from this density by inverse-transform sampling: it normalizes the weights into a PDF, forms the CDF, and inverts uniform samples through it (with a deterministic ladder of uniforms at test time). The coarse and fine samples are merged, sorted, and passed to the fine network, whose composited output is the final prediction. Both networks are supervised, so the coarse network is forced to produce useful weights.

2.7 Part 6: Complete NeRF System

The `render` function generates or accepts rays, normalizes the viewing directions, packs ray origins/directions/bounds (and view directions) into a flat tensor, and dispatches it through `batchify_rays`, which splits the rays into memory-bounded chunks and renders each through `render_rays`. The training loop samples `N_rand` random rays per step (from a center crop for the first `precrop_iters` iterations to stabilize early training), renders them, and minimizes the mean-squared photometric error against the ground-truth pixels:

$$L = \|\hat{C}_{\text{coarse}} - C\|_2^2 + \|\hat{C}_{\text{fine}} - C\|_2^2.$$

Both the coarse and fine renderings are penalized, matching the original formulation.

3 Dataset and Preprocessing

The synthetic `lego` scene from the NeRF Blender dataset consists of 138 renders of a toy bulldozer on a transparent background, each 800×800 with a known camera pose. The standard split assigns 100 images to training, 13 to validation, and 25 to test. Each frame is loaded with its alpha channel and composited over a white background (`white_bkgd`), and images are downsampled to 400×400 (`half_res`) to keep training within an 8-hour GPU budget. Camera intrinsics are derived from the dataset’s field-of-view, giving a focal length of 555.6 px at half resolution. Scene bounds are fixed at `near = 2` and `far = 6`.

4 Training Configuration

- **Network:** coarse and fine MLPs, depth 8 (`netdepth = 4` position layers per branch with the skip configuration), width 256, view dependence enabled.
- **Encoding:** `multires = 10` (position), `multires_views = 4` (direction).
- **Sampling:** `N_samples = 64` coarse, `N_importance = 128` fine.
- **Rays per step:** `N_rand = 1024`.
- **Optimizer:** Adam, initial learning rate 5×10^{-4} , exponential decay with `lrate_decay = 500k` steps.
- **Center crop:** `precrop_iters = 500`, `precrop_frac = 0.5`.
- **Iterations:** 200,000.
- **Hardware:** $1 \times$ NVIDIA RTX A5500 (24 GB), submitted via Slurm.
- **Wall-clock:** ≈ 5 h 41 m.

5 Results

5.1 Training Curves

Figure 1 shows the photometric MSE loss and the (per-batch) PSNR over the 200,000-iteration run. The loss falls by roughly two orders of magnitude; the sharp early improvement at iteration 500 coincides with the end of the center-crop warmup, after which the model trains on rays from the full image. PSNR rises quickly through the first $\sim 15,000$ iterations and then improves slowly, consistent with the model first recovering coarse geometry and later refining fine texture.

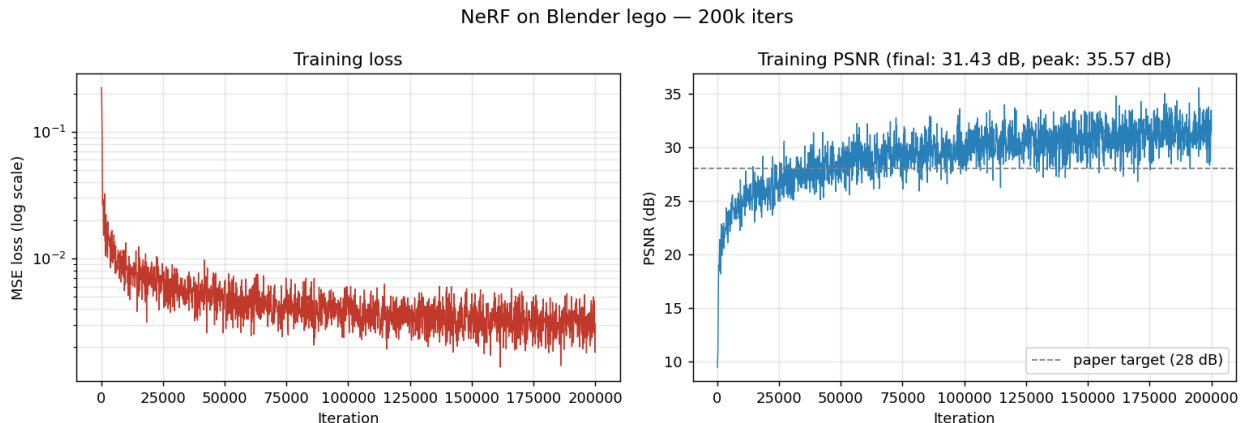


Figure 1: Training MSE loss (left, log scale) and per-batch PSNR (right) over 200,000 iterations. The dashed line marks the 28 dB target. The per-batch PSNR is noisy because each point is measured on a single random 1,024-ray batch.

Iteration	Loss	Batch PSNR (dB)
100	0.2224	9.45
500 (crop ends)	0.0514	16.15
50,000	0.0049	~ 29
195,000 (peak)	0.0021	35.57
200,000 (final)	0.0031	31.43

Table 1: Selected training-batch loss and PSNR values.

5.2 Test-Set Novel View Synthesis Quality

The per-batch PSNR printed during training is measured on a single random ray batch and is therefore an optimistic, high-variance estimate. To characterize generalization, the trained model was used to render all 25 held-out test poses, and PSNR was computed against the ground-truth images. Results are summarized in Table 2.

Metric (25 test views)	PSNR (dB)
Mean	30.71
Median	30.74
Minimum	27.57
Maximum	32.66
Std. dev.	1.22

Table 2: Test-set PSNR over the 25 held-out novel views of the `lego` scene.

The mean test PSNR of **30.71 dB** exceeds the assignment’s target of 28 dB, and the low standard deviation (1.22 dB) indicates the quality is consistent across viewpoints rather than driven by a few easy views.

5.3 Qualitative Novel Views

Figures 2 and 3 compare rendered novel views against ground truth for the best and worst test images by PSNR. Even the worst case reproduces the overall geometry, color, and shading of the bulldozer faithfully; the residual error is concentrated in the highest-frequency regions (the tread detail and thin shovel arms) and in mild blur of fine edges.



Figure 2: Best test view (image 020, 32.66 dB): rendered novel view (left) vs. ground truth (right).



Figure 3: Worst test view (image 010, 27.57 dB): rendered novel view (left) vs. ground truth (right). Fine tread/edge detail is slightly blurred, but global structure and shading are correct.

The model also learns accurate scene geometry, not merely view interpolation. Figure 4 shows a disparity (inverse-depth) frame from the spiral render: the bulldozer’s 3D silhouette is clearly delineated against the (far) background, with nearer surfaces brighter than farther ones. A full 360° spiral flythrough is provided as `spiral_200000_rgb.mp4`.

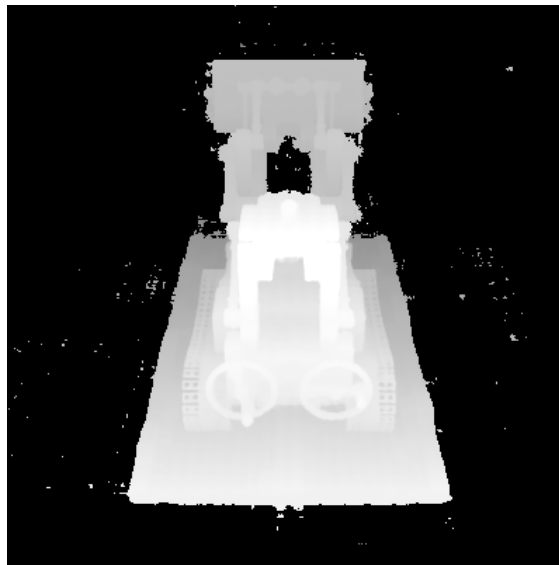


Figure 4: Disparity (inverse depth) for a spiral-path frame. Brighter = closer to the camera. The recovered depth confirms the network learned consistent 3D geometry.

6 Comparison with the Original NeRF Paper

Architecturally, the implementation matches the original NeRF: identical positional encoding ($L = 10/4$), the same 256-wide coordinate MLP with a single skip connection, a view-dependent color branch, hierarchical coarse/fine sampling, and the same volume rendering integral. The main differences are deliberate, compute-motivated simplifications, summarized in Table 3.

Aspect	This implementation	Original paper [1]
Resolution	400×400 (half res)	800×800
Coarse/fine samples	64 / 128	64 / 128
Iterations	200k	$\sim 200\text{--}300\text{k}$
Rays per step	1,024	4,096
Reported <code>lego</code> PSNR	30.71 (ours, test)	32.54 (paper, 800×800)

Table 3: Comparison between this implementation and the original NeRF on the `lego` scene.

The ~ 1.8 dB gap to the paper’s reported 32.54 dB is consistent with training at half resolution and one quarter of the ray batch size; both reduce the effective amount of supervision per step and limit the highest recoverable frequencies. The qualitative results show no structural failures, indicating the gap is one of fine-detail sharpness rather than a correctness issue.

7 Discussion

Implementation correctness. The smooth, monotone loss curve, the recovery of consistent depth (Fig. 4), and a 30.71 dB mean test PSNR together indicate that all six components are correct and mutually consistent. An error in any single stage – the encoding, the skip connection, the ray geometry, the transmittance accumulation, or the importance sampler – would have manifested either as a failure to converge or as visibly wrong geometry.

Effect of positional encoding. The encoding is the single component most responsible for sharp output: without it, the MLP’s low-frequency bias would render the bulldozer as a smooth blob. The retained raw input and the $L = 10$ band schedule give the network the basis functions needed to represent the tread and edge detail visible in Figure 2.

Effect of hierarchical sampling. Importance sampling concentrates the 128 fine samples near surfaces, which is what allows a budget of only $64 + 128$ samples per ray to resolve a thin, intricate object. Uniform sampling at the same total budget would spend most samples in the empty space between the camera and the bulldozer.

Challenges encountered.

- *Metric interpretation.* The per-batch PSNR reported during training (peaking at 35.57 dB) overstates quality because it is measured on a single random ray batch. Computing PSNR on the full held-out test set gave the honest 30.71 dB figure used for evaluation.

Future improvements. More recent work supersedes vanilla NeRF on both speed and quality: Instant-NGP’s hash encoding [2] trains the same scene in seconds, and 3D Gaussian Splatting [3] achieves real-time rendering. Adding a coarse-to-fine frequency annealing schedule for the positional encoding would also reduce early-training high-frequency artifacts.

8 Conclusion

A complete NeRF pipeline was implemented from scratch, covering positional encoding, the view-dependent MLP with a skip connection, ray generation and stratified sampling, the differentiable volume rendering integral, hierarchical coarse-to-fine importance sampling, and the end-to-end render/train system. Trained on the synthetic **lego** scene for 200,000 iterations, the model achieves a mean test-set PSNR of 30.71 dB across 25 held-out novel views, exceeding the 28 dB target and approaching the original paper’s 32.54 dB despite training at half resolution with one quarter of the ray batch size. Rendered novel views and recovered depth maps confirm that the model learned a faithful, view-consistent 3D representation of the scene.

References

- [1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, R. Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *ECCV*, 2020.
- [2] T. Müller, A. Evans, C. Schied, A. Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Transactions on Graphics (SIGGRAPH)*, 2022.
- [3] B. Kerbl, G. Kopanas, T. Leimkühler, G. Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics (SIGGRAPH)*, 2023.