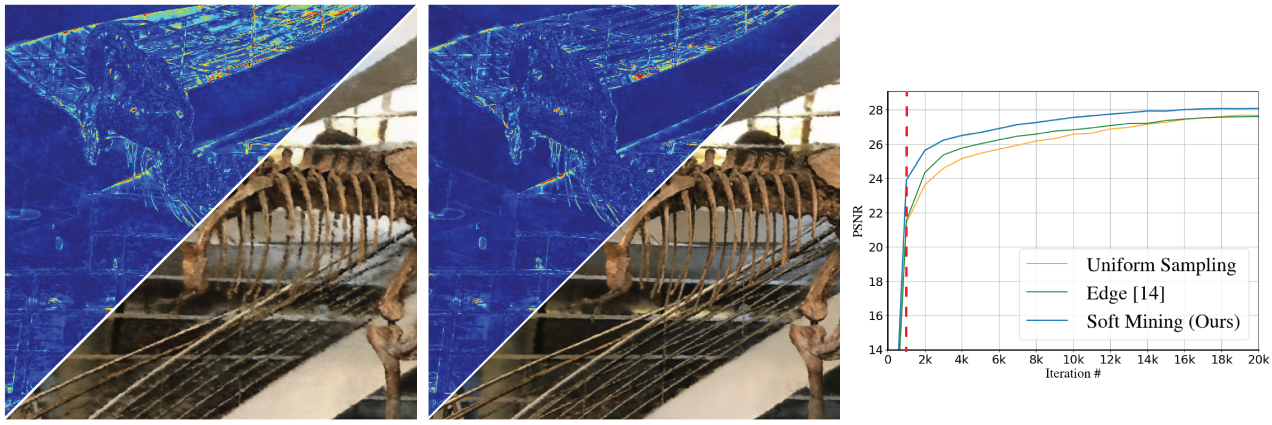
Accelerating Neural Field Training via Soft Mining

Shakiba Kheradmand[1], Daniel Rebain[1], Gopal Sharma[1], Hossam Isack[2],
Abhishek Kar[2], Andrea Tagliasacchi[3, 4, 5], Kwang Moo Yi[1]
[1] University of British Columbia, [2] Google Research,
[3] Google DeepMind, [4] Simon Fraser University, [5] University of Toronto



Uniform Sampling (1k iterations) Soft Mining (Ours) (1k iterations) PSNR Over Iterations

Figure 1. **Teaser:** we introduce "soft mining" to accelerate neural field training. When applied to Neural Radiance Field (NeRF) training, our method significantly improves convergence. We visualize the error maps (blue denotes low error and red denotes high error) and the rendered novel views for uniform sampling and our method. We plot the convergence showing the Peak Signal-to-Noise Ratio (PSNR) for the corresponding scene. We render both images at 1k iterations of training, specified by the red dashed line in the **(right)** graph. Our method achieves the same PSNR significantly faster than the baselines.

Abstract

We present an approach to accelerate Neural Field training by efficiently selecting sampling locations. While Neural Fields have recently become popular, it is often trained by uniformly sampling the training domain, or through handcrafted heuristics. We show that improved convergence and final training quality can be achieved by a soft mining technique based on importance sampling: rather than either considering or ignoring a pixel completely, we weigh the corresponding loss by a scalar. To implement our idea we use Langevin Monte-Carlo sampling. We show that by doing so, regions with higher error are being selected more frequently, leading to more than 2x improvement in convergence speed. The code and related resources for this study are publicly available at project page.

1. Introduction

Neural fields [45] have recently become popular due to their versatile nature, and their ease of integration with popular workloads like novel view synthesis [22]. Neural fields map input coordinates to output values and are typically implemented as variants of multi-layer perceptrons (MLP) [19, 35, 39]. They have demonstrated impressive capabilities in representing signals for 1D audio [35], 2D images [23, 39], 3D shapes [22, 26], and 4D light fields [37, 48]. Beyond modeling and compressing signals, they have also been utilized to simulate physics [6, 27].

While neural fields provide impressive results, training them can be lengthy, e.g. on modern GPUs training the original Neural Radiance Field (NeRF) [22] for a single scene can take hours. Researchers have since improved the training process through alterations to network architectures [9, 23], and through better loss functions [2].

An angle that none of the papers above consider is how *training batches* are formed. That is, which pixels in the 2D image, or which rays in the radiance field are used for each optimization step. These methods typically rely on simple *uniform* sampling, which may lead to sub-optimal training

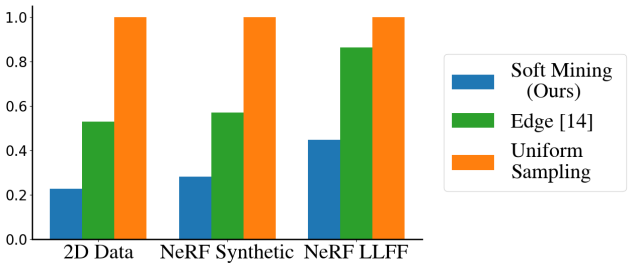


Figure 2. **Convergence**: we show the relative number of iterations compared to uniform sampling required to reach PSNR value of 35 dB for 2D image fitting, 30 dB for the NeRF Synthetic dataset, and 25 dB for the NeRF LLFF dataset. Our method requires significantly less number of iterations, specifically less than half of what is required with uniform sampling.

performance. For example, as shown for the NeRF training example in Figure 1, if the signal has smooth regions such as walls, we can expect diminishing returns when sampling from those smooth areas as training progresses. There have been some very recent attempts to address this problem either through a heuristic that focuses training to image edges [14], or by analyzing images through quad-trees [50]. However, these methods are hand-designed for NeRF and do not generalize to other neural field training. Moreover, the gains that these methods provide are marginal, especially with real-world data, as we will show empirically.

In this work, we propose a principled method to improve the convergence of neural field training by improving the sampling mechanisms. Akin to core research in image classification that introduced the concept of hard mining [32, 33], we propose to focus on 'hard' samples within a neural field workload. However, we empirically discovered that straightforward hard mining does not improve training. We thus re-formulate neural field training as importance sampling and derive a relaxed 'soft' hard mining formulation, analogous to the soft mining heuristic introduced for metric learning [41]. To implement our idea, inspired by the rich literature on Bayesian optimization for efficient sampling of distributions [3, 12, 43], we opt to use Langevin Monte Carlo (LMC) sampling [5, 25].

To verify the efficacy of our method, we apply it to two common use cases of neural fields: 2D image fitting and NeRF. As shown in Fig. 2, our method at least *doubles* the convergence speed for all tasks. Compared to the edge-based heuristic [14] for NeRF, our method approximately doubles the convergence speed for both synthetic and real-world data.

2. Related Works

Neural fields have gained substantial popularity as a representation method for various types of signals in a wide variety of applications [46]. This paradigm was first popularized as a way of representing 3D scenes and objects, beginning with Scene Representation Networks [34] in the form of learnable ray marching, which was soon followed by Neural Radiance Fields [22] and Implicit Differentiable Renderer [47], which adopted volumetric and distance field representations respectively. Initially, work on neural scene representations focused on architectures where the field coordinate is a spatial position, but later works generalized this to cases where the parameters of a viewing ray are used to define the field [30, 36]. These methods share a common photometric reconstruction strategy for supervision in which the network is trained to reproduce pixel values of the training images when rendered. Due to the computational expense of rendering, this supervision is implemented as a stochastic approximation in which mini-batches of pixels are sampled from the training images to be used as supervision at each step. It is this pixel sampling process that we aim to improve, so to achieve better training convergence and higher accuracy.

Accelerating neural fields. Neural scene representations, particularly those based on volume rendering, quickly gained a reputation for being very computationally expensive and therefore slow to train. Consequently, a number of methods have been proposed for accelerating both training and inference by modifying the model architecture to be less expensive. The majority of these approaches have tried to achieve their speed-ups by reducing the number of times the underlying field needs to be evaluated [20, 23, 31, 48], by altering the memory-computation trade-off of the neural architecture [15, 28, 29, 31, 38], and/or by optimizing their implementations to take maximum advantage of acceleration hardware with efficient compressed field representations [7–10, 13, 23]. While these methods are largely aligned with our goal of improving efficiency, and thereby quality achievable within a given compute budget, their methods are *complementary* to ours, which is compatible with a wide variety of efficient architectures. We note also that none of these methods focus on choosing which rays (queries) to use for training, which is the primary focus of our work.

Efficient selection of queries. Some prior works have explored modified ray sampling schemes for training NeRF models. Gai et al. [14], which we refer to as Edge, samples rays based on detected edges in training images, taking advantage of the fact that rendering error tends to be higher in these regions. Zhang et al. [50] adopts a strategy that incorporates a prior probability derived from local color variance, and further tracks photometric error throughout training with an adaptive quadtree structure, allocating more samples to regions with higher error. Unlike these hand-crafted techniques, our method seeks to improve training by allocating samples in a principled manner, and in a way that is not tightly coupled to a specific task. By doing so we

show that a significant speed up in convergence is achievable compared to existing methods.

3. Method

Let us start by formalizing neural field training (Sec. 3.1). We will then introduce our soft mining approach (Sec. 3.2) as well as how to create batches effectively with minimal overhead via LMC sampling (Sec. 3.3).

3.1. Neural field training

A neural field f_{ψ} with learnable parameters ψ defines a mapping from a bounded set of coordinates $\mathcal{R} \subset \mathbb{R}^D$ to outputs $\mathcal{O} \in \mathbb{R}^F$ as $f_{\psi} : \mathbb{R}^D \rightarrow \mathbb{R}^F$, where for example $f_{\psi} : \mathbb{R}^3 \rightarrow \mathbb{R}^1$ maps positions to signed distance in [26, 35], and $f_{\psi} : \mathbb{R}^5 \rightarrow \mathbb{R}^4$ maps positions to view-dependent radiance and density in [22]. These neural fields are typically trained with a loss function that is associated with the error that the neural field function is making while predicting a ground-truth signal $f_{\text{gt}}(\mathbf{x})$. Formally, we define an error function that takes a neural field $f_{\psi}(\mathbf{x})$ at the coordinates $\mathbf{x}$ and outputs a value representing some disparity metric between the prediction and ground truth $f_{\text{gt}}(\mathbf{x})$ as

$$\text{err}(\mathbf{x}) = \text{err}(f_{\psi}(\mathbf{x}), f_{\text{gt}}(\mathbf{x})) \in \mathbb{R}. \tag{1}$$

We can then write the loss function that is minimized for training to be the Monte Carlo estimate of the expectation of this error $\text{err}(\mathbf{x})$:

$$\begin{aligned}\mathcal{L} &= \frac{1}{N}\sum_{n=1}^{N} \text{err}(\mathbf{x}_n) \approx \mathbb{E}_{\mathbf{x}\sim P(\mathbf{x})}\left[\text{err}(\mathbf{x})\right] \\ &= \int \text{err}(\mathbf{x}) P(\mathbf{x})\, d\mathbf{x}\,, \end{aligned} \tag{2}$$

where N is the number of training samples in a batch and P is the distribution of the sampled data points $\mathbf{x}_n$. In (2), it is important to note that P affects the outcome of the integral and the expectation, and changes in the sampling scheme effectively result in changes in the loss—naively choosing a batch construction scheme can therefore be harmful and requires care. Typically, a *uniform* distribution is employed as P for most neural field applications; *e.g.*, in Mildenhall et al. [22]. Let us now discuss how to perform (soft) hard mining with importance sampling and how it tightly relates to the original objective in (2).

3.2. Soft mining with importance sampling

We first start by introducing *importance sampling* for neural field training. To allow for different strategies to be used for batch construction in (2), we introduce an importance distribution $Q(\mathbf{x})$ and create training batches as:

$$\begin{aligned}\int \text{err}(\mathbf{x}) P(\mathbf{x})\, d\mathbf{x} &= \int \frac{\text{err}(\mathbf{x})P(\mathbf{x})}{Q(\mathbf{x})} Q(\mathbf{x})\, d\mathbf{x} \\ &= \mathbb{E}_{\mathbf{x}\sim Q(\mathbf{x})}\left[\frac{\text{err}(\mathbf{x})P(\mathbf{x})}{Q(\mathbf{x})}\right]. \end{aligned} \tag{3}$$

When P is a uniform distribution, which is reasonable to assume given that we typically want the signal to be well-represented everywhere, the Probability Density Function (PDF) of the uniform distribution is constant, and we can remove $P(\mathbf{x})$ from (3) and rewrite it as:

$$\mathbb{E}_{\mathbf{x}\sim Q(\mathbf{x})}\left[\frac{\text{err}(\mathbf{x})}{Q(\mathbf{x})}\right]. \tag{4}$$

We can thus further rewrite (2) as:

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N} \frac{\text{err}(\mathbf{x}_n)}{Q(\mathbf{x}_n)}, \quad \text{where} \;\; \mathbf{x}_n \sim Q(\mathbf{x}). \tag{5}$$

Note that training with (5) is impractical for an arbitrary distribution $Q(\mathbf{x})$ as one must differentiate through the sampling process. Hence, we simply add a stop gradient operator $\text{sg}(\cdot)$ as in van den Oord et al. [40]:

$$\mathcal{L} \approx \mathbb{E}_{\mathbf{x}\sim \text{sg}(Q(\mathbf{x}))}\left[\frac{\text{err}(\mathbf{x})}{\text{sg}(Q(\mathbf{x}))}\right]. \tag{6}$$

Note that this effectively amounts to simple sample reweighing. Without affecting the initial training objective, equation (6) now allows us to focus our training efforts by choosing an appropriate $Q(\mathbf{x})$. Note also that this loss scaling is *not accounted for* in methods that employ heuristics for constructing training batches [14, 50].

To focus our training on regions with high error we simply define $Q(\mathbf{x})$ to be *proportional* to $\text{err}(\mathbf{x})$. Among various possibilities, for our experiments, we opt for squared error for $\text{err}(\mathbf{x})$, which is the typical choice for training neural fields, and set $Q(\mathbf{x})$ to be the L1 norm of the error, so to avoid focusing too much on outliers:

$$\text{err}(\mathbf{x}) = \|f_{\psi}(\mathbf{x}) - f_{\text{gt}}(\mathbf{x})\|_2^2, \tag{7}$$

$$Q(\mathbf{x}) = \|f_{\psi}(\mathbf{x}) - f_{\text{gt}}(\mathbf{x})\|_1. \tag{8}$$

Soft mining. While importance sampling mathematically allows us to optimize the original objective with potentially more effective samples at hard regions, our experiments in Sec. 4.3 show that purely relying on it does not improve performance drastically. We also show that ignoring the importance weight $Q(\mathbf{x})^{-1}$, which is equivalent to the commonly-used hard mining [32, 33], is sub-optimal as well. While more emphasis should be given to the samples that are 'hard' to learn, focusing exclusively on the hard

samples biases the training too far away from the original training objective. To address this issue, we propose an alternative that strikes a middle-ground, and write:

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}\left[\frac{\mathrm{err}(\mathbf{x}_n)}{\mathrm{sg}(Q(\mathbf{x}_n))^{\alpha}}\right], \quad \text{where } \alpha \in [0,1], \tag{9}$$

where α controls the 'softness' of the mining, with α=0 corresponding to (pure) hard mining, and α=1 corresponding to (pure) importance sampling. In our experiments we typically utilize $\alpha \in [0.6, 0.8]$, thus behaving as soft mining. We ablate our choice in Sec. 4.3.

3.3. Sampling via Langevin Monte Carlo

To implement our method we require that we can sample from an arbitrary distribution $Q(\mathbf{x})$. This is non-trivial, and we thus rely on Markov Chain Monte Carlo (MCMC). Among the family of MCMC methods, we specifically use Langevin Monte Carlo (LMC) [5, 25] thanks to its simplicity and effectiveness. Its *deterministic* nature, driven by the gradient of the log posterior distribution, steers the exploration effectively towards regions of higher probability. Meanwhile, its *stochastic* nature facilitates a comprehensive exploration of the parameter landscape, aiding in evading local minima and promoting convergence to the target distribution. Formally, to sample from $Q(\mathbf{x})$, denoting a sample location at sampling step t as $\mathbf{x}_t$ we write:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + a\nabla \log Q(\mathbf{x}_t) + b\boldsymbol{\eta}_{t+1}, \tag{10}$$

where a>0 is a hyperparameter defining the step size for the gradient-based walks, and b>0 is a hyperparameter defining the step size for the random walk $\boldsymbol{\eta}_{t+1}\sim\mathcal{N}(0,\mathbf{1})$. Note that for simplicity in notation, we have folded the two hyperparameters associated with the random walk in Brosse et al. [5, Eq. 2] into a single parameter b. Critically, note here that (10) depends only on the local gradient of the sampling distribution $Q(\mathbf{x})$ and random noise $\boldsymbol{\eta}_{t+1}$. Because the method is local, we can perform sampling with minimal overhead as we already compute the backward pass for training our neural field.

Sample (re-)initialization. While LMC eventually converges to the desired distribution, it is well known that MCMC methods require careful (re-)initialization for effective sampling [16]. We first initialize the sampling distribution to be uniform over the domain of interest as $\mathbf{x}_0\sim\mathcal{U}(\mathcal{R})$. We further re-initialize samples that either move out of $\mathcal{R}$ or have too low error value causing samples to get 'stuck'. We use uniform sampling as well as edge-based sampling for 2D workloads.

Warming up soft mining. We empirically noticed that in very early training iterations ($\leq$1k iterations) LMC requires warm-up time, which is commonly the case for MCMC methods [16]. This makes applying the corrections in (9) unreliable as our samples are not yet exactly following $Q(\mathbf{x})$. We thus start with α=0, *i.e.*, no correction, then linearly increase it to the desired α value at 1k iterations.

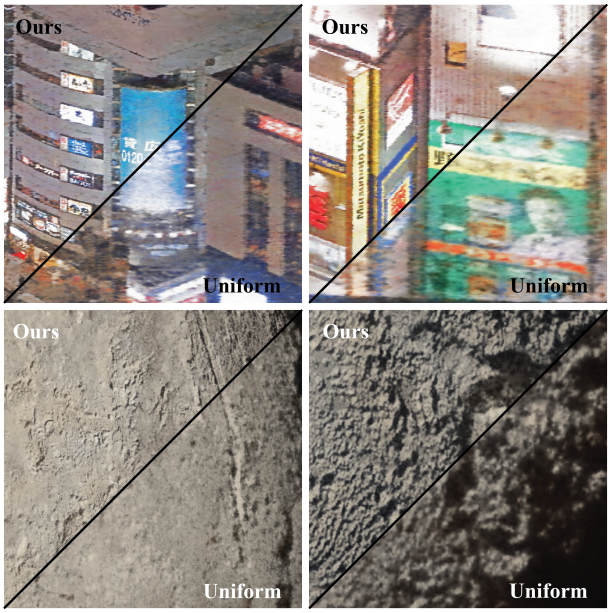
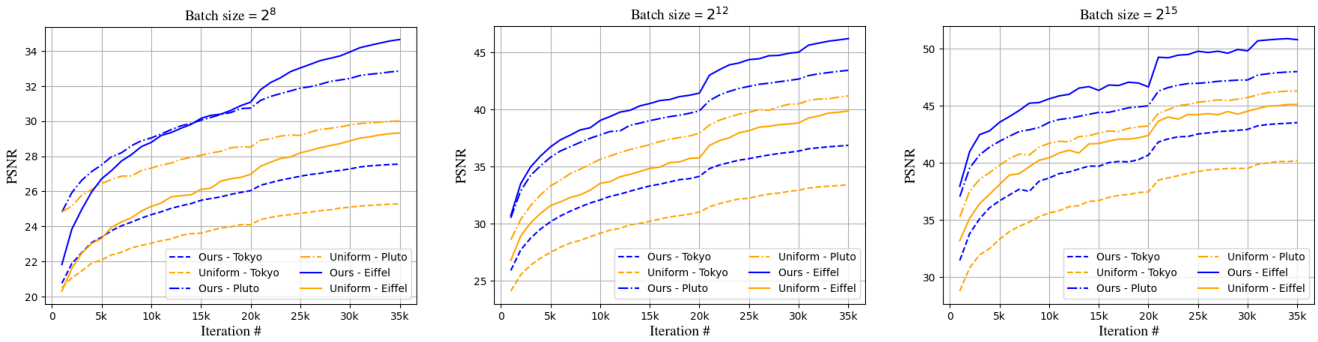


Figure 3. **2D image fitting examples**: We show 2D image fitting example for our method and uniform sampling for **(top)** the two regions on the Tokyo image and **(bottom)** the two regions on the Pluto image. The **(left)** column shows results for the batch size of 256 trained for 10k iterations and the **(right)** column shows the result for the batch size of 4096 trained for 1k iterations. As shown, our results are sharper, especially noticeable around the texts and finer details.

Alternative: multinomial sampling. For many neural fields use cases [22, 35], the modeling space is discrete – *e.g.* pixels. In this case, $Q(\mathbf{x})$ would be a multinomial distribution that could be explicitly modeled and sampled from. While we show in Sec. 4 that this approach indeed provides an improvement, it is *computationally impractical*. To use multinomial sampling, one needs to do a forward pass of all data points to build a probability density function, which is computationally expensive. Even a naive strategy to prevent these forward passes, such as bookkeeping a moving average of error can be costly in a large dataset. Hence an alternative strategy, such as those based on Markov Chain Monte Carlo (MCMC) is required.

4. Results

To validate the effectiveness of our method we focus on two popular applications: 2D image fitting and Neural Radiance Fields (NeRF). We first present results for these two applications and then provide ablation studies. We note that in all of our experiments, to account for the randomness of neural field training, all reported results are the average out-



	batch size	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}
Pluto	Uniform	30.01	32.34	35.27	38.50	41.19	43.22	45.00	46.29	47.47	48.28	48.79
	Ours	**32.85**	**35.60**	**38.55**	**41.16**	**43.42**	**45.33**	**46.88**	**47.99**	**48.88**	**49.43**	**49.97**
Eiffel	Uniform	29.32	31.96	34.59	37.21	39.87	42.11	44.07	45.12	45.45	46.00	46.14
	Ours	**34.65**	**37.81**	**40.90**	**44.00**	**46.21**	**48.31**	**49.62**	**50.79**	**51.72**	**52.33**	**52.67**
Tokyo	Uniform	25.30	26.95	28.87	30.95	33.41	36.07	38.56	40.19	41.46	42.15	42.66
	Ours	**27.55**	**29.51**	**31.73**	**34.18**	**36.86**	**39.39**	**41.57**	**43.52**	**45.05**	**45.99**	**46.62**

Figure 4. **Convergence – image fitting:** we report the PSNR values for the Pluto, Eiffel Tower, and Tokyo images, with different batch sizes, for both our method and uniform sampling. We also show the convergence graphs for two representative batch sizes. Regardless of the batch size, our method provides faster convergence.

comes of three runs. We will release the code to ensure full reproducibility.

4.1. 2D image fitting

We first apply our method to the task of fitting a neural field to an image, that is, the task of image memorization. We compare our method to uniform sampling, with the same Instant-NGP [23] backbone. We implement our framework based on the official Instant-NGP implementation [23, 24].

Dataset and metrics. We use three high-resolution images for evaluation: Eiffel Tower (3024×4032), Pluto (8000×8000) and Tokyo (6144×2324). Pluto image is a high-resolution image of Pluto.[1] The latter two were used to benchmark methods in previous works [23, 44]. We compare each method using PSNR.

Hyperparameters. We keep the same hyperparameter setting for all our 2D image-fitting experiments. We use a learning rate of 0.01 and a multi-step learning rate scheduler (decaying at 20k and 30k iterations) following the base implementation [23]. We set α=0.6 in (9). We leverage image edges for re-initialization (see Sec. 3.3). We execute Sobel edge detection and normalize the edge scores to turn them into a probability distribution. We then randomly choose pixels from this distribution to re-initialize our LMC samples, those that are the bottom 10% of the LMC sampling pool according to $Q(\mathbf{x})$. We further keep 10% of our training batch to be sampled uniformly to avoid completely ignoring some pixels. Finally, for (10), we choose a=$1e-5$ and b=$1e-3$ via hyperparameter search, which we found to work well for most images.

Results. As shown in Fig. 3, our method provides significantly higher reconstruction quality for the same number of iterations, *i.e.*, our method converges faster. We further show convergence graphs in Fig. 4 along with the quantitative results for different batch size configurations. Notably, our method not only leads to faster convergence but also to higher PSNR at the end. We emphasize once more that the only difference between the baselines is the sampling strategy for constructing batches. Yet, there is a significant gap, demonstrating the importance of choosing which points to sample. As depicted in Fig. 2, we converge almost four times faster than uniform sampling to a PSNR of 35dB (averaged over all three images and all batch sizes)—it takes ≈2,400 iterations with our method and ≈10,600 with uniform sampling.

4.2. Neural radiance fields

We further experiment on learning NeRF [22], arguably one of the most popular applications for neural fields. NeRF [22] takes a 3D position and a direction vector and outputs radiance and density values used to volume render an image. NeRF training involves sampling light rays that correspond to each pixel to construct training batches, which are then used to train neural fields with a pixel-wise color reconstruction loss. We defer the exact details of NeRF to Mildenhall et al. [22], as here we are interested

[1] Image courtesy of NASA's Photojournal (Image ID: PIA19952). The image is in the public domain.

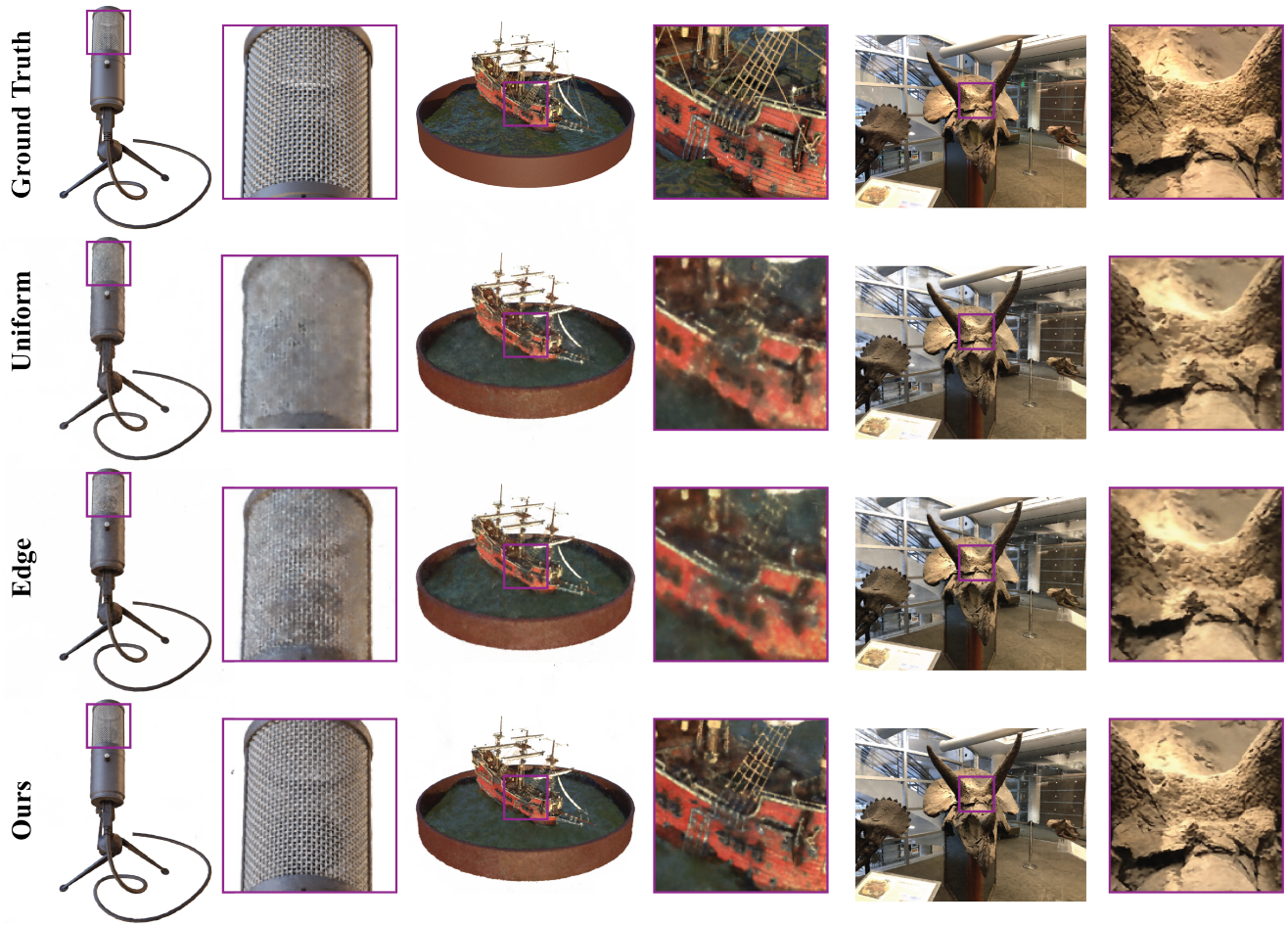


Figure 5. **Qualitative examples – NeRF**: we show example rendering for each sampling method after 1k training iterations. Our renderings are significantly sharper than other methods, demonstrating faster convergence. Best viewed when zoomed in.

only in evaluating how our sampling of the light rays for constructing batches helps convergence. We compare our method to two baselines: uniform sampling, and Edge [14]. As no public implementation exists, we re-implement Gai et al. [14] faithfully following the paper.

Dataset and metrics. We experiment with the NeRF Synthetic dataset [22] consisting of 8 object-centric scenes with white background, as well as the LLFF dataset [21, 22] consisting of 8 real-world forward-facing scenes. For the NeRF framework, we use NerfAcc [18], a popular repository which is known to closely reproduce the results of Instant-NGP [23] (as the public implementation of [23] does not reproduce the results in the paper for NeRF experiments). We keep all aspects the same except for the sampling process. We use the standard image quality metrics: PSNR, SSIM [42], and LPIPS [49].

Hyperparameters. We train each method with a learning rate of 0.01, a cosine annealing learning rate scheduler. We use α=0.6 for NeRF Synthetic scenes and α=0.8 for the LLFF dataset, as we found that real and synthetic scenes exhibit different characteristics. We keep all other hyperparameters the same for all experiments. We use the same sample re-initialization as in Sec. 4.1, and use a=2e1 and $b = 2e - 2$ for (10).

Computation time. Before we dive into the results, we first measure the computation time with and without our method with the NeRF application. We measure the computation time on a system equipped with an Intel Core i7-11700K @ 2.50GHz CPU and NVIDIA GeForce RTX 3090 GPU, with a batch size of 300 rays, with three different runs, each running 20k iterations. With our pure PyTorch implementation, running 20k iterations takes 248 seconds, whereas with our method 257 seconds. This amounts to a less than 4% increase in processing time. Given the more-than-twice increase in convergence speed, we argue that the 4% increase is negligible. Furthermore, examination with a GPU profiler reveals that the majority of the increase is due to CPU overhead in the backward pass, suggesting that an implementation using a pre-compiled graph such as TorchScript [1] or JAX [4] would eliminate this slowdown. In other words, a more optimized implementation should be able to reduce

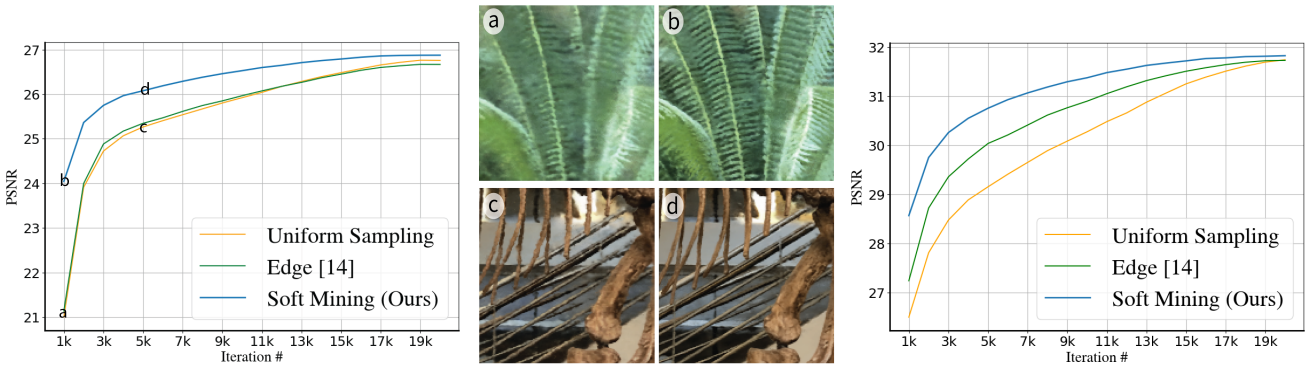


Figure 6. **Convergence – NeRF**: PSNR vs. the number of iterations for the test views of (left) the LLFF dataset and (right) the NeRF Synthetic dataset, with the (middle) zoomed-in rendering (fern and trex scenes) for selected iterations. Our method significantly speeds up convergence for both datasets, whereas edge-based heuristic [14] only works well for the synthetic dataset.

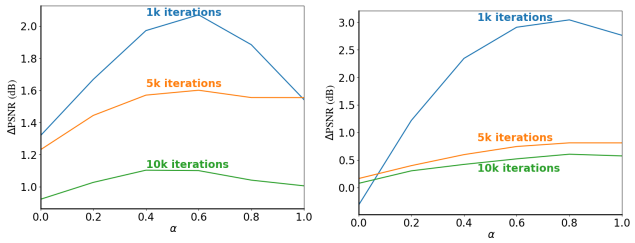
	1k iterations (PSNR / SSIM / LPIPS)			5k iterations (PSNR / SSIM / LPIPS)			10k iterations (PSNR / SSIM / LPIPS)		
	Uniform	Edge [14]	Ours	Uniform	Edge [14]	Ours	Uniform	Edge [14]	Ours
Orchids	19.03 / 0.56 / 0.44	19.14 / 0.56 / 0.45	**19.66 / 0.59 / 0.41**	20.01 / 0.64 / 0.34	20.06 / 0.63 / 0.36	**20.14 / 0.66 / 0.31**	20.13 / 0.65 / 0.31	**20.18** / 0.64 / 0.33	20.09 / **0.67 / 0.28**
Trex	21.38 / 0.75 / 0.41	21.52 / 0.76 / 0.41	**23.89 / 0.81 / 0.37**	25.46 / 0.87 / 0.28	26.01 / 0.87 / 0.28	**26.67 / 0.90 / 0.23**	26.57 / 0.89 / 0.24	26.83 / 0.89 / 0.24	**27.55 / 0.91 / 0.20**
Leaves	18.87 / 0.60 / 0.44	18.81 / 0.58 / 0.47	**19.33 / 0.61 / 0.42**	20.51 / 0.68 / 0.33	20.40 / 0.66 / 0.37	**20.63 / 0.71 / 0.30**	**20.85** / 0.71 / 0.30	20.73 / 0.69 / 0.34	20.82 / **0.72 / 0.26**
Horns	20.19 / 0.71 / 0.42	20.49 / 0.71 / 0.43	**23.74 / 0.75 / 0.39**	26.21 / 0.82 / 0.29	26.18 / 0.81 / 0.30	**27.20 / 0.85 / 0.24**	27.04 / 0.85 / 0.25	26.99 / 0.84 / 0.26	**27.94 / 0.88 / 0.20**
Fern	15.91 / 0.62 / 0.49	16.02 / 0.61 / 0.51	**23.52 / 0.72 / 0.40**	24.49 / 0.78 / 0.31	24.35 / 0.76 / 0.34	**25.14 / 0.81 / 0.29**	25.00 / 0.80 / 0.27	24.82 / 0.79 / 0.30	**25.42 / 0.82 / 0.26**
Fortress	21.98 / 0.70 / 0.41	23.13 / 0.72 / 0.39	**28.49 / 0.78 / 0.34**	29.37 / 0.83 / 0.27	29.11 / 0.82 / 0.28	**30.40 / 0.86 / 0.22**	29.92 / 0.85 / 0.23	29.78 / 0.84 / 0.24	**30.71 / 0.88 / 0.19**
Flower	24.07 / 0.75 / 0.34	24.19 / 0.74 / 0.35	**25.49 / 0.77 / 0.31**	26.75 / 0.81 / 0.25	26.67 / 0.80 / 0.27	**27.26 / 0.83 / 0.21**	27.39 / 0.83 / 0.22	27.42 / 0.83 / 0.23	**27.66 / 0.85 / 0.18**
Room	26.57 / 0.87 / 0.39	25.77 / 0.87 / 0.39	**28.22 / 0.90 / 0.34**	29.33 / 0.92 / 0.28	30.01 / 0.93 / 0.26	**31.19 / 0.94 / 0.21**	30.48 / 0.94 / 0.23	30.98 / 0.94 / 0.22	**32.05 / 0.95 / 0.18**
Average	21.00 / 0.70 / 0.42	21.13 / 0.69 / 0.43	**24.04 / 0.74 / 0.37**	25.27 / 0.79 / 0.29	25.35 / 0.79 / 0.31	**26.08 / 0.82 / 0.25**	25.92 / 0.81 / 0.26	25.97 / 0.81 / 0.27	**26.53 / 0.83 / 0.22**
Mic	28.29 / 0.95 / 0.08	29.55 / 0.96 / 0.08	**30.90 / 0.97 / 0.06**	31.16 / 0.97 / 0.05	32.79 / **0.98** / 0.04	**33.94 / 0.98 / 0.03**	32.61 / 0.98 / **0.03**	33.96 / 0.98 / **0.03**	**34.66 / 0.99 / 0.03**
Ship	24.78 / 0.80 / 0.28	25.28 / 0.80 / 0.30	**25.99 / 0.83 / 0.26**	27.61 / 0.85 / 0.20	27.96 / 0.85 / 0.20	**28.33 / 0.87 / 0.18**	28.66 / 0.86 / 0.18	28.81 / 0.86 / 0.18	**29.06 / 0.87 / 0.17**
Lego	27.04 / 0.90 / 0.14	27.81 / 0.91 / 0.15	**29.58 / 0.94 / 0.09**	30.44 / 0.95 / 0.08	31.69 / 0.95 / 0.08	**32.70 / 0.97 / 0.04**	32.06 / 0.96 / 0.06	32.90 / 0.96 / 0.06	**33.85 / 0.97 / 0.03**
Chair	28.62 / 0.93 / 0.10	29.50 / 0.94 / 0.10	**31.12 / 0.96 / 0.07**	31.41 / 0.96 / 0.06	32.42 / 0.97 / 0.06	**33.46 / 0.98 / 0.04**	32.60 / 0.97 / 0.05	33.31 / 0.97 / 0.05	**34.15 / 0.98 / 0.03**
Materials	24.21 / 0.88 / 0.16	24.34 / 0.87 / 0.19	**25.22 / 0.90 / 0.14**	26.13 / 0.91 / 0.12	26.69 / 0.91 / 0.12	**27.29 / 0.93 / 0.10**	27.17 / 0.93 / 0.10	27.60 / 0.92 / 0.10	**27.87 / 0.94 / 0.09**
Hotdog	31.02 / 0.95 / 0.12	31.73 / 0.95 / 0.12	**33.44 / 0.96 / 0.10**	33.76 / 0.97 / 0.07	34.57 / 0.97 / 0.08	**35.59 / 0.98 / 0.06**	34.85 / 0.97 / 0.06	35.41 / 0.97 / 0.06	**36.12 / 0.98 / 0.05**
Drums	22.83 / 0.89 / **0.15**	23.01 / 0.88 / 0.20	**23.31 / 0.91 / 0.15**	23.93 / 0.91 / **0.12**	24.08 / 0.91 / 0.13	**24.23 / 0.92 / 0.12**	24.43 / 0.92 / **0.10**	**24.44** / 0.92 / 0.11	**24.44 / 0.93** / 0.12
Ficus	25.19 / 0.92 / 0.15	26.71 / 0.93 / 0.19	**28.97 / 0.95 / 0.11**	28.80 / 0.95 / **0.06**	30.09 / 0.96 / **0.06**	**30.51 / 0.97 / 0.06**	29.81 / 0.96 / **0.05**	30.76 / **0.97 / 0.05**	**30.85 / 0.97** / 0.06
Average	26.50 / 0.90 / 0.15	27.24 / 0.90 / 0.17	**28.57 / 0.93 / 0.12**	29.16 / 0.93 / 0.10	30.04 / 0.94 / 0.10	**30.76 / 0.95 / 0.08**	30.27 / 0.94 / 0.08	30.90 / 0.94 / 0.08	**31.38 / 0.95 / 0.07**

Table 1. **Convergence – NeRF**: for **(top rows)** LLFF dataset [21, 22] and the **(bottom rows)** the Synthetic dataset [22]. Our method provides best results for all cases for early iterations, and almost every case at 10k iterations, when training nearly converges.

the computation load even further. The only computation time that is added is the LMC update rule and the backward step of the last layer that computes the gradients w.r.t the input coordinates. Both of these should be insignificant compared to the actual neural field training.

Results. Fig. 6 and Tab. 1 shows the PSNR, SSIM, and LPIPS values for both the NeRF Synthetic and the LLFF datasets. As shown, our method is able to speed up convergence in almost all cases, significantly. Note especially the gap in performance in the earlier iterations. As depicted in Fig. 2, we more than double the convergence speed compared to uniform sampling and approximately double that of Edge [14]. More specifically, to reach a PSNR of 25 dB on the LLFF dataset our method takes ≈1,700 iterations, uniform sampling takes ≈3,800 iterations, and Edge [14] takes ≈3,300 iterations. On the NeRF Synthetic dataset to reach a PSNR of 30 dB it takes ≈2,400 iterations using our method, ≈8,500 iterations with uniform sampling, and ≈4,800 iterations with Edge [14]. Note that in the case of [50], another method based on image contexts and quadtree subdivision, their relative convergence gain with respect to uniform sampling is 18% and 15% for the NeRF Synthetic and the LLFF dataset with a final gain of 0.4 PSNR, which we comfortably outperform.

It is worth noting that while the Edge [14] baseline provides significantly improved results for the synthetic dataset, it does not perform as well on the LLFF scenes. We suspect that this is because the synthetic dataset is highly particular in that it is object-centric, with a flat white background, while the LLFF scenes are of natural images, thus with a rich background that can have much texture. We note here that our findings are different from what is reported in Edge [14], as they report ≈1 dB improvement on average, mostly coming from the 'Horns' sequence (21.24 dB with uniform sampling vs 25.45 dB with Edge [14]). However, with our NerfAcc implementation, uniform sampling already provides 27.04 dB at 10k iterations for this scene, and Edge [14] provides 26.99 dB. This difference could be due to implementation details, but we believe the gap in convergence speed between our method and uniform sam-



(a) NeRF Synthetic Dataset (b) LLFF Dataset

Figure 7. **Ablation – soft mining parameter** α: we report the effect of different α in terms of PSNR gain compared to uniform sampling for various training iterations. In both cases, the optimal choice of α is within (0,1). Note that simple hard mining, α=0 does not improve convergence for the LLFF dataset, whereas with α=0.8, our choice, it does.

pling is large enough, even when considering the reported difference. Further, our method improves convergence regardless of the data type.

4.3. Ablation studies

Effect of the soft mining parameter α. To examine the effect of the soft mining parameter α in (9), we look at the gain in PSNR values compared to that of uniform sampling for varying α in Fig. 7. As shown, neither complete hard mining nor pure importance sampling is optimal. Our method provides an effective compromise. Also note that the gains are more substantial in earlier stages of training as expected, as they both converge to similar solutions, but ours converges much faster. It is also important to notice that with α=0, which is equivalent to hard mining, results for the LLFF dataset do not improve—rather it degrades. As already discussed theoretically in Sec. 3.2, this hard mining would be a change of the actual objective being minimized, which could cause this performance degradation.

How effective is Langevin Monte Carlo? While we propose to use Langevin Monte Carlo (LMC) to sample with minimal sampling overhead, we also investigate how effective this is compared to the impractical multinomial sampling discussed in Sec. 3.3. Instead of LMC, at each batch construction step, we evaluate *all pixels* and form the training batch by sampling according to the true importance distribution $Q(\mathbf{x})$, via multinomial sampling. This is very costly, *e.g.*, increasing the training time to hours or days from minutes depending on the dataset, which destroys any practical gains. Nonetheless, it can be understood as the upper bound for what can be achieved when infinite compute and resource is available. Due to heavy computational demand, we only performed this experiment for the Room scene in the LLFF dataset. We report our results in Fig. 8. For both multinomial sampling and LMC, we keep the same hyperparameters as in other experiments, that is, α=0.8. As

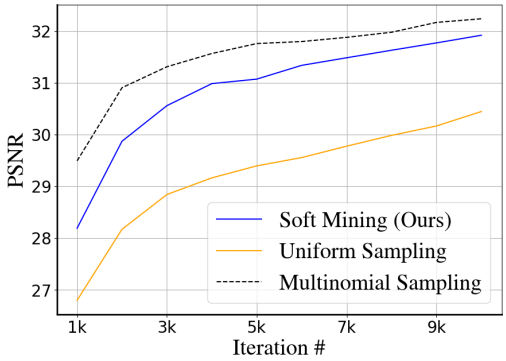


Figure 8. **Soft Mining with LMC vs ideal sampling of** $Q(\mathbf{x})$: we replace LMC sampling in our method with Multinomial Sampling, which, while computationally heavy, samples directly from $Q(\mathbf{x})$. Due to the heavy compute of Multinomial Sampling, we only report PSNR curves for the Room scene in the LLFF dataset. Our method provides an effective compromise with improved convergence with minimal increase in compute.

w/o Uniform	w/o Re-initialization	with both (Ours)
23.91	23.57	24.04

Table 2. **Uniform sampling and re-initialization**: We report PSNR of our method on LLFF dataset after 1k iterations of training, with either uniform sampling or re-initialization disabled.

shown, by exactly sampling from $Q(\mathbf{x})$ convergence is even higher compared to LMC, as, while samples from LMC theoretically converge to $Q(\mathbf{x})$, with finite samples there is approximation error. Regardless, even with this error, our LMC sampling performs better than uniform sampling, and provides an effective compromise given the small amount of compute it requires.

How important is (re-)initialization? We also validate the importance of uniform samples and re-initialization. As we report in Tab. 2, both help improve reconstruction quality. We found them to be particularly useful for achieving good final converged performance.

5. Conclusions

We presented how to accelerate neural field training by introducing soft mining in the construction of training batches, which we implement via Langevin Monte Carlo. We have demonstrated on 2D image fitting and NeRF, that our method leads to a two-fold+ improvement in convergence speed.

Limitations and future work. While our methods significantly improve results, we still rely purely on the loss function, which may not directly correlate with the application at hand. In NeRF, for example, training losses may not di-

rectly correlate with the novel-view rendering quality. As our framework does not depend on the choice of $Q(\mathbf{x})$, it could be possible to perhaps choose a different distribution to sample from, *e.g.*, depending on ray uncertainties [17] or based on active learning [11]. Our method sets a framework that allows easy exploration of such design choices, which was not possible before.

6. Acknowledgments

The authors would like to thank Ivan Krasin and David Fleet for their constructive feedback and support of this work. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant, NSERC Collaborative Research and Development Grant, Google, Digital Research Alliance of Canada, and Advanced Research Computing at the University of British Columbia.

References

[1] TorchScript Documentation. https://pytorch.org/docs/stable/jit.html, 2023. Accessed: 2023-11-16. 6

[2] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields. *arXiv preprint arXiv:2304.06706*, 2023. 1

[3] Antoine Blanchard and Themistoklis Sapsis. Bayesian Optimization with Output-weighted Optimal Sampling. *Journal of Computational Physics*, 425:109901, 2021. 2

[4] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable Transformations of Python+NumPy Programs, 2018. Version 0.3.13. 6

[5] Nicolas Brosse, Éric Moulines, and Alain Durmus. The Promises and Pitfalls of Stochastic Gradient Langevin Dynamics. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018. 2, 4

[6] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-Informed Neural Networks (PINNs) for Fluid Mechanics: A Review. *Acta Mechanica Sinica*, 37(12):1727–1738, 2021. 1

[7] Ang Cao and Justin Johnson. Hexplane: A Fast Representation for Dynamic Scenes. *arXiv preprint arXiv:2301.09632*, 2023. 2

[8] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient Geometry-aware 3D Generative Adversarial Networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022.

[9] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial Radiance Fields. In *Eur. Conf. Comput. Vis.*, 2022. 1

[10] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. MobileNeRF: Exploiting the Polygon Rasterization Pipeline for Efficient Neural Field Rendering on Mobile Architectures. *arXiv preprint arXiv:2208.00277*, 2022. 2

[11] Jongwon Choi, Kwang Moo Yi, Jihoon Kim, Jincho Choo, Byoungjip Kim, Jin-Yeop Chang, Youngjune Gwon, and Hyung Jin Chang. VaB-AL: Incorporating Class Imbalance and Difficulty with Variational Bayes for Active Learning. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 9

[12] Alain Durmus, Eric Moulines, and Marcelo Pereyra. Efficient Bayesian Computation by Proximal Markov Chain Monte Carlo: When Langevin Meets Moreau. *SIAM Journal on Imaging Sciences*, 11(1):473–506, 2018. 2

[13] Sara Fridovich-Keil, Giacomo Meanti, Frederik Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit Radiance Fields in Space, Time, and Appearance. *arXiv preprint arXiv:2301.10241*, 2023. 2

[14] Zhenbiao Gai, Zhenyang Liu, Min Tan, Jiajun Ding, Jun Yu, Mingzhao Tong, and Junqing Yuan. EGRA-NeRF: Edge-Guided Ray Allocation for Neural Radiance Fields. *Image and Vision Computing*, 134:104670, 2023. 2, 3, 6, 7

[15] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity Neural Rendering at 200fps. In *Int. Conf. Comput. Vis.*, pages 14346–14355, 2021. 2

[16] Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov chain Monte Carlo in Practice*. CRC press, 1995. 4

[17] Lily Goli, Cody Reading, Silvia Sellán, Alec Jacobson, and Andrea Tagliasacchi. Bayes' Rays: Uncertainty Quantification for Neural Radiance Fields. In *arXiv preprint arXiv:2309.03185*, 2023. 9

[18] Ruilong Li, Hang Gao, Matthew Tancik, and Angjoo Kanazawa. NerfAcc: Efficient Sampling Accelerates NeRFs. *arXiv preprint arXiv:2305.04966*, 2023. 6

[19] David B Lindell, Dave Van Veen, Jeong Joon Park, and Gordon Wetzstein. Bacon: Band-limited Coordinate Networks for Multiscale Scene Representation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 16252–16262, 2022. 1

[20] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural Sparse Voxel Fields. In *Adv. Neural Inform. Process. Syst.*, 2020. 2

[21] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines. *ACM Trans. Graph.*, 2019. 6, 7

[22] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *Eur. Conf. Comput. Vis.*, 2020. 1, 2, 3, 4, 5, 6, 7

[23] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.*, 2022. 1, 2, 5, 6

[24] Müller, Thomas. tiny-cuda-nn, 2021. Version 1.7, BSD-3-Clause License. 5

[25] Radford Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2012. 2, 4

[26] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 1, 3

[27] Akarsh Pokkunuru, Pedram Rooshenas, Thilo Strauss, Anuj Abhishek, and Taufiquar Khan. Improved Training of Physics-Informed Neural Networks Using Energy-Based Priors: a Study on Electrical Impedance Tomography. In *The Eleventh International Conference on Learning Representations*, 2022. 1

[28] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. Derf: Decomposed Radiance Fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 14153–14161, 2021. 2

[29] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. In *Int. Conf. Comput. Vis.*, pages 14335–14345, 2021. 2

[30] Mehdi S. M. Sajjadi, Henning Meyer, Etienne Pot, Urs Bergmann, Klaus Greff, Noha Radwan, Suhani Vora, Mario Lucic, Daniel Duckworth, Alexey Dosovitskiy, Jakob Uszkoreit, Thomas Funkhouser, and Andrea Tagliasacchi. Scene Representation Transformer: Geometry-Free Novel View Synthesis Through Set-Latent Scene Representations. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2

[31] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance Fields without Neural Networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2

[32] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training Region-based Object Detectors with Online Hard Example Mining. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 761–769, 2016. 2, 3

[33] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. Discriminative Learning of Deep Convolutional Feature Point Descriptors. In *Proceedings of the IEEE international conference on computer vision*, pages 118–126, 2015. 2, 3

[34] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations. In *Adv. Neural Inform. Process. Syst.*, 2019. 2

[35] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit Neural Representations with Periodic Activation Functions. In *Adv. Neural Inform. Process. Syst.*, 2020. 1, 3, 4

[36] Vincent Sitzmann, Semon Rezchikov, William T. Freeman, Joshua B. Tenenbaum, and Fredo Durand. Light Field Networks: Neural Scene Representations with Single-Evaluation Rendering. In *Adv. Neural Inform. Process. Syst.*, 2021. 2

[37] Mohammed Suhail, Carlos Esteves, Leonid Sigal, and Ameesh Makadia. Light Field Neural Rendering. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 1

[38] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5459–5469, 2022. 2

[39] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. In *Adv. Neural Inform. Process. Syst.*, 2020. 1

[40] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural Discrete Representation Learning. In *Adv. Neural Inform. Process. Syst.*, 2017. 3

[41] Xinshao Wang, Yang Hua, Elyor Kodirov, Guosheng Hu, and Neil M Robertson. Deep Metric Learning by Online Soft Mining and Class-aware Attention. In *AAAI*, 2019. 2

[42] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Trans. Image Process.*, 13(4): 600–612, 2004. 6

[43] Max Welling and Yee W Teh. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *Int. Conf. on Machine Learning.*, pages 681–688, 2011. 2

[44] Zhijie Wu, Yuhe Jin, and Kwang Moo Yi. Neural Fourier Filter Bank. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 14153–14163, 2023. 5

[45] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. In *Computer Graphics Forum*. Wiley Online Library, 2022. 1

[46] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural Fields in Visual Computing and Beyond. *Comput. Graph. Forum*, 2022. 2

[47] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance. In *Adv. Neural Inform. Process. Syst.*, 2020. 2

[48] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *Int. Conf. Comput. Vis.*, 2021. 1, 2

[49] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 586–595, 2018. 6

[50] Wenyuan Zhang, Ruofan Xing, Yunfan Zeng, Yu-Shen Liu, Kanle Shi, and Zhizhong Han. Fast Learning Radiance Fields by Shooting Much Fewer Rays. *IEEE Trans. Image Process.*, 2023. 2, 3, 7