Supplementary material for NeuralLift-360: Lifting An In-the-wild 2D Photo to A 3D Object with 360° Views

Dejia Xu, Yifan Jiang, Peihao Wang, Zhiwen Fan, Yi Wang, Zhangyang Wang
VITA Group, University of Texas at Austin
{dejia,atlaswang}@utexas.edu

1. Introduction

In this Supplementary Material, we first provide a detailed derivation of our probabilistic driven diffusion prior, then additional training recipe for NeuralLift-360. Furthermore, we present more implementation details and experimental comparisons.

2. Detailed Derivation.

In this section, we provide detailed mathematical derivation towards our objective loss. We repeat our notations and settings as below.

Notations. Given an image $\boldsymbol{y} \in \mathbb{R}^N$ and its text description $\boldsymbol{z} \in \mathbb{R}^D$, NeuralLift-360 intends to reconstruct a 3D scene $\boldsymbol{V} \in \mathbb{R}^M$, where $\boldsymbol{V}$ is a parameterization of 3D scene that can be be either a radiance volume or the implicit neural representation. We regard $\boldsymbol{y}$, $\boldsymbol{z}$ and $\boldsymbol{V}$ as random variables. Further on, we define $h(\boldsymbol{V}, \boldsymbol{\Phi})$ as the rendering function that displays $\boldsymbol{V}$ with respect to the camera pose $\boldsymbol{\Phi}$.

2.1. Pose Conditioned Latent Model

To reconstruct $\boldsymbol{V}$ from $\boldsymbol{y}$ and $\boldsymbol{z}$, we maximize a log-posterior and apply Bayesian rule:

$$\log p(\boldsymbol{V}|\boldsymbol{y}, \boldsymbol{z}) = \underbrace{\log p(\boldsymbol{y}|\boldsymbol{V}, \boldsymbol{z})}_{\text{likelihood}} + \underbrace{\log p(\boldsymbol{V}|\boldsymbol{z})}_{\text{prior}} + const., \tag{1}$$

where $const.$ denotes the evidence term, a constant. Next step is to introduce camera pose $\boldsymbol{\Phi} \in \mathbb{SO}(3) \times \mathbb{R}^3$ as a latent variable, then the likelihood term can be rewritten as:

$$p(\boldsymbol{y}|\boldsymbol{V}, \boldsymbol{z}) = \int p(\boldsymbol{y}|\boldsymbol{V}, \boldsymbol{\Phi}, \boldsymbol{z}) p(\boldsymbol{\Phi}|\boldsymbol{V}, \boldsymbol{z}) d\boldsymbol{\Phi} \tag{2}$$

$$= \int p(\boldsymbol{y}|\boldsymbol{V}, \boldsymbol{\Phi}, \boldsymbol{z}) \frac{p(\boldsymbol{V}|\boldsymbol{\Phi}, \boldsymbol{z}) p(\boldsymbol{\Phi})}{p(\boldsymbol{V}|\boldsymbol{z})} d\boldsymbol{\Phi} \tag{3}$$

$$= \frac{1}{p(\boldsymbol{V}|\boldsymbol{z})} \mathbb{E}_{\boldsymbol{\Phi}} \left[p(\boldsymbol{y}|\boldsymbol{V}, \boldsymbol{\Phi}, \boldsymbol{z}) p(\boldsymbol{V}|\boldsymbol{\Phi}, \boldsymbol{z}) \right]. \tag{4}$$

where we assume $\boldsymbol{\Phi}$ is independent of $\boldsymbol{z}$. We proceed by substituting Eq. 3 into Eq. 1:

$$\log p(\boldsymbol{V}|\boldsymbol{y}, \boldsymbol{z}) = \log \frac{\mathbb{E}_{\boldsymbol{\Phi}} \left[p(\boldsymbol{y}|\boldsymbol{V}, \boldsymbol{\Phi}, \boldsymbol{z}) p(\boldsymbol{V}|\boldsymbol{\Phi}, \boldsymbol{z}) \right]}{p(\boldsymbol{V}|\boldsymbol{z})} + \log p(\boldsymbol{V}|\boldsymbol{z}) + const. \tag{5}$$

$$= \log \mathbb{E}_{\boldsymbol{\Phi}} \left[p(\boldsymbol{y}|\boldsymbol{V}, \boldsymbol{\Phi}, \boldsymbol{z}) p(\boldsymbol{V}|\boldsymbol{\Phi}, \boldsymbol{z}) \right] + const. \tag{6}$$

After we apply Jensen inequality to obtain an evidence lower evidence of Eq. 1:

$$\log \mathbb{E}_{\boldsymbol{\Phi}} \left[p(\boldsymbol{y}|\boldsymbol{V}, \boldsymbol{\Phi}, \boldsymbol{z}) p(\boldsymbol{V}|\boldsymbol{\Phi}, \boldsymbol{z}) \right] + const.$$
$$\geq \mathbb{E}_{\boldsymbol{\Phi}} \left[\log p(\boldsymbol{y}|\boldsymbol{V}, \boldsymbol{\Phi}, \boldsymbol{z}) + \log p(\boldsymbol{V}|\boldsymbol{\Phi}, \boldsymbol{z}) \right] + const. \tag{7}$$

Afterwards, we use rendering function $h(\boldsymbol{V}, \boldsymbol{\Phi})$ to bridge the 3D likelihood estimation to the image domain. Specifically, we let $p(\boldsymbol{y}|\boldsymbol{V}, \boldsymbol{\Phi}, \boldsymbol{z}) = p(\boldsymbol{y}|h(\boldsymbol{V}, \boldsymbol{\Phi}), \boldsymbol{z})$ and $p(\boldsymbol{V}|\boldsymbol{\Phi}, \boldsymbol{z}) = p(h(\boldsymbol{V}, \boldsymbol{\Phi})|\boldsymbol{z})$. Then we derive a general training objective (omitting the constant):

$$\mathcal{L} = -\mathbb{E}_{\boldsymbol{\Phi}} \left[\underbrace{\log p(\boldsymbol{y}|h(\boldsymbol{V}, \boldsymbol{\Phi}), \boldsymbol{z})}_{\text{referenced loss}} + \underbrace{\log p(h(\boldsymbol{V}, \boldsymbol{\Phi})|\boldsymbol{z})}_{\text{non-referenced loss}} \right] \tag{8}$$

$$= -\mathbb{E}_{\boldsymbol{\Phi}} \left[\log p(h(\boldsymbol{V}, \boldsymbol{\Phi})|\boldsymbol{y}, \boldsymbol{z}) \right] + const., \tag{9}$$

where the constant equals to $-\log p(\boldsymbol{y}|\boldsymbol{z})$.

2.2. Diffusion Model as Evidence Lower Bound

Our next step is to surrogate the probability densities with the score matching loss, which can leverage pre-trained generative prior for view regularization. Our derivation mainly follows from [6, 10]. Below we summarize key steps to attain the ELBO, and more details can be found in

Chap. 3 of [10]:

$$\log p(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z}) \geq \mathbb{E}\left[\log \frac{p(\boldsymbol{x}_1,\cdots,\boldsymbol{x}_T|\boldsymbol{y},\boldsymbol{z})}{q(\boldsymbol{x}_1,\cdots,\boldsymbol{x}_T|\boldsymbol{x}_0)}\right]$$
$$= \sum_{t=2}^{T} \mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)\|p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{y},\boldsymbol{z}))\right]$$
$$+ \mathbb{E}_{q(\boldsymbol{x}_1|\boldsymbol{x}_0)}\left[\log p_\theta(\boldsymbol{x}_0|\boldsymbol{x}_1)\right] + const. \tag{10}$$

Consider a diffusion process with marginal distribution: $q(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\alpha_t\boldsymbol{x}_0, \sigma_t^2\boldsymbol{I})$. As shown by [6, 17], the KL divergence term can be simplified as:

$$\mathcal{D}_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)\|p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{y},\boldsymbol{z})) \tag{11}$$
$$= w(t)\,\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\boldsymbol{0},\boldsymbol{I})}\left[\|\boldsymbol{\epsilon}_\theta(\alpha_t\boldsymbol{x}_0 + \sigma_t\boldsymbol{\epsilon}|\boldsymbol{y},\boldsymbol{z}) - \boldsymbol{\epsilon}\|_2^2\right], \tag{12}$$

and similarly, we have:

$$\mathbb{E}_{q(\boldsymbol{x}_1|\boldsymbol{x}_0)}\left[\log p_\theta(\boldsymbol{x}_0|\boldsymbol{x}_1)\right] \tag{13}$$
$$= w(1)\,\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\boldsymbol{0},\boldsymbol{I})}\left[\|\boldsymbol{\epsilon}_\theta(\alpha_1\boldsymbol{x}_0 + \sigma_1\boldsymbol{\epsilon}|\boldsymbol{y},\boldsymbol{z}) - \boldsymbol{\epsilon}\|_2^2\right], \tag{14}$$

where $w(t)$ is a time dependent coefficient, $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z}) = \nabla\log p_\theta(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z})$ is known as the score function of the approximated image distribution. Altogether, we can utilize the following objective in the place of Eq. 8 (omitting constants):

$$\mathcal{L}_{diff} = -\sum_{t=1}^{T} w(t)\,\mathbb{E}_{\boldsymbol{\Phi},\boldsymbol{\epsilon}}\left[\|\boldsymbol{\epsilon}_\theta(\alpha_t h(\boldsymbol{V},\boldsymbol{\Phi}) + \sigma_t\boldsymbol{\epsilon}|\boldsymbol{y},\boldsymbol{z}) - \boldsymbol{\epsilon}\|_2^2\right]. \tag{15}$$

Finally, we introduce how to leverage classifier guidance idea [8] to compute $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z})$. Define some off-the-shelf text-to-image diffusion model as $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}|\boldsymbol{z})$. Note that it does not condition on the reference image $\boldsymbol{y}$. First, we use simple Bayesian rule to cast $p_\theta(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z})$ as:

$$p_\theta(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z}) = \frac{p_\theta(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{z})p_\theta(\boldsymbol{x}|\boldsymbol{z})}{p_\theta(\boldsymbol{y}|\boldsymbol{z})}. \tag{16}$$

Then the score function can be written as:

$$\boldsymbol{\epsilon}_\theta(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z}) = \nabla\log\left(\frac{p_\theta(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{z})p_\theta(\boldsymbol{x}|\boldsymbol{z})}{p_\theta(\boldsymbol{y}|\boldsymbol{z})}\right) \tag{17}$$
$$= \nabla\log p_\theta(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{z}) + \boldsymbol{\epsilon}_\theta(\boldsymbol{x}|\boldsymbol{z}). \tag{18}$$

One can further use classifier-free guidance to further improve Eq. 18 as:

$$\boldsymbol{\epsilon}_\theta(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z}) = \nabla\log p_\theta(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{z}) + (1+\omega)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}|\boldsymbol{z}) - \omega\boldsymbol{\epsilon}_\theta(\boldsymbol{x}), \tag{19}$$

where ω is the guidance strength of the classifier.

2.3. Computational Specifications

After going through all the probabilistic derivations, we specify several implementations leading towards our final loss function:

CLIP Similarity Guidance. We choose to use CLIP as our reference image guidance. Specifically, we measure the discrimination with a distance metric on the feature space:

$$p_\theta(\boldsymbol{y}|\boldsymbol{x}_t,\boldsymbol{z}) \propto \exp\left[-\phi\left(\frac{\boldsymbol{x}_t - \sigma_t\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t|\boldsymbol{z})}{\alpha_t}, \boldsymbol{y}\right)\right], \tag{20}$$

where we choose the inner product on the CLIP embedding space [13] as the similarity metric [1], i.e., $\phi(\boldsymbol{x},\boldsymbol{y}) = \langle\boldsymbol{x},\boldsymbol{y}\rangle$. Then we can derive the guidance penalty term we use in our implementation:

$$\log p_\theta(\boldsymbol{y}|\boldsymbol{x}_t,\boldsymbol{z}) \propto -\left\langle F\left(\frac{\boldsymbol{x}_t - \sigma_t\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t|\boldsymbol{z})}{\alpha_t}\right), F(\boldsymbol{y})\right\rangle, \tag{21}$$

where $F(\cdot)$ is a CLIP image encoder.

Camera Sampling Strategy. Camera poses $\boldsymbol{\Phi}$ can be in general uniformly sampled over $\mathbb{SO}(3)\times\mathbb{R}^3$ (with reasonably bounded sub-region). Since our reconstruction tasks are object centric, we first sample the camera positions over a unit sphere and control the camera orientation to look at the origin point. What's more, we can leverage a more strategic scheme to fully utilize the reference view. To be more concrete, we consider a Bernounii random variable $B \sim \mathrm{Bern}(\lambda)$ (as if a switch). When the variable is turned on (i.e., $B = 1$), we utilize the aforementioned sampling algorithm on the unit sphere, and apply $\mathcal{L}_{diff}$ as our loss function. When it is turned off, we only select the fixed camera pose $\boldsymbol{\Phi}_0$ associated with our reference image (i.e., a Dirac delta distribution centered at $\boldsymbol{\Phi}_0$), and only maximize a likelihood between the synthesized image and the reference image:

$$p_\theta(h(\boldsymbol{V},\boldsymbol{\Phi})|\boldsymbol{y},\boldsymbol{z},B=0) = \mathcal{N}(h(\boldsymbol{V},\boldsymbol{\Phi})|\boldsymbol{y},\sigma^2\boldsymbol{I}). \tag{22}$$

This leads to the definition of our final objective:

$$\mathcal{L}_{total} = -\mathbb{E}_{\boldsymbol{\Phi}}\left[\log p(h(\boldsymbol{V},\boldsymbol{\Phi})|\boldsymbol{y},\boldsymbol{z})\right] \tag{23}$$
$$= -p(B=1)\,\mathbb{E}_{\boldsymbol{\Phi}|B=1}\left[\log p(h(\boldsymbol{V},\boldsymbol{\Phi})|\boldsymbol{y},\boldsymbol{z},B=1)\right]$$
$$- p(B=0)\log p(h(\boldsymbol{V},\boldsymbol{\Phi}_0)|\boldsymbol{y},\boldsymbol{z},B=0) \tag{24}$$
$$\leq \lambda\mathcal{L}_{diff} + (1-\lambda)/\sigma^2\,\|h(\boldsymbol{V},\boldsymbol{\Phi}_0) - \boldsymbol{y}\|_2^2. \tag{25}$$

[1] Rigorously speaking, inner product is not a metric. However, it is widely used as a similarity score.

Fine-tuning Diffusion Model. We also interpret our domain adaption finetuning as a step to condition the diffusion model $\epsilon(\boldsymbol{x}|\boldsymbol{z})$ on the reference information $\boldsymbol{y}$:

$$\boldsymbol{\epsilon}_\theta(\boldsymbol{x}|\boldsymbol{z}) \xrightarrow{\mathcal{L}_{\text{finetune w/ } \boldsymbol{y}}} \boldsymbol{\epsilon}_{\theta^*}(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z}). \tag{26}$$

Pseudo-Depth Supervision. The pseudo-depth supervision on the reference view can be viewed as adding a regularization term onto Eq. 22:

$$\begin{aligned} &p_\theta(h(\boldsymbol{V},\boldsymbol{\Phi})|\boldsymbol{y},\boldsymbol{z},B=0) = \\ &\mathcal{N}(h_{rgb}(\boldsymbol{V},\boldsymbol{\Phi})|\boldsymbol{y},\sigma^2\boldsymbol{I})\cdot p(h_{depth}(\boldsymbol{V},\boldsymbol{\Phi})|\boldsymbol{y}), \end{aligned} \tag{27}$$

where we extend the renderer $h(\boldsymbol{V},\boldsymbol{\Phi})$ to independently render RGB image and depth map, and we further define $p(\boldsymbol{d}|\boldsymbol{y}) \propto \exp(-\mathcal{L}_{\text{ranking}}(\boldsymbol{d}, G(\boldsymbol{y})))$, $G(\cdot)$ is a monocular depth estimator.

3. Additional Training Recipe

Lighting Augmentation with Shading NeRF is prone to poor geometry, so we incorporate the supervision of surface normal from RefNeRF [20] to improve the geometry quality. Similar to DreamFusion [12] and RefNeRF [20], we replace NeRF's parameterization of view-dependent outgoing radiance with the surface itself. Compared with traditional NeRF that emits radiance conditioning on the viewing direction, our implementation expresses the geometry itself and allows additional shading.

The surface normal vector is defined as the negative gradient of the volume density with respect to the 3D location [3, 18, 20],

$$\hat{\mathbf{n}}(\mathbf{x}) = -\frac{\nabla\sigma(\mathbf{x})}{\|\nabla\sigma(\mathbf{x})\|} \tag{28}$$

As a result, we render the color along the ray during diffuse reflectance [14] as follows:

$$\mathbf{c} = \boldsymbol{\rho}\circ(\boldsymbol{\ell}_\rho \circ \max(0, \boldsymbol{n}\cdot(\boldsymbol{\ell}-\boldsymbol{\mu})/\|\boldsymbol{\ell}-\boldsymbol{\mu}\|) + \boldsymbol{\ell}_a), \tag{29}$$

where l is the point light location, l_p is the color of the point light, and l_a is the color of the ambient light. We generate the point light location by randomly sampling an offset from the camera location $l \sim \mathcal{N}(l_{\text{cam}}, \mathbf{I})$.

Geometry Regularizations Without any regularization, the NeRF model is free to generate arbitrary geometry in those unobserved regions. Traditionally, this doesn't affect the image quality, but when we perform additional shading, the image quality becomes dependent on the geometry quality, which requires additional priors.

We observe that the generated 3D object tends to produce a flat object with foggy floaters in the back. This is due to the object generating a semi-transparent surface in the back and emitting the front view directly. We avoid this issue by penalizing the backward-facing surface normal [20],

$$\mathcal{L}_{\text{orient}} = \sum_i \text{detach}(w_i)\max(0, \hat{\mathbf{n}}_i'\cdot\mathbf{d})^2, \tag{30}$$

where d is the ray direction. This orient loss prevents the surface normal from being in the same direction as the ray direction, so that the shading will not result in a totally black surface.

Since we don't have depth regularization for exact values, the generated contents might be floating in front of the camera. While this can deliver good RGB images in some viewing directions, the exploding geometry usually suffers from severe artifacts. To this end, we utilize distortion loss [2, 19] and sparsity loss to provide unsupervised regularizations. The distortion loss encourages each ray to be as compact as possible:

$$\begin{aligned} \mathcal{L}_{\text{dist}}(s,w) = &\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} w_i w_j \left|\frac{s_i+s_{i+1}}{2} - \frac{s_j+s_{j+1}}{2}\right| \\ &+ \frac{1}{3}\sum_{i=0}^{N-1} w_i^2 (s_{i+1}-s_i), \end{aligned} \tag{31}$$

where $(s_{i+1}-s_i)$ is the length and $(\frac{s_i+s_{i+1}}{2})$ is the midpoint of the i-th query interval. The sparsity loss further improves the sparsity of the occupancy by imposing an L1 regularization on the generated alpha map.

$$\mathcal{L}_{\text{sparsity}} = \left\| \int_{t_n}^{t_f} T(t)\sigma(r(t))dt \right\|. \tag{32}$$

To avoid the depth we generate being spiky, we further incorporate the inverse depth smoothing loss [21] as a self-supervision:

$$\mathcal{L}_{\text{smooth}}(d_i) = e^{-\nabla^2\mathcal{I}(\mathbf{x}_i)}\left(|\partial_{xx}d_i| + |\partial_{xy}d_i| + |\partial_{yy}d_i|\right), \tag{33}$$

where d_i is the depth map, $\nabla^2\mathcal{I}(\mathbf{x}_i)$ refers to the Laplacian of pixel value at location x_i. This helps our rendered depth to be consistent with the rendered RGB, instead of only following the internal ranking relationship.

Multi-resolution rendering Since we use Instant-NGP [11] based implementation to support rendering patches of 128×128 during training, obtaining the gradient using autograd is tedious to implement. We instead utilize the finite difference method to obtain the density gradient. To do so, we have to query our model eight more times in the 3D neighborhood of each point. The GPU consumption is huge if we try to obtain a normal map in high resolution, so we only render a low resolution 100×100 normal map to lower the computation burden.

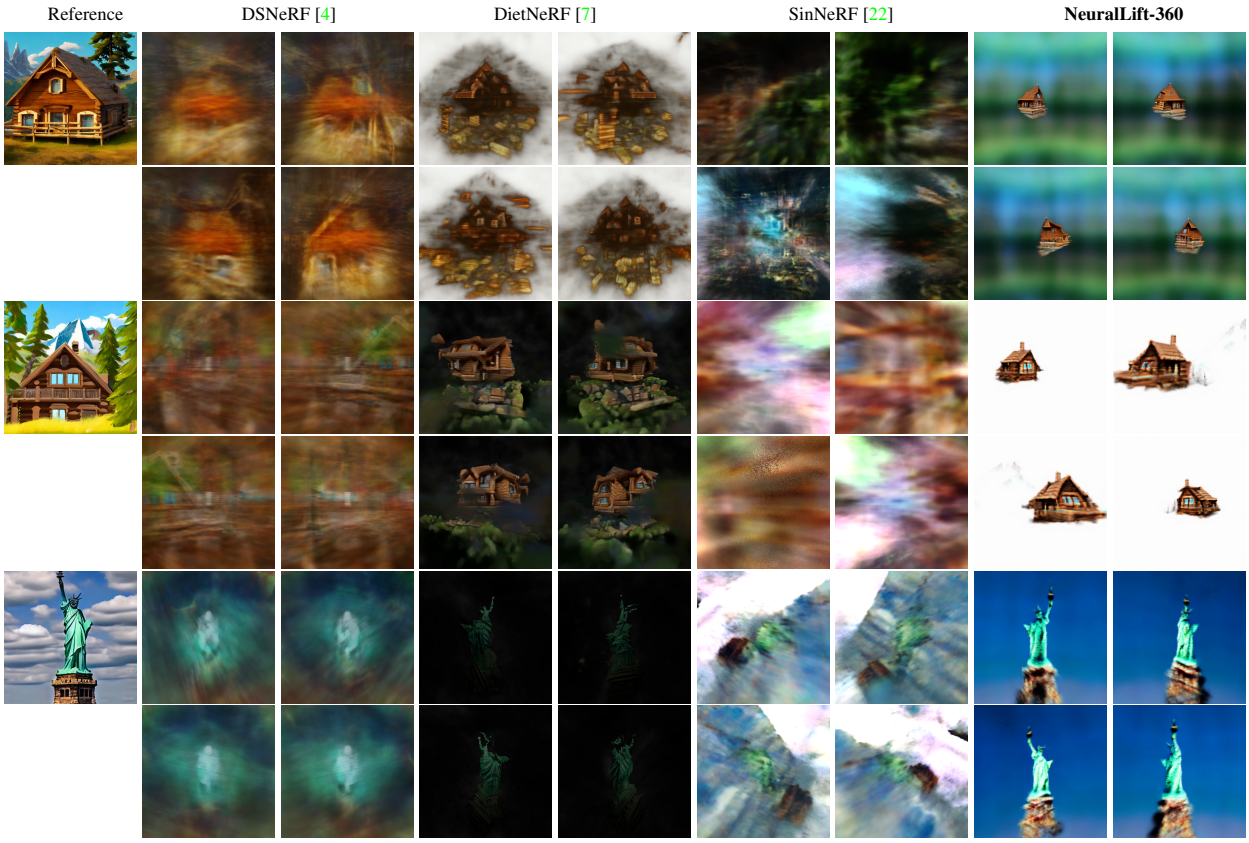

Figure 1. More visual comparisons with state-of-the-art methods. We also provide **a video comparison in "video.mp4".**

Background Scene Representation The diffusion models are usually trained on images with diverse backgrounds, but we only focus on the foreground object. To solve this gap, we utilize another background NeRF to generate coherent background. The background NeRF takes viewing direction as input and generates a harmonious background. We empirically find that incorporating this background module alleviates the burden for the foreground NeRF, and it helps to remove floating artifacts.

Guidance weight choice Large guidance weight hurts the performance of general text-to-image models such as Imagen [16] and Stable Diffusion [15] because they are performing the iterative sampling. Given a large guidance weight, the result x_0 prediction might exceed the range of $[-1, 1]$. This leads to a train-test mismatch for them since, during training, the input for the diffusion models is generally inside the range of $[-1, 1]$. However, such a range issue doesn't exist in our case since we only do one iteration of sampling for each rendered image. We empirically choose the guidance weight as 100.

4. Implementation Details

NeRF Architecture We implement our NeRF architecture based on the open-source implementation torch-ngp [1]. The NeRF consists of a hash grid encoder with 16 levels and an MLP to query the features. The coarse resolution is 16, and the finest resolution is 2048. The number of feature dimensions per entry is 2 for the hash grid. Our MLP consists of a sequence of four fully-connected layers with residual connections. The activations layers in between are SiLU [5]. The final output layer of the NeRF MLP is four channels, consisting of color and density. We further employ a Sigmoid activation on the density before volumetric rendering. We also adopt a background NeRF to module the background scene. The background module conditions on the ray direction and has three fully-connected layers.

Diffusion Prior Settings We utilize Stable Diffusion version 1.4 as our diffusion prior. During training, we uniformly sample the timesteps in the range of $[50, 950]$. This design stabilizes training by avoiding very high and very low noise levels. We use PNDM [9] Sampler with a training timestep of 1,000 to perform the noise perturbation and

calculate the $\hat{x_0}$ estimate. The β range is [0.00085, 0.012], and the β schedule is scaled linear.

Training Settings We implement our framework using PyTorch. To speed up the training, we implement gradient propagation for depth using CUDA with the help of PyTorch extensions. The whole framework is trained on a single A6000 GPU for 10k steps. The overall training takes about 1.5 hours. For the first 1k steps, we disable the shading, and for the remaining steps, we use diffuse shading for 50% of the time. During inference, we disable the shading.

The initial learning rate is $1e-3$, and decay the learning rate using LambdaLR Scheduler in PyTorch. The optimizer is Adam, and the betas are set to $(0.9, 0.99)$. We set the weight for $\mathcal{L}_{\text{orient}}$ to be 10. For $\mathcal{L}_{\text{dist}}$, we find reasonable weights to lie in $[1e-2, 1e-1]$. For $\mathcal{L}_{\text{sparsity}}$, we set the weight to be $1e-3$. For $\mathcal{L}_{\text{smooth}}$, we set the weight to be 0.1.

References

[1] Torch ngp. https://github.com/ashawkey/torch-ngp. 4

[2] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022. 3

[3] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T Barron, Ce Liu, and Hendrik Lensch. Nerd: Neural reflectance decomposition from image collections. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12684–12694, 2021. 3

[4] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised nerf: Fewer views and faster training for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12882–12891, 2022. 4

[5] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018. 4

[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. 1, 2

[7] Ajay Jain, Matthew Tancik, and Pieter Abbeel. Putting nerf on a diet: Semantically consistent few-shot view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5885–5894, 2021. 4

[8] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and Fujie Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006. 2

[9] Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo numerical methods for diffusion models on manifolds. *arXiv preprint arXiv:2202.09778*, 2022. 4

[10] Calvin Luo. Understanding diffusion models: A unified perspective. *arXiv preprint arXiv:2208.11970*, 2022. 1, 2

[11] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022. 3

[12] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022. 3

[13] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021. 2

[14] Ravi Ramamoorthi and Pat Hanrahan. A signal-processing framework for inverse rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 117–128, 2001. 3

[15] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022. 4

[16] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022. 4

[17] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 2

[18] Pratul P Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T Barron. Nerv: Neural reflectance and visibility fields for relighting and view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7495–7504, 2021. 3

[19] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Improved direct voxel grid optimization for radiance fields reconstruction. *arxiv cs.GR 2206.05085*, 2022. 3

[20] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T Barron, and Pratul P Srinivasan. Ref-nerf: Structured view-dependent appearance for neural radiance fields. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5481–5490. IEEE, 2022. 3

[21] Chaoyang Wang, José Miguel Buenaposada, Rui Zhu, and Simon Lucey. Learning depth from monocular videos using direct methods. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2022–2030, 2018. 3

[22] Dejia Xu, Yifan Jiang, Peihao Wang, Zhiwen Fan, Humphrey Shi, and Zhangyang Wang. Sinnerf: Training neural radiance fields on complex scenes from a single image. *arXiv preprint arXiv:2204.00928*, 2022. 4