Appendix

A. Coordinate System & Camera Model

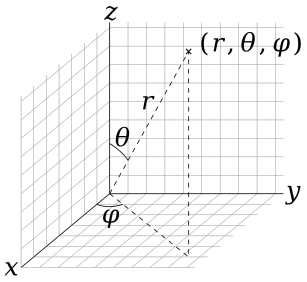

Figure 1: Spherical Coordinate System [**?**].

We use a spherical coordinate system to represent camera locations and their relative transformations. As shown in Figure 1, assuming the center of the object is the origin of the coordinate system, we can use θ, ϕ, and r to represent the polar angle, azimuth angle, and radius (distance away from the center) respectively. For the creation of the dataset, we normalize all assets to be contained inside the XYZ unit cube $[-0.5, 0.5]^3$. Then, we sample camera viewpoints such that $\theta \in [0, \pi]$, $\phi \in [0, 2\pi]$ uniformly cover the unit sphere, and r is sampled uniformly in the interval $[1.5, 2.2]$. During training, when two images from different viewpoints are sampled, let their camera locations be (θ_1, ϕ_1, r_1) and (θ_2, ϕ_2, r_2). We denote their *relative* camera transformation as $(\theta_2 - \theta_1, \phi_2 - \phi_1, r_2 - r_1)$. Since the camera is always pointed at the center of the coordinate system, the extrinsics matrices are uniquely defined by the location of the camera in a spherical coordinate system. We assume the horizontal field of view of the camera to be $49.1°$, and follow a pinhole camera model.

Due to the incontinuity of the azimuth angle, we encode it with $\phi \mapsto [\sin(\phi), \cos(\phi)]$. Subsequently, at both training and inference time, four values representing the relative camera viewpoint change, $[\theta, \sin(\phi), \cos(\phi), r]$ are fed to the model, along with an input image, in order to generate the novel view.

B. Dataset Creation

We use Blender [**?**] to render training images of the finetuning dataset. The specific rendering code is inherited from a publicly released repository[1] by authors of Objaverse [2]. For each object in Objaverse, we randomly sample 12 views and use the Cycles engine in Blender with 128 samples per ray along with a denoising step to render each image. We render all images in 512×512 resolution and pad transparent backgrounds with white color. We also apply randomized area lighting. In total, we rendered a dataset of around 10M images for finetuning.

[1] https://github.com/allenai/objaverse-rendering

C. Finetuning Stable Diffusion

We use the rendered dataset to finetune a pretrained Stable Diffusion model for performing novel view synthesis. Since the original Stable Diffusion network is not conditioned on multimodal text embeddings, the original Stable Diffusion architecture needs to be tweaked and finetuned to be able to take conditional information from an image. This is done in [1], and we use their released checkpoints. To further adapt the model to accept conditional information from an image along with a relative camera pose, we concatenate the image CLIP embedding (dimension 768) and the pose vector (dimension 4) and initialize another fully-connected layer ($772 \mapsto 768$) to ensure compatibility with the diffusion model architecture. The learning rate of this layer is scaled up to be $10\times$ larger than the other layers. The rest of the network architecture is kept the same as the original Stable Diffusion.

C.1. Training Details

We use AdamW [**?**] with a learning rate of 10^{-4} for training. First, we attempted a batch size of 192 while maintaining the original resolution (image dimension 512×512, latent dimension 64×64) for training. However, we discovered that this led to a slower convergence rate and higher variance across batches. Because the original Stable Diffusion training procedure used a batch size of 3072, we subsequently reduce the image size to 256×256 (and thus the corresponding latent dimension to 32×32), in order to be able to increase the batch size to 1536. This increase in batch size has led to better training stability and a significantly improved convergence rate. We finetuned our model on an $8\times$A100-80GB machine for 7 days.

C.2. Inference Details

To generate a novel view, Zero-1-to-3 takes only 2 seconds on an RTX A6000 GPU. Note that in prior works, typically a NeRF is trained in order to render novel views, which takes significantly longer. In comparison, our approach inverts the order of 3D reconstruction and novel view synthesis, causing the novel view synthesis process to be fast and contain diversity under uncertainty. Since this paper addresses the problem of a single image to a 3D object, when an in-the-wild image is used during inference, we apply an off-the-shelf background removal tool [4] to every image before using it as input to Zero-1-to-3.

D. 3D Reconstruction

Different from the original Score Jacobian Chaining (SJC) implementation, we removed the "emptiness loss" and "center loss". To regularize the VoxelRF representation,

we differentiably render a depth map, and apply a smoothness loss to the depth map. This is based on the prior knowledge that the geometry of an object typically contains less high-frequency information than its texture. It is particularly helpful in removing holes in the object representation. We also apply a near-view consistency loss to regularize the difference between an image rendered from one view and another image rendered from a nearby randomly sampled view. We found this to be very helpful in improving the cross-view consistency of an object's texture. All implementation details can be found in the code that is submitted as part of the appendix. Running a full 3D reconstruction on an image takes around 30 minutes on an RTX A6000 GPU.

Mesh extraction. We extract the 3D mesh from the VoxelRF representation as follows. We first query the density grids at resolution 200^3. Then we smooth the density grids using a mean filter of size $(7, 7, 7)$, followed by an erosion operator of size $(5, 5, 5)$. Finally, we run marching cubes on the resulting density grids. Let $\bar{d}$ denote the average value of the density grids. For the GSO dataset, we use a density threshold of $8\bar{d}$. For the RTMV dataset, we use a density threshold of $4\bar{d}$.

Evaluation. The ground truth 3D shape and the predicted 3D shape are first normalized within the unit cube. To compute the chamfer distance (CD), we randomly sample 2000 points. For Point-E and MCC, we sample from their predicted point clouds directly. For our method and SJC-I, we sample points from the reconstructed 3D mesh. We compute the volumetric IoU at resolution 64^3. For our method, Point-E and SJC-I, we vocalize the reconstructed 3D surface meshes using marching cubes. For MCC, we directly voxelize the predicted dense point clouds by occupancy.

E. Baselines

To be consistent with the scope of our method, we compare only to methods that (1) operate in a zero-shot setting, (2) use single-view RGB images as input, and (3) have official reference implementations available online that can be adapted in a reasonable timeframe. In the following sections, we describe the implementation details of our baselines.

E.1. DietNeRF

We use the official implementation located on GitHub[2], which, at the time of writing, has code for low-view NeRF optimization from scratch with a joint MSE and consistency loss, though provides no functionality related to finetuning PixelNeRF. For fairness, we use the same hyperparameters as the experiments performed with the NeRF synthetic dataset in [3]. For the evaluation of novel view synthesis, we render the resulting NeRF from the designated camera poses in the test set.

E.2. Point-E

We use the official implementation and pretrained models located on GitHub[3]. We keep all the hyperparameters and follow their demo example to do 3D reconstruction from single input image. The prediction is already normalized, so we do not need to perform any rescaling to match the ground truth. For surface mesh extraction, we use their default method with a grid size of 128.

E.3. MCC

We use the official implementation located on GitHub[4]. Since this approach requires a colorized point cloud as input rather than an RGB image, we first apply an online off-the-self foreground segmentation method [4] as well as a state-of-the-art depth estimation method [6, 5] for preprocessing. For fairness, we keep all hyperparameters the same as the zero-shot, in-the-wild experiments described in [7]. For the evaluation of 3D reconstruction, we normalize the prediction, rotate it according to camera extrinsics, and compare it with the 3D ground truth.

References

[1] Stable diffusion image variations - a hugging face space by lambdalabs. 1

[2] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3D objects. *arXiv preprint arXiv:2212.08051*, 2022. 1

[3] Ajay Jain, Matthew Tancik, and Pieter Abbeel. Putting NeRF on a Diet: Semantically consistent few-shot view synthesis. In *ICCV*, 2021. 2

[4] OPHoperHPO. Ophoperhpo/image-background-remove-tool: automated high-quality background removal framework for an image using neural networks. . 1, 2

[5] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12179–12188, 2021. 2

[6] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3):1623–1637, 2020. 2

[7] Chao-Yuan Wu, Justin Johnson, Jitendra Malik, Christoph Feichtenhofer, and Georgia Gkioxari. Multiview compressive coding for 3D reconstruction. *arXiv:2301.08247*, 2023. 2

[2] https://github.com/ajayjain/DietNeRF

[3] https://github.com/openai/point-e

[4] https://github.com/facebookresearch/MCC