ReconFusion: 3D Reconstruction with Diffusion Priors

Rundi Wu[1,2*] Ben Mildenhall[1*] Philipp Henzler[1] Keunhong Park[1] Ruiqi Gao[3] Daniel Watson[3]
Pratul P. Srinivasan[1] Dor Verbin[1] Jonathan T. Barron[1] Ben Poole[3] Aleksander Hołyński[1*]

[1]Google Research [2]Columbia University [3]Google DeepMind

[*] denotes equal contribution

A. Diffusion Model Details

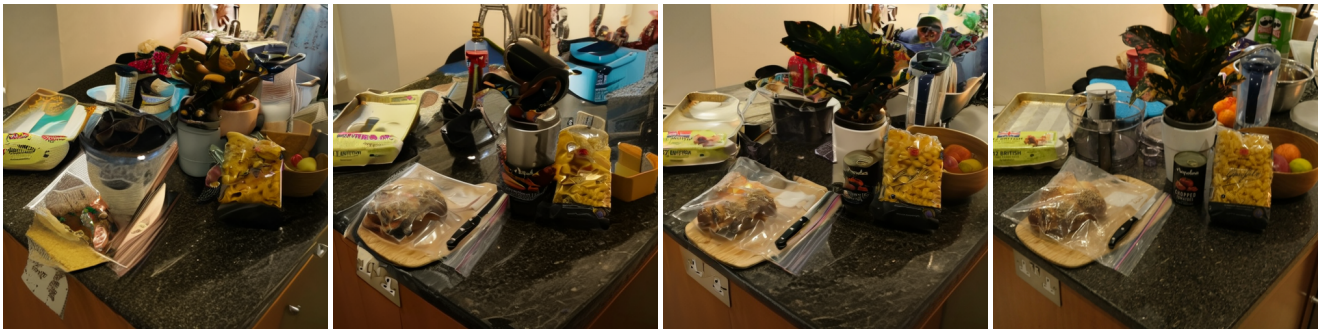
Our diffusion model is adapted from a pre-trained text-to-image latent diffusion model that maps $512 \times 512 \times 3$ inputs into a latent dimension of $64 \times 64 \times 8$. We modify this initial model to accept the necessary conditioning signals for text-free novel-view synthesis. We replace the inputs to the cross-attention pathway (which typically consist of a sequence of CLIP text embeddings) with the outputs of an additional dense layer. The input to this dense layer is a concatenated tensor consisting of (1) the unconditional CLIP text embedding (*i.e*. the empty string `""`), and (2) the CLIP image embeddings of each of the input conditioning frames. We initialize the weights of this dense layer such that it produces the unconditional CLIP text embedding at the start of fine-tuning. This mechanism is inspired by the fine-tuning process in Zero-1-to-3 [3]. Zero-1-to-3 fine-tunes from an *image variations* base model that has been previously fine-tuned to enable conditioning on CLIP image embeddings. We fine-tune directly from a text-conditioned model, but our architecture can learn image variation-like behavior through the dense layer. Furthermore, unlike Zero-1-to-3, our dense layer does not take pose as input, since the 3D transformation between the input conditioning frames and the target frame is applied through the PixelNeRF rendering process. In addition to the cross-attention modifications, we concatenate the outputs of the PixelNeRF model (a $64 \times 64 \times 131$ tensor consisting of RGB and features) to the input noise that is passed to the U-Net. As in prior work [2], we initialize the additional convolutional weights to zero such that the added inputs have no effect at the start of fine-tuning.

To enable classifier-free guidance on our added conditioning signals, we drop out all conditioning images for a training example with 10% probability. We drop out the CLIP and PixelNeRF conditioning pathways independently in order to enable separate guidance, although we found empirically that using the same guidance weight across both conditioning signals (*i.e*. performing joint CFG across both conditioning signals) produces optimal results. We train our model for $250{,}000$ iterations with a learning rate of 10^{-4} and a batch size of 128. Our training examples consist of 3 input conditioning images, 1 target image, and the corresponding relative poses between each input image and the target image. This data is sampled from CO3D, RealEstate10k, MVImgNet, and Objaverse with uniform probability. For Objaverse, we light the object with random environment maps and compose it onto a random solid background at each training iteration.

B. PixelNeRF Details

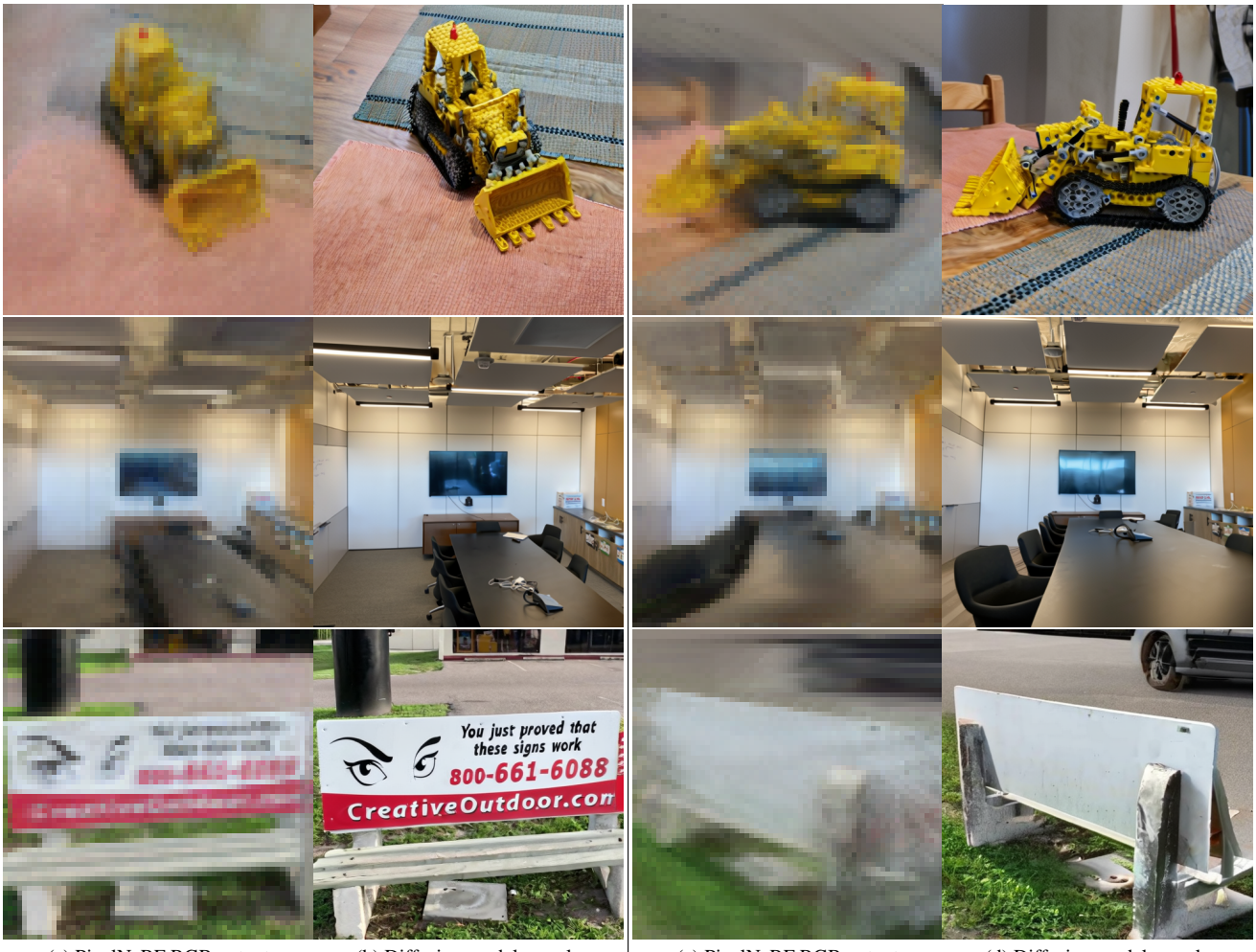
Our PixelNeRF module is inspired by but not identical to the architecture proposed in the original work [9]. The inputs are N images along with their camera poses (extrinsic and intrinsic matrices), along with a target camera pose. The output is an approximate rendering at the target camera pose (both RGB and feature channels), which is concatenated with the input to the diffusion U-Net to provide a strong conditioning signal that encodes the pose and image content of the target novel view. During inference, we typically resize (but do not crop) the inputs to PixelNeRF such that their shorter dimension is 512 pixels, since the model was trained on 512×512 resolution inputs. The output target image is always $64 \times 64 \times 131$ (3 RGB channels plus 128 feature channels), to match the latent resolution of the diffusion model.

The PixelNeRF module begins by passing all input images through a 2D U-Net to create feature images of equal spatial resolution with 128 channels. We then cast rays through each pixel of the target image, and sample 128 points along each ray from depth 0.5 to ∞ (uniform up to distance 1, then linear in disparity). We reproject these points into each of the input cameras and gather corresponding features from the feature images to make a gathered tensor of size $64 \times 64 \times N \times 128$. We append positionally-encoded 3D locations, as well as the mean and variance of these features over the N-long dimension corresponding to the number of inputs. A small MLP then processes this full tensor along the channel dimension to output a new set of features and weights. These weights are then used to compute a weighted sum along the N-long dimension, thereby producing a new tensor of size $64 \times 64 \times 128$.



(a) 3 input views (b) 6 input views (c) 9 input views (d) Ground truth

Figure 1. **More training views leads to better samples**. Here we show samples from the diffusion model at a novel viewpoint while varying the number of training images. In all cases, the diffusion model is given the three nearest images to the novel viewpoint. We see that as we increase the number of known scene observations, the expected distance to the nearest training view decreases, therefore increasing the fidelity of diffusion model samples.



(a) PixelNeRF RGB output (b) Diffusion model sample (c) PixelNeRF RGB output (d) Diffusion model sample

Figure 2. **PixelNeRF Visualization**. Here we show (a,c) a visualization of the 64×64 RGB component of the PixelNeRF output, and (b,d) the corresponding sample from the diffusion model, which is conditioned on the PixelNeRF outputs.

A second MLP then processes this summed tensor along the channel dimension to produce the final output of size $64 \times 64 \times (3 + 128)$.

All learned components (including the 2D U-Net used to extract image features) are initialized randomly and optimized jointly with the fine-tuned diffusion model U-Net. As mentioned in the main text, we apply an RGB reconstruction loss to encourage the PixelNeRF module to learn a useful conditioning signal. See Figure 2 for a visualization of our PixelNeRF model.

C. Dataset Details

For LLFF and DTU, we use the standard train/test splits proposed by earlier works. For RealEstate10k and CO3D, we select the training views evenly from all the frames and use every 8th of the remaining frames for evaluation. For the mip-NeRF 360 dataset we design a heuristic to choose a train split of views that are uniformly distributed around the hemisphere and pointed toward the central object of interest: We randomly sample 10^6 different 9-view splits and use the one that minimizes these heuristic losses, then further choose the 6- and 3-view splits to be subsets of the 9-view split.

We carefully rescale each dataset to be compatible with the near plane of 0.5 expected by the PixelNeRF module. DTU, CO3D, mip-NeRF 360 are rescaled by setting the "focus point" of the data to the origin and rescaling camera positions to fit inside a $[-1, 1]^3$ cube. RealEstate10k is pre-scaled by its creators to have a reasonable near distance of 1.0, so we simply multiply its camera positions by 0.5. LLFF similarly provides a near bound based on the COLMAP point cloud, which we use to rescale the data to allow a 0.5 near plane.

D. Baselines Details

Our Zip-NeRF [1] baseline has slight hyperparameter modifications from the original that better suit few-view reconstruction. This was done primarily to provide a maximally competitive baseline for our model, but these same hyperparameters are used by our model as well, and we observe a modest performance improvement due to them. In particular, we use:

- Distortion loss with weight 0.01,
- Normalized weight decay on the NGP grid parameters with strength 0.1,
- A smaller view-dependence network with width 32 and depth 1, to avoid overfitting,
- No hexagonal spiral control points, to accelerate rendering at the cost of introducing some aliasing,
- A downweighted density in the "contracted" region of space outside of the unit sphere, wherein we multiply the density emitted by Zip-NeRF by $|\det(\mathbf{J}_{\mathcal{C}}(\mathbf{x}))|$ (the isotropic scaling induced by the contraction function, see the supplement of Barron et al. [1]).

We find that this baseline performs competitively on forward-facing scenes such as LLFF and RealEstate10k, especially with 9 input views, but often produces many floaters or fails to reconstruct any meaningful geometry on the more difficult datasets. Further tuning and the addition of other heuristic regularizers (*e.g.*, the techniques used in RegNeRF or FreeNeRF) would likely improve results. However, the point of this model is to show baseline reconstruction performance with our diffusion model regularizer disabled, rather than to be a state-of-the-art few-view method. To re-emphasize this: the **only** difference between results labeled "Ours" and "Zip-NeRF" is that the diffusion regularizer weight is set to 0, all other hyperparameters are identical.

For RegNeRF [4] and FreeNeRF [8], we use the result images shared by the authors for the LLFF and DTU datasets, and run the authors' code for FreeNeRF on the other three datasets. For DiffusioNeRF [7], we use the authors' result images on the DTU dataset, and run their code on the other four datasets. Because SimpleNeRF [6] used different train/test splits for LLFF and DTU, we run the authors' code on all five datasets.

SparseFusion [10] was originally trained on CO3D using masked images of foreground objects. We re-train their models using the unmasked images in a category-specific manner, then use their 3D distillation pipeline to obtain the final rendered images.

ZeroNVS [5] is a concurrent work that trains a diffusion model for novel view synthesis of scenes from a single image. To evaluate its performance on multiview inputs, we modify its reconstruction pipeline by using the input view closest to the sampled random view for conditioning the diffusion model. For DTU and mip-NeRF 360 scenes (which they evaluate in their paper for the single input case), we follow their viewpoint selection strategy for sampling new views. For CO3D, we use the same strategy that is employed for the mip-NeRF 360 scenes. For RealEstate10K, we sample new views on a spline path fitted from the input views, then perturb them. For LLFF, we sample new views on a circle fitted from the input views, then perturb them.

References

[1] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Zip-NeRF: Anti-aliased grid-based neural radiance fields. *ICCV*, 2023. 3

[2] Tim Brooks, Aleksander Holynski, and Alexei A Efros. InstructPix2Pix: Learning to Follow Image Editing Instructions. *CVPR*, 2023. 1

[3] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-Shot One Image to 3D Object. *arXiv*, 2023. 1

[4] Michael Niemeyer, Jonathan T. Barron, Ben Mildenhall, Mehdi SM Sajjadi, Andreas Geiger, and Noha Radwan. RegNeRF: Regularizing Neural Radiance Fields for View Synthesis from Sparse Inputs. *CVPR*, 2022. 3

[5] Kyle Sargent, Zizhang Li, Tanmay Shah, Charles Herrmann, Hong-Xing Yu, Yunzhi Zhang, Eric Ryan Chan, Dmitry Lagun, Li Fei-Fei, Deqing Sun, et al. ZeroNVS: Zero-Shot 360-Degree View Synthesis from a Single Real Image. *arXiv:2310.17994*, 2023. 3

[6] Nagabhushan Somraj, Adithyan Karanayil, and Rajiv Soundararajan. SimpleNeRF: Regularizing Sparse Input Neural Radiance Fields with Simpler Solutions. *SIGGRAPH Asia*, 2023. 3

[7] Jamie Wynn and Daniyar Turmukhambetov. DiffusioNeRF: Regularizing neural radiance fields with denoising diffusion models. *CVPR*, 2023. 3

[8] Jiawei Yang, Marco Pavone, and Yue Wang. FreeNeRF: Improving Few-shot Neural Rendering with Free Frequency Regularization. *CVPR*, 2023. 3

[9] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural Radiance Fields from One or Few Images. *CVPR*, 2021. 1

[10] Zhizhuo Zhou and Shubham Tulsiani. SparseFusion: Distilling View-conditioned Diffusion for 3D Reconstruction. *CVPR*, 2023. 3