Supplementary Material for RealFusion 360° Reconstruction of Any Object from a Single Image

Luke Melas-Kyriazi Iro Laina Christian Rupprecht Andrea Vedaldi

A. Implementation Details

In this section, we provide full implementation details which were omitted from the main text due to space constraints. Most of these details follow [9], but a few are slightly modified.

Emission-absorption. Let $I \in \mathbb{R}^{3\times H\times W}$ be an image, so that $I(u) \in \mathbb{R}^3$ is the color of pixel u. In order to compute $I(u)$, one casts a ray r_u from the camera center through the pixel, interpreted as a point on the 3D image plane (this implicitly accounts for the camera viewpoint $\pi \in SE(3)$). Then, one takes a certain number of samples $(\boldsymbol{x}_i \in r_u)_{i\in\mathcal{N}}$, for indices $\mathcal{N} = \{1, \dots, N\}$ taken with constant spacing Δ. The color is obtained as:

$$I(u) = \mathcal{R}_u(\sigma, c, \pi) = \sum_{i\in\mathcal{N}} (T_{i+1} - T_i)c(\boldsymbol{x}_i), \qquad (1)$$

where $T_i = \exp(-\Delta \sum_{j=0}^{i-1} \sigma(\boldsymbol{x}_j))$ is the probability that a photon is transmitted from point $\boldsymbol{x}_i$ back to the camera sensor without being absorbed by the material.

Shading. We consider three different types of shading: albedo, diffuse, and textureless. For albedo, we simply render the RGB color of each ray as given by our model:

$$I(u) = I_\rho(u) = \mathcal{R}_u(\sigma, c, \pi)$$

For diffuse, we also compute the surface normal n as the normalized negative gradient of the density with respect to u. Then, given a point light l with color l_ρ and an ambient light with color l_a, we render

$$I(u) = I_\rho(u) \circ (l_\rho \circ \max(0, n \cdot \tfrac{l-u}{||l-u||} + l_a))$$

For textureless, we use the same equation with $I_\rho(u)$ replaced by white $(1, 1, 1)$.

For the reconstruction view, we only use albedo shading. For the random view (i.e. the view used for the prior objectives), we use albedo shading for the first 1000 steps of training by setting $l_a = 1.0$ and $l_\rho = 0.0$. Afterwards we use $l_a = 0.1$ and $l_\rho = 0.9$, and we select stochastically between albedo, diffuse, and textureless with probabilities 0.2, 0.4, and 0.4, respectively.

We obtain the surface normal using finite differences:

$$n = \frac{1}{2\cdot\epsilon}\begin{pmatrix} I(u+\epsilon_x) - I(u-\epsilon_x) \\ I(u+\epsilon_y) - I(u-\epsilon_y) \\ I(u+\epsilon_z) - I(u-\epsilon_z) \end{pmatrix}$$

where $\epsilon_x = (\epsilon, 0, 0)$, $\epsilon_y = (0, \epsilon, 0)$, and $\epsilon_z = (0, 0, \epsilon)$

Density bias. As in [9], we add a small Gaussian blob of density to the origin of the scene in order to assist with the early stages of optimization. This density takes the form

$$\sigma_{\text{init}}(\mu) = \lambda \cdot e^{-||\mu||^2/(2\nu^2)}$$

with $\lambda = 5$ and $\nu = 0.2$.

Camera. The fixed camera for reconstruction is placed at a distance of 1.8 from the origin, oriented toward the origin, at an elevation of $15°$ above the horizontal plane. For a small number of scenes in which the object of interest is clearly seen from overhead, the reconstruction camera is placed at an elevation of $40°$.

The camera for the prior objectives is sampled randomly at each iteration. Its distance from the origin is sampled uniformly from $[1.0, 1.5]$. Its azimuthal angle is sampled uniformly at random from the $360°$ around the object. Its elevation is sampled uniformly in degree space from $-10°$ to $90°$ with probability 0.5 and uniformly on the upper hemisphere with probability 0.5. The field of view is uniformly sampled between 40 and 70. The camera is oriented toward the origin. Additionally, every tenth iteration, we place the prior camera near the reconstruction camera: its location is sampled from the prior camera's location perturbed by Gaussian noise with mean 0 and variance 1.

Lighting. We sample the position of the point light by adding a noise vector $\eta \sim \mathcal{N}(0, 1)$ to the position of the prior camera.

View-Dependent Prompt. We add a view-dependent suffix to our text prompt based on the location of the prior camera relative to the reconstruction camera. If the prior camera is placed at an elevation of above $60°$, the text prompt receives the suffix "overhead view." If it is at an elevation below $0°$, the text receives "bottom view." Otherwise, for azimuthal angles of $\pm 30°$, $\pm 30 - 90°$, or $\pm 90 - 180°$ in

either direction of the reconstruction camera, it receives the suffices "front view," "side view," or "bottom view," respectively.

InstantNGP. Our InstantNGP [8] parameterizes the density and albedo inside a bounding box around the origin with side length 0.75. It is a multi-resolution feature grid with 16 levels, each with a feature dimension of 2. The maximum resolution is 2048. With coarse-to-fine training, only the first 8 (lowest-resolution) levels are used during the first half of training, while the others are masked with zeros. Each feature grid has dimensionality 2. The features from these grids are stacked and fed to a 3-layer MLP with 64 hidden units.

Rendering and diffusion prior. We render at resolution 96px. Since Stable Diffusion [11] is designed for images with resolution 512px, we upsample renders to 512px before passing them to the Stable Diffusion latent space encoder (i.e. the VAE). We add noise in latent space, sampling $t \sim \mathcal{U}(0.02, 0.98)$. We use classifier-free guidance strength 100. We found that results with classifier-free guidance strength above 30 produced good results; below 30 led to many more geometric deformities. Although we do not backpropagate through the Stable Diffusion UNet for $\mathcal{L}_{\text{SDS}}$, we do backpropagate through the latent space encoder.

Optimization. We optimize using the Adam [6] optimizer with learning rate $1e-3$ for 5000 iterations. The optimization process takes approximately 45 minutes on a single V100 GPU.

Background model. For our background model, we use a two-layer MLP which takes the viewing direction as input. This model is purposefully weak, such that the model cannot trivially optimize its objectives by using the background.

Additional regularizers. We additionally employ two regularizers on our density field. The first is the orientation loss from Ref-NeRF [12], also used in DreamFusion [9], for which we use $\lambda_{\text{orient}} = 0.01$. The second is an entropy loss which encourages points to be either fully transparent or fully opaque: $\mathcal{L}_{\text{entropy}} = (w \cdot \log_2(w) - (1-w) \cdot \log_2(1-w)$ where w is the cumulative sum of density weights computed as part of the NeRF rendering equation (Equation 1).

Single-image textual inversion. Our single-image textual inversion step, which is a variant of textual inversion [4], entails optimizing a token $\mathbf{e}$ introduced into the diffusion model text encode to match an input image. The key to making this optimization successful given only a single image is the use of heavy image augmentations, shown in Fig. 1. We optimize using these augmentations for a total of 3000 steps using the Adam optimizer [6] with image size 512px, batch size 16, learning rate $5 \cdot 10^{-4}$, and weight decay $1 \cdot 10^{-2}$.

The embedding $\mathbf{e}$ can be initialized either randomly, manually (by selecting a token from the vocabulary that matches the object), or using an automated method.

One automated method that we found to be successful was to use CLIP (which is also the text encoder of the Stable Diffusion model) to infer a starting token to initialize the inversion procedure. For this automated procedure, we begin by considering the set of all tokens in the CLIP text tokenizer which are nouns, according to the WordNet [3] database. We use only nouns because we aim to reconstruct objects, not reproduce styles or visual properties. We then compute text embeddings for captions of the form "An image of a $\langle$`token`$\rangle$" using each of these tokens. Separately, we compute the image embedding for the input image. Finally, we take the token whose caption is most similar to the image embedding as initialization for our textual inversion procedure.

We use the manual initialization method for the examples in the main paper and we use the automated initialization method for the examples in the supplemental material (i.e. those included below).

B. Additional Qualitative Examples

In Fig. 2, we show additional examples of reconstructions from our model. We see that our method is often able to reconstruct plausible geometries and object backsides.

C. Additional Comparisons

We provide additional comparisons to recent single-view reconstruction methods on the lego scene from the synthetic NeRF [7] dataset. We compare on the special test set created by SinNeRF [13], which consists of 60 views very close to the reference view. We emphasize that our method is not tailored to this setting, whereas the other methods are designed specifically for it. For example, some other methods work by warping the input image, which only performs well for novel views close to the reference view.

Method	Depth	PSNR	SSIM	LPIPS
PixelNeRF [14]		14.3	0.72	0.22
DietNeRF [5]		15.0	0.72	0.20
DS-NeRF [2]	✓	16.6	0.77	0.16
SinNeRF [13]	✓	**21.0**	**0.82**	**0.09**
RealFusion		16.5	0.76	0.25

Table 1. **Novel view synthesis comparison.** A comparison of RealFusion against recent single-view reconstruction methods on the task of novel view synthesis on the synthetic lego scene from NeRF [7]. These numbers are computed on the test set rendered by SinNeRF [13], which contains 60 views very close to the reference view. This is a setting highly favorable to methods that use depth supervision, such as DS-NeRF and SinNeRF .

```
transform = T.Compose([
    T.RandomApply([T.RandomRotation(degrees=10, fill=255)], p=0.75),
    T.RandomResizedCrop(image_size, scale=(0.70, 1.3)),
    T.RandomApply([T.ColorJitter(0.04, 0.04, 0.04, 0.04)], p=0.75),
    T.RandomGrayscale(p=0.10),
    T.RandomApply([T.GaussianBlur(5, (0.1, 2))], p=0.10),
    T.RandomHorizontalFlip(),
])
```

Figure 1. PyTorch code for the image augmentations used for single-image textual inversion.

D. Text-to-Image-to-3D

In this section, we explore the idea of reconstructing a 3D object from a text prompt alone by first using the text prompt to generate an image, and then reconstructing this image using RealFusion.

We show examples of text-to-image-to-3D generation in Fig. 3.

Compared to the one-step procedure of [9] (i.e. text-to-3D), this two-step procedure (i.e. text-to-image-to-3D) has the advantage that it may be easier for users to control. Under our setup, users can first sample a large number of images from a 2D diffusion model such as Stable Diffusion, select their desired image, and then lift it to 3D using RealFusion. It is possible that this setup could help help address the issue of diversity of generation discussed in [9]. Additionally, tn this setting, we find that it is usually not necessary to use single-image textual inversion, since the images sampled in the first stage are already extremely well-aligned with their respective prompts.

E. Analysis of Failure Cases

In Fig. 4, we show additional examples of failure cases from our model. Below, we analyzed what we find to be our three most common failure cases. The techniques we apply in RealFusion (single-image textual inversion, normals smoothing, and coarse-to-fine training) make these failure cases less frequent and less severe, but they still occur on various images.

Neural fields lacking well-defined geometry. One failure case of our method consists of the generation of a semi-transparent neural field which does not have a well-defined geometry. These fields tend to look like the input image when seen from the reference viewpoint, but do not resemble plausible objects when seen from other viewpoints. We note that this behavior is extremely common when using CLIP as a prior model, but it occurs occasionally even when using Stable Diffusion and $\mathcal{L}_{\text{SDS}}$.

Floaters. Another failure case involves "floaters," or disconnected parts of the scene which appear close to the camera. These floaters sometimes appear in front of the reference view as to make the corresponding render look like the input image. Without image-specific prompts, these floaters are a very big issue, appearing in the majority of reconstructions. When using image-specific prompts, the issue of floaters is greatly (but not entirely) alleviated.

The Janus Problem. Named after the two-faced Roman god Janus, the "Janus problem" refers to reconstructions which have two or more faces. This problem arises because the loss function tries to make the render of every view look like the input image, at least to a certain extent.

Our use of view-specific prompting partially alleviates this issue. For example, when we render an image of a panda from the back, we optimize using the text prompt "An image of a ⟨object⟩, back view", where "⟨object⟩" is our image-specific token corresponding to the image of a panda. However, even with view-specific prompting, this problem still occurs. This problem is visible with the panda in Fig. 3 (row 2). We note that this problem is not unique to our method; it can also be seen with [9] (see Figure 9, last row).

F. Unsuccessful Experiments and Regularization Losses

In the process of developing our method, we experimented with numerous ideas, losses, and regularization terms which were not included in our final method because they either did not improve reconstruction quality or did not improve it enough to justify their complexity. Here, we describe some of these ideas for the benefit of future researchers working on this problem.

Using DM for reconstruction loss. One idea we tried involved using the diffusion model within our reconstruction objective as well as our prior objective. This involved a modified version of $\mathcal{L}_{\text{SDS}}$ in which we compared the noise predicted by the diffusion model for our noisy rendered image to the noise predicted by the diffusion model for a noisy version of our input image. We found that with this loss we were able to reconstruct the input image to a certain degree, but that we did not match the exact input image colors or textures.

Normals smoothing in 3D. Our normals smoothing term operates in 2D, using normals rendered via the NeRF equation. We also tried different ways of smoothing normals in 3D. However, possibly due to our grid-based radiance field and/or our finite difference-based normals computation, we found that these regularization terms were all very noisy and harmful to reconstruction quality.

Using monocular depth. We tried incorporating monocular depth predictions into the pipeline, using pre-trained monocular depth networks such as MiDaS [10]. Specifically, we enforced that the depth rendered from the reference view matched the depth predicted by MiDaS for the input image. We found that this additional depth loss in most instances did not noticeably improve reconstruction quality and in some cases was harmful. Nonetheless, these results are not conclusive and future work could pursue other ways of integrating these components.

Using LPIPS and SSIM reconstruction losses. We tried using LPIPS [15] and SSIM losses in place of our L2 reconstruction loss. We found that LPIPS performed similarly to L2, but incurred additional computation and memory usage. We found that SSIM without either L2 and LPIPS resulted in worse reconstruction quality, but that it yielded fine results when combined with them. We did not include it in our final objective for the sake of simplicity.

Rendering at higher resolutions. Since Stable Diffusion operates on images of resolution 512px, it is conceivable that rendering at higher resolution would be benefitial with regard to the prior loss. However, we found no noticeable difference in quality when rendering at higher resolutions than 96px or 128px. For computational purposes, we used resolution 96px for all experiments in the main paper.

Using DINO-based prior losses. Similarly to the CLIP prior loss, one could imagine using other networks to encourage renders from novel views to be semantically similar to the input image. Due to the widespread success of the DINO [1] models in unsupervised learning, we tried using DINO feature losses in addition to the Stable Diffusion prior loss. Specifically, for each image rendered from a novel view, we computed a DINO image embedding and maximized its cosine similarity with the DINO image embedding of the reference image. We found that this did not noticeably improve or degrade performance. For purposes of simplicity, we did not include it.

G. Links to Images for Qualitative Results

For our qualitative results, we primarily use images from datasets such as Co3D. We also use a small number of images sourced directly from the web to show that our method works on uncurated web data. We provide links to all of these images on our project website.

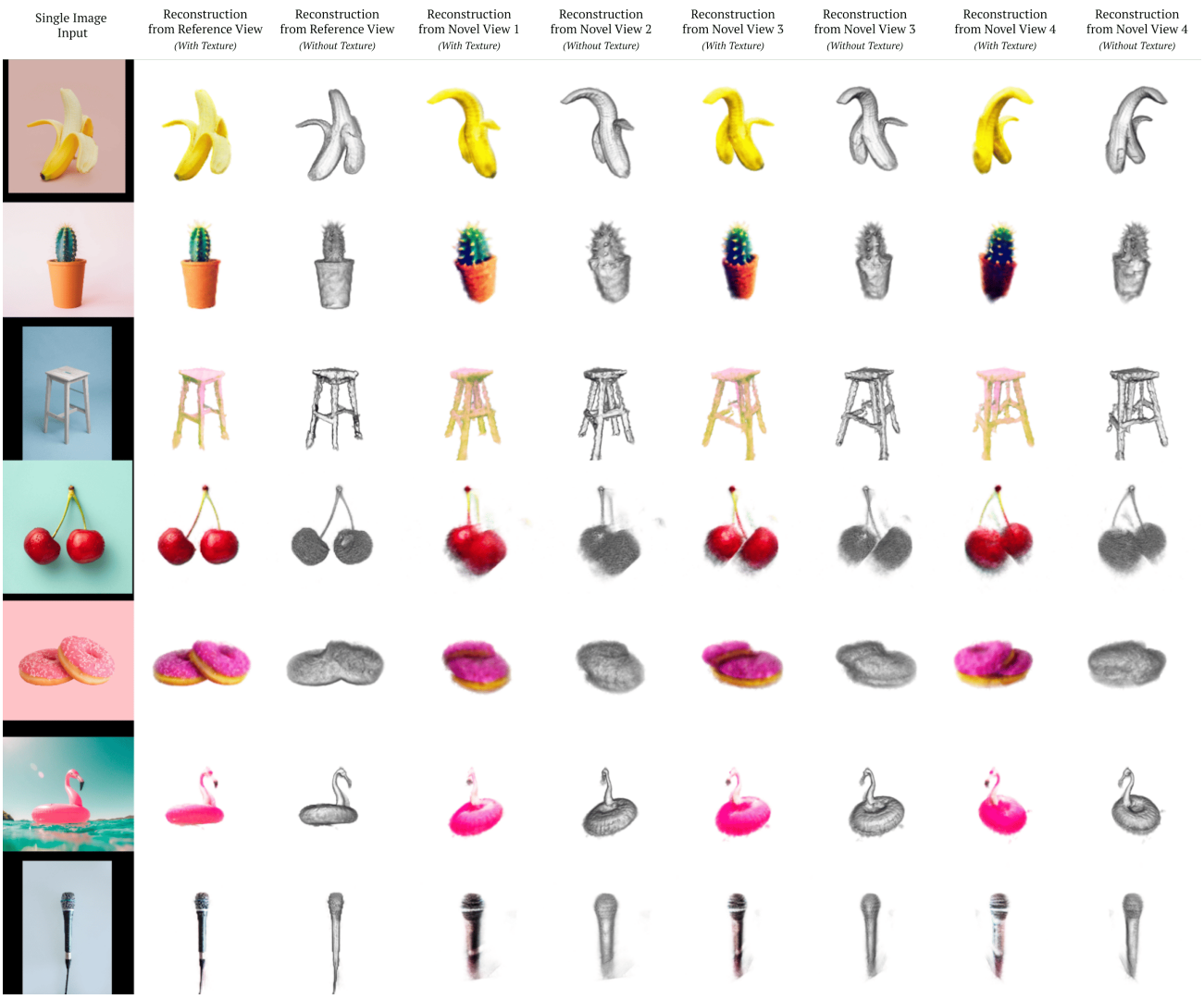

Figure 2. **Additional qualitative examples.** This figure presents additional qualitative examples from our model. The first column shows the input image. The second column shows the reconstruction from the reference viewpoint. The following columns show renders from novel viewpoints, demonstrating that our model is able to reconstruct plausible object shapes.

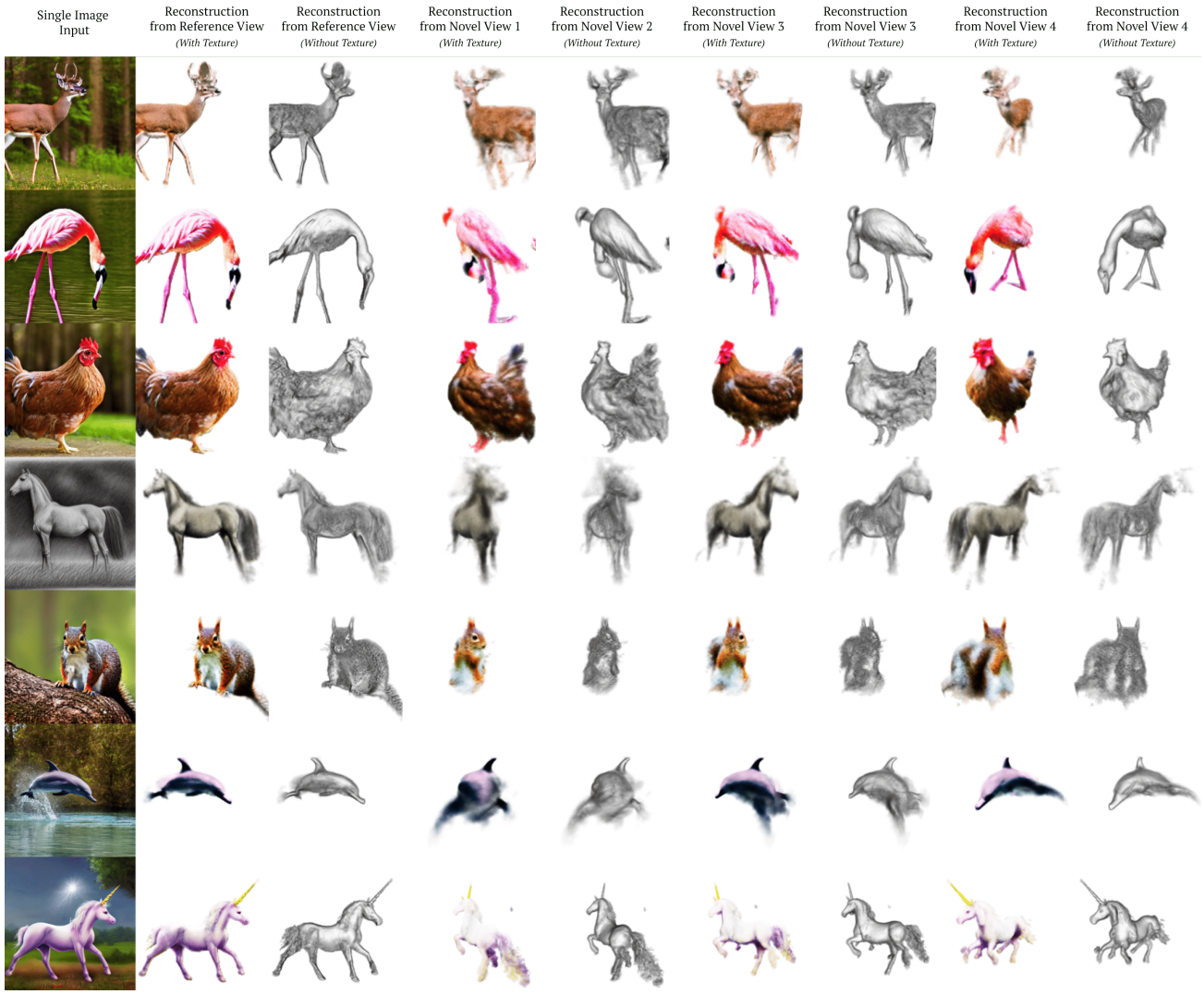

Figure 3. **Text-to-Image-to-3D.** This figure presents examples from our model using images generated directly from text prompts using Stable Diffusion [11]. The images were generated with the prompt "An image of a ____" where the blank space is replaced by "deer", "flamingo", "hen", "pencil drawing of a horse", "squirrel", "dolphin", and "unicorn", respectively. The results demonstrate that our method is able to reconstruct plausible object shapes even from synthetic images.

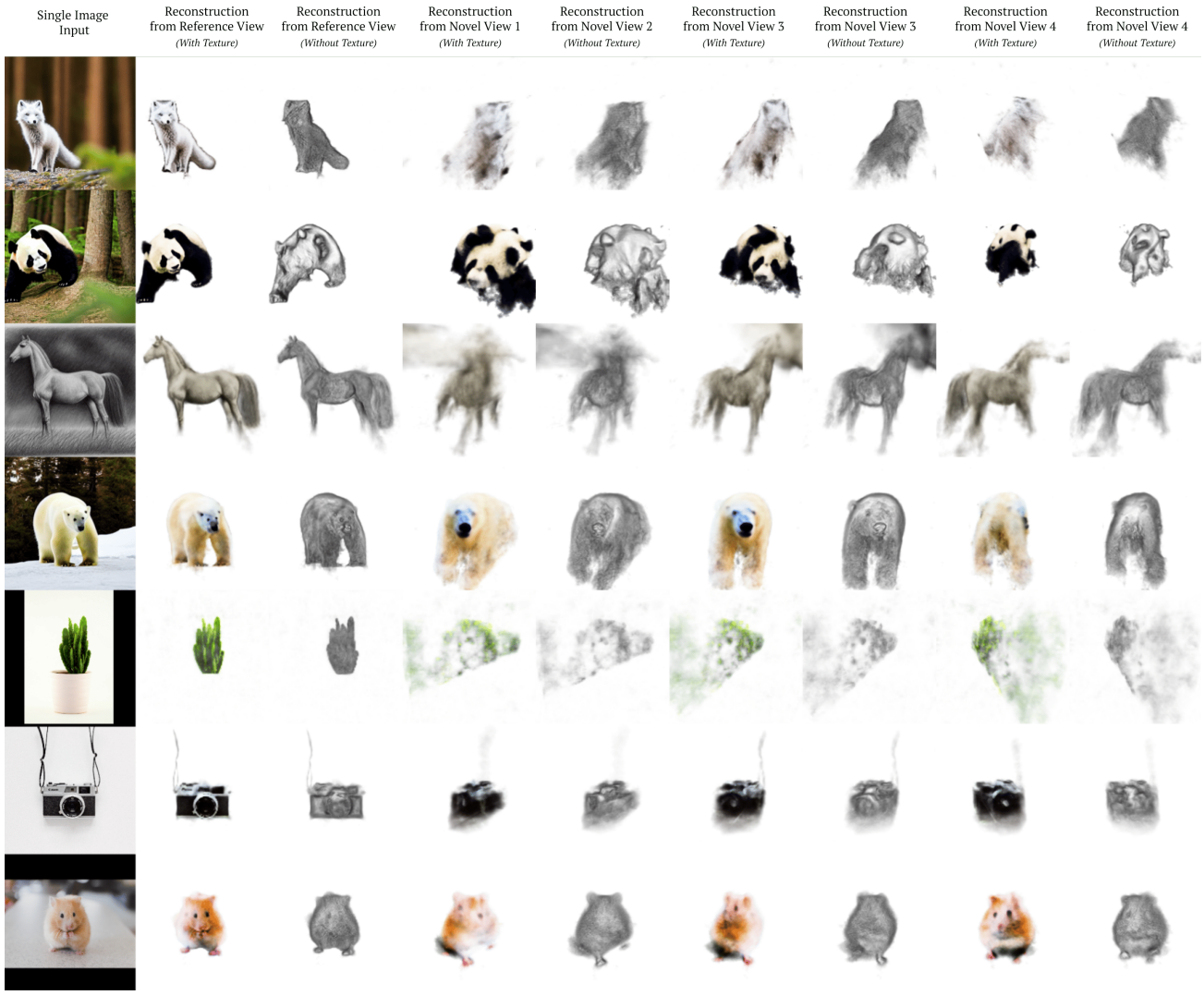

Figure 4. **Additional failure cases.** This figure presents additional failure cases from our model. The first column shows the input image. The second column shows the reconstruction from the reference viewpoint. The following columns show renders from novel viewpoints, which make clear why these examples are failure cases. Note that some examples (for example, the panda bear in the second row and the hamster in the last row) suffer from the Janus problem.

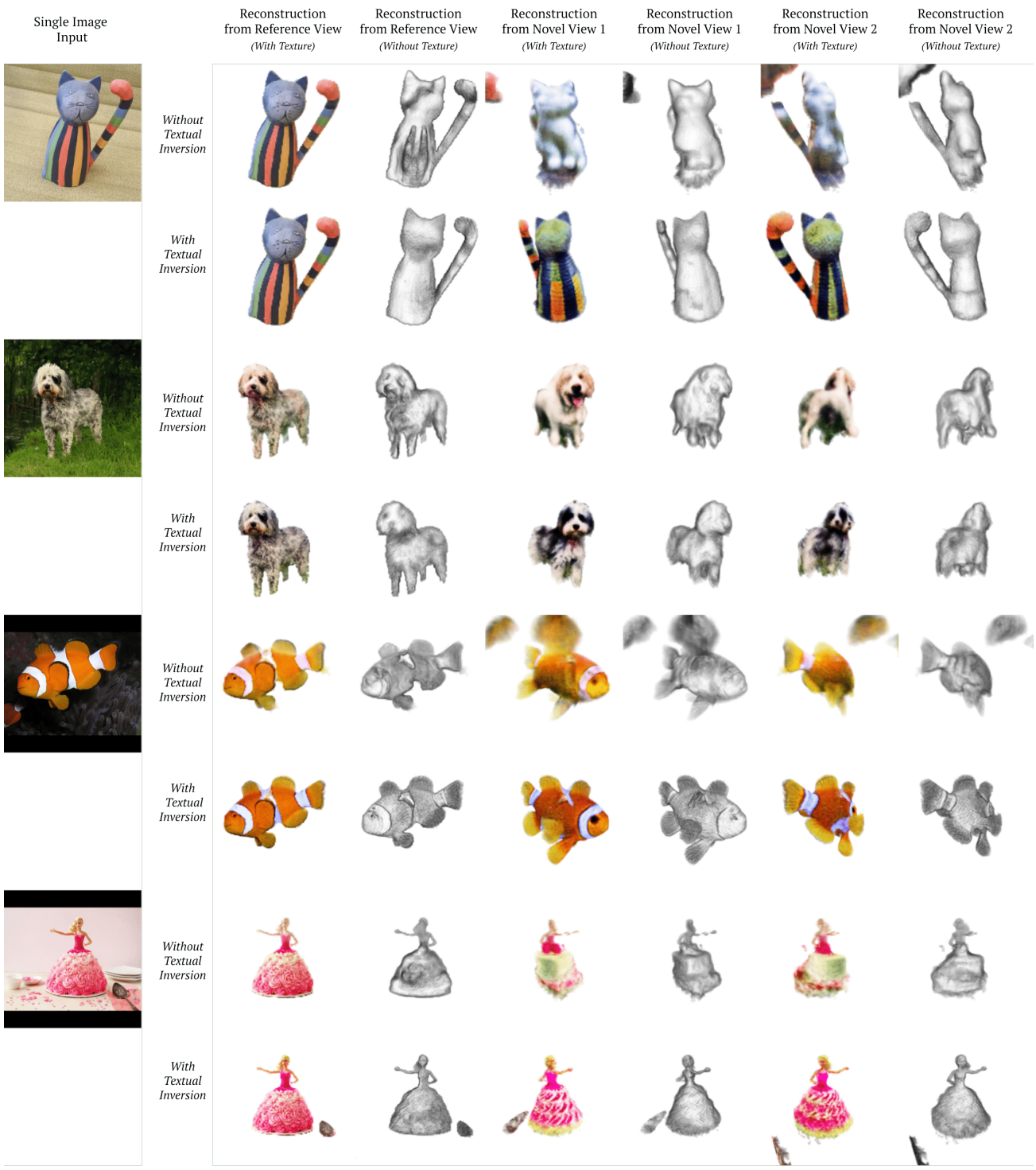

Figure 5. **A visualization of the effect of single-image textual inversion on reconstruction quality.** An expanded version of Figure 7 in the main paper showing the effect of single-image textual inversion on reconstruction quality. The top row in each pair of rows shows reconstruction results using a standard text prompt, whereas the bottom row shows reconstruction results using single-image textual inversion. The novel views are chosen to show the back side of the object; note how the examples without textual inversion look like highly-generic versions of the objects in the input image.

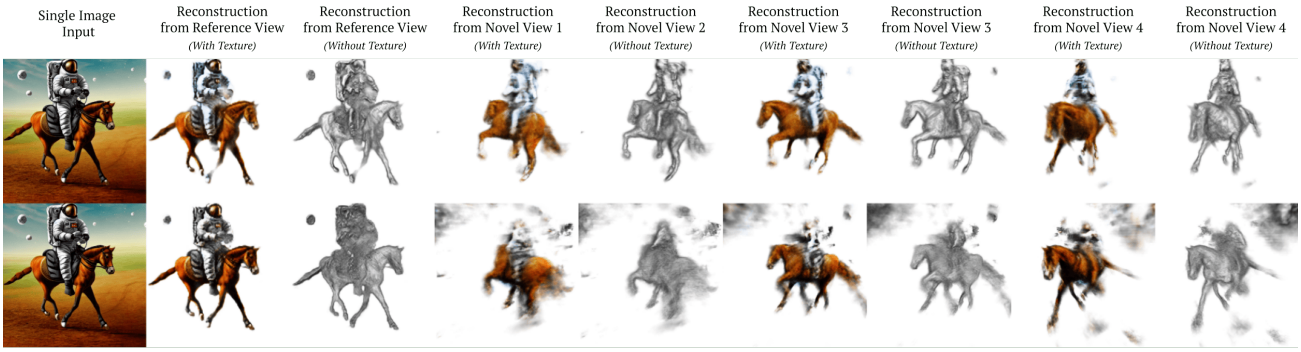

Figure 6. **An example of variation across random seeds for a challenging input image.** As described in the main paper, our model is able to generate multiple reconstructions for a given input image. For this figure, we apply our method (in a text-to-image-to-3D manner) to a highly challenging image produced by Stable Diffusion from the text prompt "An image of an astronaut riding a horse." We run reconstruction using two different seeds: one of these (top) yields a reasonable shape, whereas the other is a failure case that does not yield a reasonable shape. This example both highlights the ability of our method to reconstruct highly challenging shapes and also demonstrates how future work could aim to improve reconstruction consistency and quality.

References

[1] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proc. ICCV*, 2021. 4

[2] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised NeRF: Fewer views and faster training for free. In *Proc. CVPR*, June 2022. 2

[3] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998. 2

[4] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion. *arXiv preprint arXiv:2208.01618*, 2022. 2

[5] Ajay Jain, Matthew Tancik, and Pieter Abbeel. Putting nerf on a diet: Semantically consistent few-shot view synthesis. In *Proc. ICCV*, pages 5885–5894, October 2021. 2

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2

[7] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 2

[8] Thomas Muller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. 2

[9] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv.cs*, abs/2209.14988, 2022. 1, 2, 3

[10] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *arXiv:1907.01341*, 2019. 4

[11] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proc. CVPR*, 2022. 2, 6

[12] 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 *Proc. CVPR*, pages 5481–5490. IEEE, 2022. 2

[13] 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. In *Proc. ECCV*, 2022. 2

[14] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proc. CVPR*, pages 4578–4587, 2021. 2

[15] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 4