In-N-Out: Faithful 3D GAN Inversion with Volumetric Decomposition for Face Editing
Supplementary Material

In this supplementary material, we present additional visual results and implementation details to complement the main paper.

1. Overview

We include the following contents as our supplementary material:
- The implementation details of our proposed approach. Our code and models will be publicly available upon publication.
- Additional results, organized in our project page - https://in-n-out-3d.github.io/.

2. Implementation details

This section discusses further details not included in the main paper due to the page limit.

2.1. In-distribution inversion

We further present the details in Section 4.1 in the main paper.

Latent code initialization. Instead of initializing the latent codes $[w_1, \cdots, w_N], w_t \in \mathbb{R}^{14\times512}$ using an average latent code, we propose to use the nearest neighbor (NN) approach for the initialization. We show our approach in Algorithm 1. Please recall that we use $N = 1$ for single image.

Algorithm 1: NN_init(): Nearest Neighbor (NN) Initialization for latent codes

```
Input  : Image $\mathbf{I}_t$, or Video $\mathbf{V} = [\mathbf{I}_1, \cdots, \mathbf{I}_N]$, pre-trained generator $G$, camera parameters $[p_1, \cdots, p_N]$,
         perceptual loss $LPIPS()$, the number of samples $n$.
Output: Initialized latent codes $[w_1, \cdots, w_N]$.
1  for $t \leftarrow 1$ to $N$ do
2      $w_{samples} = []$
3      Distances $dists = []$
4      for $i \leftarrow 1$ to $n$ do
5          Draw $z_i \in \mathbb{R}^{512} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
6          Compute $w_i \leftarrow G.mapping(z_i, p_t)$
7          $w_{samples}$.insert($w_i$)
8          $dists$.insert($LPIPS(G(w_i, p_t), \mathbf{I}_t)$)
9      end
10     $w_t \leftarrow NearestNeighbor(w_{samples}, dists)$
11 end
12 Return mean($[w_1, \cdots, w_N]$).
```

Optimization. We use a residual form to represent the latent code of each frame.

$$w_t = w^{cano} + a w_t^{res}, \tag{1}$$

where w^{cano} is the canonical latent code, and w_t^{res} is the residual. In practice, we use $a = 0.7$.

To initialize the latent codes, we randomly sample $n = 500$ latent codes.

2.2. Out-of-domain inversion

We further explain the details in Section 4.2 in the main paper.

Tri-plane. We use the Tri-plane representation from EG3D [2]. For the out-of-distribution (OOD) object, we use tri-plane $\mathbf{T}^O \in \mathbb{R}^{256\times256\times32\times3}$ as its 3D representation. $\mathbf{T}^O$ is initialized from a standard normal distribution. Given a 3D position $\mathbf{x} \in \mathbb{R}^3$, we project it onto each of the plane and retrieve the corresponding feature vectors F_{xy}, F_{yz}, F_{xz} through a bilinear interpolation. Then three feature vectors are aggregated via summation.

Per-frame latent code ϕ_t. To encode the OOD object across different frames, we use a time-varying latent code $\phi_t \in \mathbb{R}^{32}$ for each frame. We draw $\phi_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. ϕ_t is optimized together with other variables in Section 4.2.

Decoder. For the out-of-distribution inversion, we use a two-layer Multi-Layer Perceptron (MLP) as the decoder to predict the pointwise color $\mathbf{c} \in \mathbb{R}^3$, density $\sigma \in \mathbb{R}$, and blending weight $b \in \mathbb{R}$ for the later composite rendering.

$$(\mathbf{c}^O, \sigma^O, b) = D^O(\mathbf{T}^O(t_k), \phi_t; \theta_{D^O})\,. \tag{2}$$

Optimization. During the optimization, we use the loss function Eqn. 8 in the main paper.

2.3. Overall optimization

In practice, we only optimize with the total loss function (Eqn. 8 in main paper, also shown below) for both in-distribution and out-of-distribution optimization. Our method can automatically split in-distribution and out-of-distribution radiance fields.

$$\mathcal{L}^{LR} = \sum_{t=1}^{N} \mathcal{L}_t^C + \lambda_\Delta \mathcal{L}_\Delta + \lambda_w \mathcal{L}_w + (\lambda_\mathcal{D} \mathcal{L}_\mathcal{D})\,, \tag{3}$$

2.4. Editing directions

In our paper, we apply InterfaceGAN [5] and StyleCLIP [3] to EG3D.

InterfaceGAN. InterfaceGAN requires a pre-trained classifier to label synthetic image data generated from the generator. We use CLIP [4] instead to label data. For each image, we use the text prompt "A portrait of a X face.", where "X" is the attribute, *e.g.*, smiling, old-man, and then compute the CLIP score. The CLIP scores act as the labels and we then apply InterfaceGAN to labelled image-score pairs for training. For each editing direction, we generate 500,000 image-score pairs for training.

StyleCLIP. We apply StyleCLIP mapper [3] to synthetic images generated from pretrained EG3D generator. For each editing direction, we use 50,000 synthetic images for training, and 10,000 images for validation.

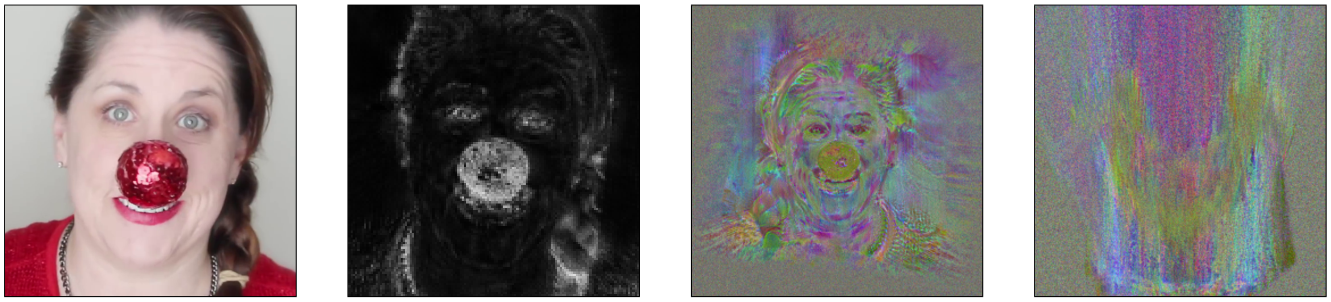

(a) Input (b) Mask (c) XY plane (d) XZ plane

Figure 1. **Visualization of learned mask and feature maps**. We visualize the learned mask and different tri-plane feature maps. The learned mask indicates the possibility of each pixel as an OOD object.

3. Additional results

3.1. Single Image Reconstruction, editing and view synthesis

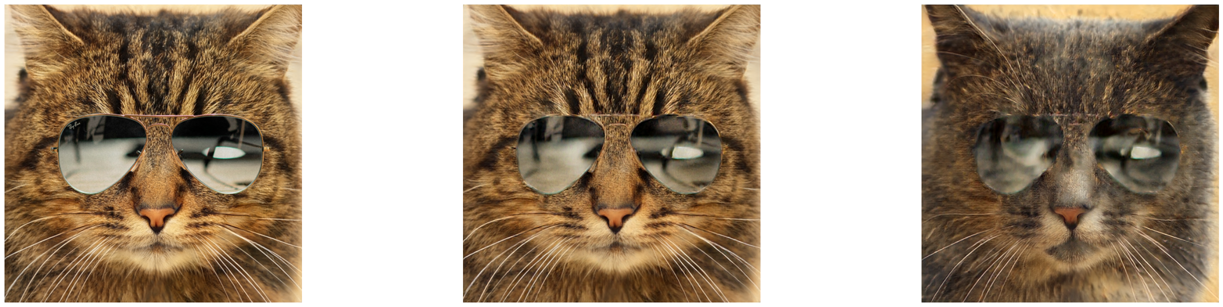
Please refer to our project page for more results.

3.2. Video Reconstruction, editing and view synthesis

We show more results in our project page. Please refer to project page.

3.3. Non-face data

Our method also applies to non-face data. Here we show an example of cat. We use the same pipeline but with a cat EG3D generator. The editing direction is “black cat” from GOAE [6].



(a) Input (b) Inv. (c) “Black cat”

3.4. OOD object removal

By setting the blending weights of the OOD objects to 0, we can remove OOD objects. Please refer to project page for more visual results.

3.5. Limitations/Failure cases

We visualize two limitations in our project page.

Floaters. Our method introduces a new tri-plane, for single image inversion, its view synthesis result may involve some “floaters”, since single-image 3D reconstruction is an ill-posed problem. Possible solution includes using distortion loss [1]. However, except for the novel views, our method outperforms other methods in terms of reconstruction quality and achieves faithful editing.

3.6. Visualization of learned masks and triplane features

We show an example in Figure 1. Please be aware of a bug in the official EG3D tri-plane implementation, where the tri-planes are actually XY, XZ, and ZX. Here we only visualize XY and XZ planes.

References

[1] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, 2021. 3

[2] Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J Guibas, Jonathan Tremblay, Sameh Khamis, et al. Efficient geometry-aware 3d generative adversarial networks. In *CVPR*, 2022. 2

[3] Or Patashnik, Zongze Wu, Eli Shechtman, Daniel Cohen-Or, and Dani Lischinski. Styleclip: Text-driven manipulation of stylegan imagery. In *ICCV*, 2021. 2

[4] 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

[5] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *CVPR*, 2020. 2

[6] Ziyang Yuan, Yiming Zhu, Yu Li, Hongyu Liu, and Chun Yuan. Make encoder great again in 3d gan inversion through geometry and occlusion-aware encoding. In *ICCV*, pages 2437–2447, 2023. 3