CS543 Final Project: Interpolation and Latent Space Walk on NeRF

Akshat Sharma
UIUC
akshat7@illinois.edu

Alan Yao
UIUC
alaney2@illinois.edu

Shengzhu Yin
UIUC
yin20@illinois.edu

Abstract

We evaluate the latent space induced by training NeRF models over multiple object instances. We learn linear walks in the NeRF latent space to determine if generative training disentangles semantically relevant attributes and learns properties such as color, zoom, and shape. We show that EditNeRF produces a latent space that is steerablet in color but can not produce zoomed-in/out object instances, which is expected based on the EditNeRF training procedure. We make our code available on GitHub.[1]

1. Introduction And Related Work

1.1. NeRF Models

NeRF models [14] use volume rendering with implicit scene representations to generate novel views of a 3D scene. NeRFs have grown in popularity due to their state-of-the-art generation quality. They have inspired much recent work in improving learned geometry [1, 7, 20], view synthesis [21, 23], training and inference speed [3, 4, 6, 22].

1.2. NeRF Background

NeRF uses a multi-layer perceptron (MLP) $\mathcal{F}_w$ as the plenoptic function, which takes $(x, y, z, \theta, \phi) \in \mathbb{R}^3 \times \mathbb{S}^2$ to $(\mathbf{c}, \sigma)$. (x, y, z) represent spatial points and angles are the viewing directions. $\mathbf{c}, \sigma$ represent the RGB color vector and the output scalar density. Querying the NeRF model for a certain viewpoint and camera angle produces the color and volume density, which are then accumulated along the ray to produce the 2D color of the ray.

As the forward pass and volumetric rendering procedures are fully differentiable, gradient descent is used to optimize the weights w of the implicit scene representation $\mathcal{F}_w$. To accumulate the color of a ray the density and color values are sampled in sets of discrete partitions along the camera ray path $\mathbf{r}(t)$, $t \in [0, 1]$. The accumulated color along the ray is given by:

[1]Code available on GitHub. Short video demo included with submission

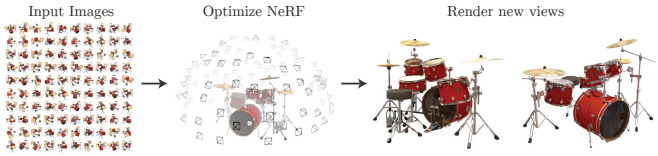


Figure 1. NeRF pipeline, figure from [14]

$$C(\mathbf{r}) = \int ds\, \mathbf{c}(\mathbf{r}(s))\sigma(\mathbf{r}(s))e^{-\int_0^s dt\, \sigma(\mathbf{r}(t))} \quad (1)$$

Here, σ is the scalar density, which can be interpreted as the differential probability that the ray terminates at $\mathbf{r}(s)$ by hitting an object particle. And $e^{-\int_0^s dt\, \sigma(\mathbf{r}(t))}$ is the probability that the ray survives up to $\mathbf{r}(s)$. This reminds one of the path integral formalism that is used in statistics and quantum mechanics except that we do not have to integrate over the possibility of all curved paths as we are assuming linear ray optics. Another way of looking at this integral is that $\sigma(\mathbf{r}(s))e^{-\int^s dt\, \sigma(\mathbf{r}(t))}$ is the probability density function (probability that propagation stops at $\mathbf{r}$) with color vector $\mathbf{c}(\mathbf{r}(s))$ [18]. In NeRF, we approximate this integral into the following form with weights in $\mathcal{F}_w$.

$$C(\mathbf{r}) \approx \hat{C}(\mathbf{r}) := \sum_i \alpha_i\, c_i\, e^{-\Sigma_{j<i}\, \sigma_j \delta_j} \quad (2)$$

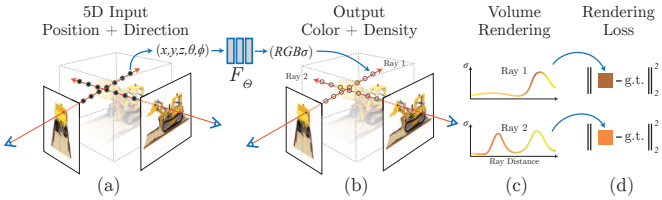


Figure 2. NeRF training pipeline, figure from [14]

Alpha-composing weights are $\alpha_i := 1 - e^{-\sigma_i\, \delta_i}$ and δ_j is the step length. The rendered view is compared against the ground truth view. See Figure 2.

$$\min_{\{w_k\}} \sum_n ||render^{(n)}(\mathcal{F}_w) - groundtruth^{(n)}||^2 \quad (3)$$

1.3. GAN latent spaces

GAN models [5] adversarially train a generator model to produce real-looking samples from noise. They have inspired much work in embedding real images into the latent domain [11, 25], traversing GAN latent spaces for controlled image synthesis [9, 10], joint code + generator training for better latent embeddings [8, 12] and disentangling latent factors via GAN training [10, 17]. In particular, we focus on [8], which measures the steerability of a GAN's latent space by attempting to learn walks in the latent space guided by manual edits to the generator output. They show that popular GAN models such as BigGAN, StyleGAN, and DCGAN [2, 10, 16] produce a steerable latent space, where trajectories in the latent space can correspond to physical world edits such as change in color, zoom, rotation, brightness, perspective, etc.

1.4. The EditNeRF Model

The EditNeRF model [13] allows one to edit and control the color and shape of the implicit neural object representation of a 3D model. EditNeRF takes as input the spatial point (x, y, z), viewing direction (θ, ϕ) and per-object shape and color codes $(\mathbf{z}^{(s)}, \mathbf{z}^{(c)})$. Hence the NeRF MLP is of the form

$$(\mathbf{c}, \sigma) = \mathcal{F}(x, y, z, \theta, \phi, \mathbf{z}^{(s)}, \mathbf{z}^{(c)}) \tag{4}$$

EditNeRF has separate color and shape feature branches, which can be updated separately. With figure 3, we can start with positional and view directional encoding $\gamma(\mathbf{x})$ and $\gamma(\mathbf{d})$. We feed this to the layers $\mathcal{F}_{share}, \mathcal{F}_{inst}$ that take as input solely the positional encoding. Next, both are fused through the network $\mathcal{F}_{fuse}$. The output from $\mathcal{F}_{fuse}$ is fed into $\mathcal{F}_{dens}, \mathcal{F}_{rad}$ to obtain scalar density σ and the radiance (color vector).

By sharing the shape branch over multiple object instances, they aim to learn a rich prior of shapes over a class of objects, e.g. chairs, cars, etc.

The EditNeRF model learns the shape and color code for each object during training. These codes are then used as inputs to produce a novel view of the object.

1.4.1 Loss Function for EditNeRF

When a user wants to edit an object's color by scribbling a local surface region, the loss function that adjusts the editing result is designed as follows: Let a set of rays that hit newly colored pixel regions, with color $\mathbf{c}_n(\mathbf{r})$ as $p_n := \{(\mathbf{r}, \mathbf{c}_n)\}$. The user can also scribble a region that should remain unchanged. Let's denote these ray locations and colors be $p_o := \{(\mathbf{r}, \mathbf{c}_o)\}$. With such user-defined input data, the loss function can be chosen as

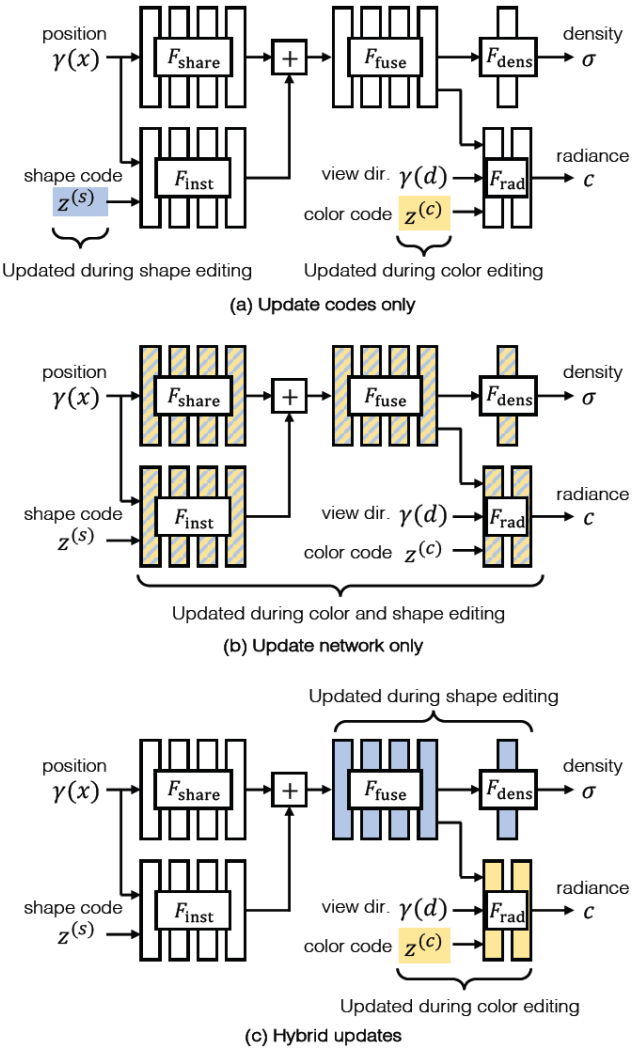


Figure 3. EditNeRF Architecture. Figure from [13]

$$\mathcal{L}_{color} = \sum_{(\mathbf{r},\mathbf{c}_n)\in p_n} ||\hat{C}(\mathbf{r}, \mathbf{z}^{(s)}, \mathbf{z}^{(c)} - \mathbf{c}_n||^2 + \sum_{(\mathbf{r},\mathbf{c}_o)\in p_o} ||\hat{C}(\mathbf{r}, \mathbf{z}^{(s)}, \mathbf{z}^{(c)} - \mathbf{c}_o||^2 + \lambda_{reg}\mathcal{L}_{reg} \tag{5}$$

One can see that the loss function minimizes when the pixel color matches maximally to user scribbles. The regularization term, which is defined as pixel color difference between iterations, is added to avoid huge deviations between each execution. In other words, this is the Lagrangian with a Lagrange multiplier of λ_{reg}.

Editing shapes has two aspects: Adding and removing materials (scalar density). Let's consider the removing part.

Using similar notation to the color editing part, take density instead of color. Let $\sigma_{\mathbf{r}}$ be the unit-normalized density values for sampled points along ray $\mathbf{r}$, and $l_n := \{(\mathbf{r}, \sigma_{\mathbf{r}})\}$. Then the loss function can be chosen as the Shannon entropy.

$$\mathcal{L}_{removal} = \mathcal{L}_{color} - \lambda_{dens} \sum_{(\mathbf{r},\sigma_{\mathbf{r}})\in l_n} \sigma_{\mathbf{r}}^T \log(\sigma_{\mathbf{r}}) \quad (6)$$

Note that the loss function is chosen this way since the entropy will be minimized when density values are close to zero vector, hence the removal effect.

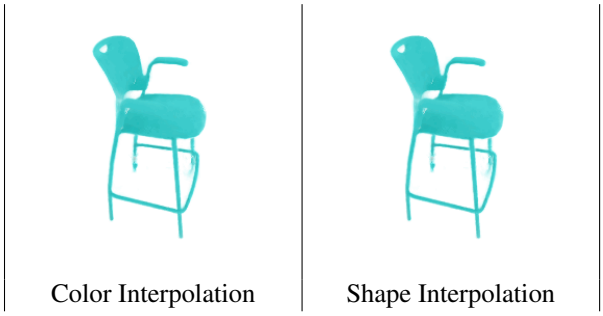
Note that EditNeRF can be interpreted as a generative decoder given the color and shape codes as input and the NeRF inference and volumetric rendering as the decoding pipeline. Due to the pipeline being fully differentiable, we attempt to determine if the learned color/shape codes are steerable using the methods of [8].

2. Approach

2.1. Manipulating color/shape codes

As argued in section 1.4, the EditNeRF model can be interpreted as a traditional generative decoder. To ascertain the same, we interpolate the color and shape codes between two object instances (see figure 4). The results are shown here. Note that EditNeRF is successfully able to interpolate between shapes and colors. (See the animation on Adobe PDF viewer.)

Figure 4. Chair data set and table data set that will act as endpoints of the interpolation experiment, Figure from [13]



2.2. Steering Generative Models in the latent space

[8] aim to learn a linear walk in the latent space, parametrized by $w \in \mathbb{R}^D$, where D represents the dimensionality of the latent code. Given a generative model $G : z \to x$, the direction w is learned by minimizing the objective function:

$$w^* = \text{argmin}_w \mathbb{E}_{z,\alpha}[L(G(z + \alpha w), \text{edit}(G(z), \alpha)] \quad (7)$$

Here, L measures the distance between the generated image after taking an α-step in the latent direction $G(z + \alpha w)$ and the target edit$(G(z), \alpha)$ derived from the source image $G(z)$. See figure 5.

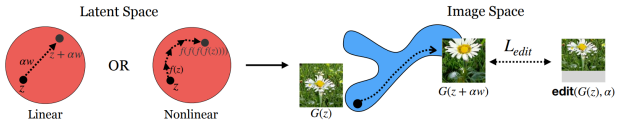


Figure 5. Taking a linear walk in z space versus a non-linear walk $G(f(f(...f(z))))$ for α times zoom, figure from [8]

We use $L2$ loss as our objective L, however, the authors report that using the LPIPS metric [24] produces similar results.

The vector α signifies the degree of transformation. For edits of type zoom, shift, rotation, and brightness α is one dimensional and scaled in $[0, 1]$.

The walk can be learned in a fully self-supervised manner, we are only required to specify the edit$(\cdot)$ operation on an image.

NeRFs are not trained over images of diverse zoom, brightness, and shifts and we do not expect their latent space to be able to represent such features. Also, NeRFs can produce images from different perspectives solely due to their rendering logic, hence we do not expect their latent space to represent perspective shifts as well. We choose to experiment with color and zoom-type edits (where we expect it to learn a walk under color edits but not the zoom edits):

1. We experiment with two types of color edits:

   (a) Color Edit I: This is closer to the editing procedure used in [8]. Given an $\alpha = (\alpha_0, \alpha_0, \alpha_0)$ we obtain the interpolated vector $z^* = z + w * \alpha$, where $z \in \mathbb{R}^D, w \in \mathbb{R}^{D\times 3}$ and $\alpha \in \mathbb{R}^3$. The editing procedure edit$(G(z), \alpha)$ multiplies each channel with the corresponding α value. While learning the walk, we perform edits by randomly sampling $\alpha_0 \in [0, 1]$. While steering EditNeRF, we generate 128 equally spaced α_0 values in $[0, 2]$.

(b) Color Edit II: The first edit procedure does not accurately represent color mixing. We propose an alternate edit procedure, where for $\alpha = (\alpha_R, \alpha_G, \alpha_B)$ we constrain $||\alpha||_2 = 1$ for α to represent an accurate color-mixing by re-weighing the color channels. The editing procedure edit$(G(z), \alpha)$ multiplies each channel with the corresponding α value (which has norm 1) to mix the color channels. While learning the walk, we perform edits by randomly sampling $\alpha \in \mathbb{R}^3$ such that $||\alpha||_2 = 1$. While steering EditNeRF, we generate 128 samples from $[1, 0, 0]$ to $[0, 1, 0]$ by linearly interpolating between the polar coordinates (θ, ϕ), which represent points on a 1-norm sphere in $\mathbb{R}^3$ by $r = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$.

2. For the zoom edit, given an $\alpha_0 \in \mathbb{R}$ we scale the image with factor $\alpha = e^{\alpha_0}$. This enables walks in the positive and negative direction in the latent space. When learning the walk, we perform edits by randomly sampling $\alpha_0 \in [-1, 1]$. While steering EditNeRF, we generate 128 equally spaced α_0 values in $[0, 2]$.

3. Results

3.1. Low-dimensional embeddings of latent codes

We plot low-dimensional embeddings of the learned codes to determine if the embeddings produce visually well-separated clusters. We use the PCA [15] and T-SNE embeddings [19]. See figure 6 for results.

Note that the embeddings contain some well-clustered instances of chairs with similar shapes and colors (marked in blue). However, some representative examples are not clustered together (marked in red).

3.2. Learning Walks in the Latent Space

3.2.1 Training curves

We monitor the loss function in eq 7 to determine if the walk vector w converges (see figure 3.2.1).

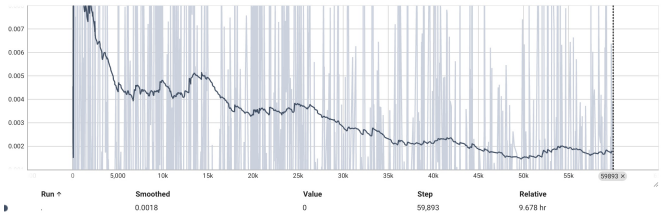


Figure 4.2.1 Training Curve for Color Edit 1, converges

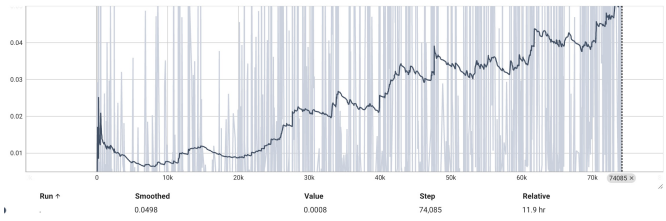


Figure 4.2.2 Training Curve for Color Edit 2, diverges

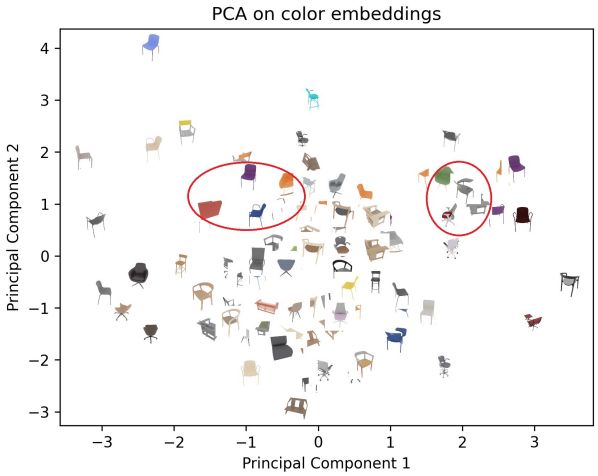
Figure 4.2.3 Training Curve for Zoom Edit, diverges

Note that only Color Edit I converges. This indicates that EditNeRF can not be steered to mix color channels and produce zoomed-in images.

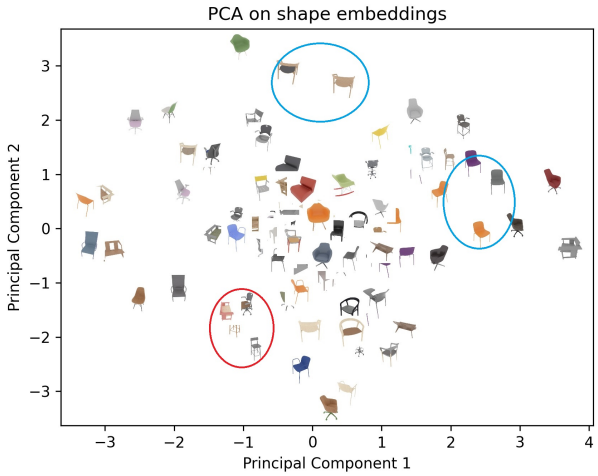
3.2.2 Walk visualizations

We attempt to steer EditNeRF under the parameters highlighted in 2.2. Note that changing the color code does not affect the object shape, this is because EditNeRF has separate shape and color branches.

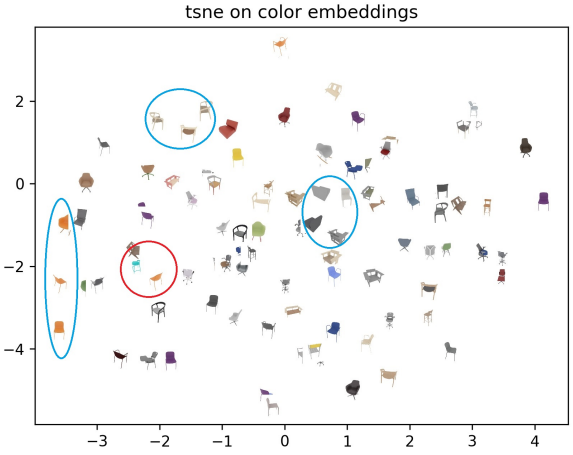
For the zoom edit, note that at the start of the interpolation, the chair size does appear to increase. However, over excessive interpolation, fine structures (thin armrests/legs) disappear. See the animation on Adobe PDF viewer.



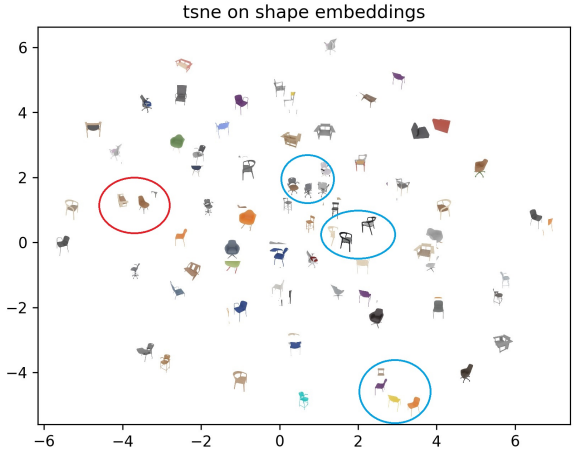
(a) PCA color embedding



(b) PCA on shape embedding



(c) T-SNE color embedding



(d) T-SNE shape embedding

Figure 6. Low-dimensional embeddings of latent codes - The blue ovals represent similar colors/shapes being clustered together, and red ovals represent dissimilar examples clustered together.

3.3. Conclusion and Future Directions

[8] show that GAN training induces steerable latent spaces even when not explicitly trained to do so. From this study, we can conclude that NeRF models don't inherently showcase this property.

Future work: Most recent work uses joint code + model training or augmenting loss functions to disentangle latent factors to learn semantically meaningful latent spaces. Updating the EditNeRF training procedure may give rise to some desirable properties of the latent space.

4. Statement of individual contribution

Akshat- Creating Editnerf interpolations 2.1, steering in the latent space 2.2, low-dimensional embeddings 3.1, learning walks in the latent space 3.2
Alan- Creating Editnerf interpolations 2.1, Steering in the latent space 2.2
Shengzhu- Steering in the latent space 2.2, Low-dimensional embeddings 3.1

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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021. 1

[2] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018. 2

[3] Boyang Deng, Jonathan T Barron, and Pratul P Srinivasan. Jaxnerf: an efficient jax implementation of nerf. *URL http://github. com/googleresearch/google-research/tree/master/jaxnerf*, 2020. 1

[4] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14346–14355, 2021. 1

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020. 2

[6] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5875–5884, 2021. 1

[7] Eldar Insafutdinov, Dylan Campbell, João F Henriques, and Andrea Vedaldi. Snes: Learning probably symmetric neural surfaces from incomplete data. In *European Conference on Computer Vision*, pages 367–383. Springer, 2022. 1

[8] Ali Jahanian, Lucy Chai, and Phillip Isola. On the" steerability" of generative adversarial networks. *arXiv preprint arXiv:1907.07171*, 2019. 2, 3, 5

[9] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017. 2

[10] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019. 2

[11] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In *International conference on machine learning*, pages 1558–1566. PMLR, 2016. 2

[12] Haozhe Liu, Wentian Zhang, Bing Li, Haoqian Wu, Nanjun He, Yawen Huang, Yuexiang Li, Bernard Ghanem, and Yefeng Zheng. Adaptivemix: Improving gan training via feature space shrinkage. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16219–16229, 2023. 2

[13] Steven Liu, Xiuming Zhang, Zhoutong Zhang, Richard Zhang, Jun-Yan Zhu, and Bryan Russell. Editing conditional radiance fields. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021. 2, 3

[14] 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, 2020. 1

[15] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901. 4

[16] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 2

[17] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9243–9252, 2020. 2

[18] Andrea Tagliasacchi and Ben Mildenhall. Volume rendering digest (for nerf), 2022. 1

[19] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008. 4

[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. 1

[21] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5438–5448, 2022. 1

[22] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021. 1

[23] Jian Zhang, Yuanqing Zhang, Huan Fu, Xiaowei Zhou, Bowen Cai, Jinchi Huang, Rongfei Jia, Binqiang Zhao, and Xing Tang. Ray priors through reprojection: Improving neural radiance fields for novel view extrapolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18376–18386, 2022. 1

[24] 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. 3

[25] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation. *Advances in neural information processing systems*, 30, 2017. 2