Unsupervised Learning of Efficient Geometry-Aware Neural Articulated Representations: Supplemental Material

Atsuhiro Noguchi[1] Xiao Sun[2] Stephen Lin[2] Tatsuya Harada[1,3]

[1] The University of Tokyo [2] Microsoft Research Asia [3] RIKEN

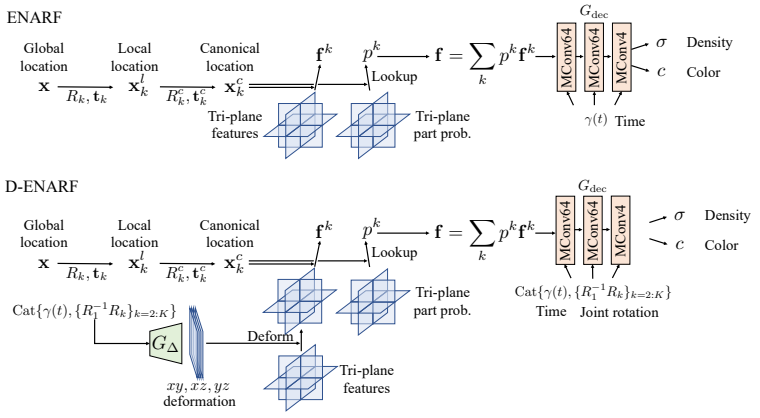

Fig. A. Network details of ENARF and D-ENARF. MConv denotes Modulated Convolution [10].

A ENARF Implementation details

A.1 Model Details

Figure A illustrates the learning pipeline of our ENAFR and D-ENAFR models. The decoder network G_{dec} is implemented with a modulated convolution as in [10]. For the ENARF model, G_{dec} is conditioned on the time input t with positional encoding $\gamma(*)$ to handle time-dependent deformations. Following [13], we use 10 frequencies in the positional encoding. For the D-ENARF model, rotation matrices are additionally used as input to handle pose-dependent deformations. To reduce the input dimension, we input the relative rotation matrices for each part to the root part R_1. To efficiently learn the deformation field in our D-ENARF model, additional tri-plane deformation Δ are learned with a StyleGAN2 [10] based generator G_{Δ},

$$\Delta = G_{\Delta}(\text{Cat}\{\gamma(t), \{R_1^{-1}R_k\}_{k=2:K}\}). \tag{1}$$

Each channel represents the relative transformation from the canonical frame in pixels for each tri-planes. The learned deformation field is used to deform the

canonical features to handle the non-rigid deformation for each time and pose. The deformed tri-plane feature F' is formulated as,

$$\begin{aligned} F'_{xy}(\mathbf{x}) &= F_{xy}([\mathbf{x}_x + \Delta_{xy}(\mathbf{x})_x, \mathbf{x}_y + \Delta_{xy}(\mathbf{x})_y, \mathbf{x}_z]) \\ F'_{yz}(\mathbf{x}) &= F_{yz}([\mathbf{x}_x, \mathbf{x}_y + \Delta_{yz}(\mathbf{x})_y, \mathbf{x}_z + \Delta_{yz}(\mathbf{x})_z]) \\ F'_{xz}(\mathbf{x}) &= F_{xz}([\mathbf{x}_x + \Delta_{xz}(\mathbf{x})_x, \mathbf{x}_y, \mathbf{x}_z + \Delta_{xz}(\mathbf{x})_z]), \end{aligned} \tag{2}$$

where $\mathbf{x} = [\mathbf{x}_x, \mathbf{x}_y, \mathbf{x}_z]$. Intuitively, $(\mathbf{x}_x, \mathbf{x}_y)$ is the 2D projection of $\mathbf{x}$ on the F_{xy} plane, and $\Delta_{xy}(\mathbf{x})$ produces the 2D translation vector for $(\mathbf{x}_x, \mathbf{x}_y)$ on the F_{xy} plane. However, this operation requires two look-up tri-planes learned from Δ and F for every 3D location $\mathbf{x}$, which is expensive. Therefore, we make an approximation of this by first computing the value of the deformed tri-plane F' at every pixel grid location using Equation 2, then sampling features from F' using Equation 7 in the main paper. The increased computational cost is thus proportional to the resolution of F', which is much smaller than the number of 3D points $\mathbf{x}$.

We used the coarse-to-fine sampling strategy as in [13] to sample the rendering points on camera rays. Instead of using two separate models to predict points at coarse and fine stages respectively [13], we use a single model to predict sampled points at both stages. Specifically, for each ray, 48 and 64 points are sampled at the coarse and fine stages, respectively.

A.2 Efficient Implementation

For efficiency, we introduce a weak shape prior for the part occupancy probability p^k in Section 3.3 of the main paper. Specifically, if a 3D location $\mathbf{x}_k^c$ in the canonical space is outside of a cube with one side of $2a$ located at the center of the part $\mathbf{p}_k^c$, we set p^k to 0, namely, $p^k \leftarrow 0$ if $\max(|\mathbf{x}_k^c - \mathbf{p}_k^c|) > a$. We set a to $\frac{1}{3}$ meter for all parts. This weak shape prior is used for all NARF-based methods. Since we do not have to compute the intermediate feature $\mathbf{f}_k$ (in Equation 7 in the main paper) for the points with part probability $p_k = 0$, the overall computational cost for feature generation is significantly reduced. We implement this efficiently by (1) gathering the valid ($p_k > 0$) canonical positions $\mathbf{x}_k^c$ and compute intermediate features $\mathbf{f}_k$ for them, (2) multiplying by p^k, and (3) summing up the feature for each part $p_k * \mathbf{f}_k$ with a scatter_add operation.

A.3 Training Details

We use the Adam [11] optimizer with an equalized learning rate [8] of 0.001. The learning rate decay rate is set to 0.99995. The ray batch sizes are set to 4096 for ENARF and 512 for NARF. The ENARF model is trained for 100,000 iterations and the NARF model is trained for 200,000 iterations with a batch size 16. The training takes 15 hours on a single A100 GPU for ENARF and 24 hours for NARF.

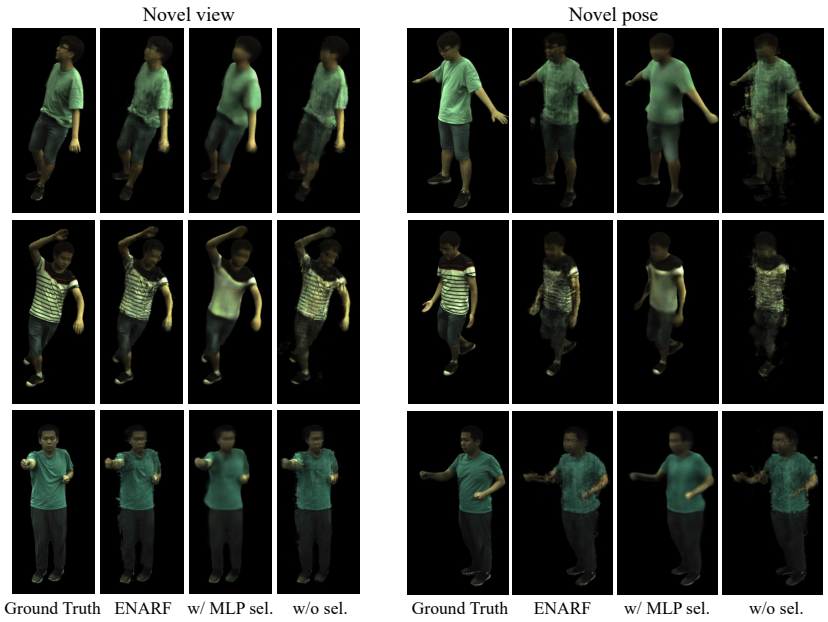

Fig. B. Ablation study on selector.

B Ablation Study on Selector (Section 4.1)

To show the effectiveness of the tri-plane based selector, we compared our model with a model without the selector and a model with an MLP based selector. In the model without a selector, we simply set $p^k = \frac{1}{K}$. In the MLP based selector, we used a two-layer MLP with a hidden layer dimension of 10 for each part, as in NARF [14]. The quantitative and qualitative comparisons are provided in Table 1 in the main paper and Figure B, respectively. The model without a selector cannot generate clear and sharp images because the feature of any location to compute the density and color is evenly contributed by all parts. The MLP based selector helps learn independent parts. However, the generated images look blurry compared to ours. Moreover, it requires much more GPU memory and FLOPS for training and testing. In summary, the proposed tri-plane based selector is superior in terms of both effectiveness and efficiency.

C Ablation Study on View Dependency

Following Efficient NeRF [4], our model does not take the view direction as input, i.e. the color is not view dependent. We note that this implementation is inconsistent with the original NeRF [13] model that takes the view direction as an input. Here, we do an ablation study on the view dependent input in our model. Specifically, the positional encoding is added to the view direction

Table A. Quantitative comparison on dynamic scenes.

	Novel view			Novel pose		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
ENARF	<u>31.94</u>	0.9655	0.04792	29.66	<u>0.953</u>	0.05702
D-ENARF	**32.93**	**0.9713**	**0.03718**	**30.06**	0.9396	<u>0.05205</u>
ENARF w/ view	29.98	0.9603	0.05129	28.42	0.9497	0.06005
D-ENARF w/ view	31.05	<u>0.9668</u>	<u>0.04162</u>	29.3	**0.9563**	**0.04711**

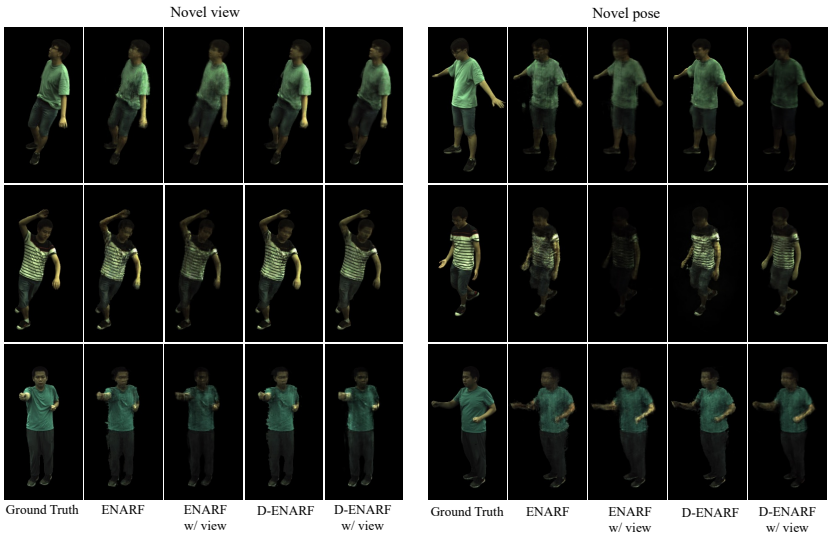

Fig. C. Ablation study on view direction.

$\gamma(\mathbf{d})$ and additionally used as the input of G_{dec}. Experimental results indicate that additional view direction input leads to darker images and degrades the performance. We thus do not use view direction as input of our models by default. Quantitative and qualitative comparisons are provided in Table A and Figure C, respectively.

D Comparison with NeuralActor

In this section, we compare ENARF with another state-of-the-art human image synthesis model NeuralActor [12]. Since the training code of NeuralActor is not publically available, we trained ENARF and D-ENARF on two sequences (S1, S2) of the DeepCap dataset [7] following NeuralActor. We then compared them with the corresponding results reported in the NeuralActor paper. NeuralActor uses richer supervision, such as the ground truth UV texture map of SMPL mesh for each frame. The qualitative results on novel pose synthesis are shown in Figure D. Without deformation modeling, ENARF tends to produce jaggy images and performs the worst due to the enormous non-rigid deforma-

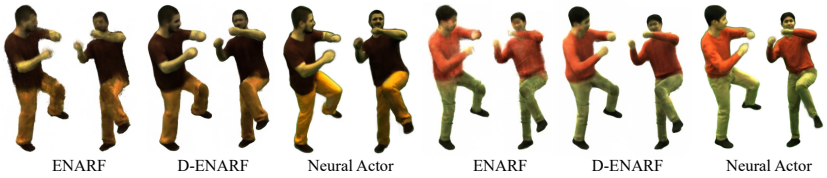

Fig. D. Qualitative results in novel pose synthesis, compared between ENARF, D-ENARF, and NeuralActor [12]. The results of NeuralActor are drawn directly from Figure 4 of the original NeuralActor paper [12]. The results of ENARF and D-ENARF are generated using similar poses and views as in NeuralActor.

tion in training frames. D-ENARF can alleviate the problem by learning the deformation field and can produce plausible results. Compared to NeuralActor, D-ENARF does not generate fine details such as wrinkles in clothing or facial expressions. Still, this gap is acceptable, given that NeuralActor uses GT SMPL meshes and textures for training. We cannot reproduce the quantitative evaluation of NeuralActor [12] (in Table 2) because some implementation details, such as the foreground cropping method, are not publicly available. We thus skip the quantitative comparison with NeuralActor.

E Implementation Details of ENAFR-GAN

E.1 Bone Region Loss $\mathcal{L}_{\text{bone}}$ (Section 3.5)

One obvious positional constraint for the foreground object is that it should be generated to cover at least the regions of bones defined by the input pose configuration. This motivates us to propose a bone region loss $\mathcal{L}_{\text{bone}}$ on the foreground mask M to facilitate model training. First, we create a skeletal image B from an input pose configuration. Examples of B are visualized in Figure E. The skeletal image B is an image in which each joint and its parent joint are linked by a straight line of 1-pixel width. The bone region loss $\mathcal{L}_{\text{bone}}$ is then defined to penalize any overlaps between the background region $(1-M)$ and the bone regions.

$$\mathcal{L}_{\text{bone}} = \frac{\sum(1-M)^2 B}{\sum B}. \tag{3}$$

We show the comparison results of training with or without $\mathcal{L}_{\text{bone}}$ in Table B and Figure F. Although the foreground image quality is comparable, Figure F shows that the generated images are not well aligned with the input pose without $\mathcal{L}_{\text{bone}}$. In Table B, we can see that PCKh@0.5 metric becomes worse without $\mathcal{L}_{\text{bone}}$.

E.2 Training Details of ENARF-GAN

We set the dimension of $\mathbf{z}_{\text{tri}}$ and $\mathbf{z}_b$ to 512, and $\mathbf{z}_{\text{ENARF}}$ to 256. We use the Adam optimizer with an equalized learning rate [8] of 0.0004. We set the batch size to

Fig. E. Examples of bone images.

Table B. Quantitative comparison on generative models. * indicates that the methods are modified from the cited papers.

	FID↓	FG-FID↓	Depth↓	PCKh@0.5↑
ENARF-GAN	22.6	21.3	8.8	0.947
ENARF-GAN w/o $\mathcal{L}_{\text{bone}}$	24.3	21.0	14.8	0.863

12 and train the model for 300,000 iterations. The training takes 4 days on a single A100 GPU. In the testing phase, a 128×128 image is rendered in 80ms (about 12 fps) using a single A100 GPU.

E.3 Shifting Regularization

To prevent the background generator from synthesizing the foreground object, we apply shifting regularization [3] for the background generator. We randomly shift and crop the background images before overlaying foreground images on them. If the background generator synthesizes the foreground object, the shifted images can be easily detected by the discriminator D, which encourages the background generator to focus only on the background. We generate 128×256 background images and randomly crop 128×128 images.

F Pose Consistency Metric

We follow the evaluation metric in the contemporary work GNARF [2] to evaluate the consistency between the input pose and the pose of the generated image. Specifically, we use an off-the-shelf 2D keypoint detector [5] pre-trained on the MPII human pose dataset [1] to detect 2D keypoints in both generated and real images with the same poses and compare the detected keypoints under the metric of PCKh@0.5 [1]. We discard keypoints with low detection confidence (< 0.8) and only compare keypoints that are confident in both generated and real images.

G Truncation Trick (Section 4.2)

The truncation trick [9] can improve the quality of the images by limiting the diversity of the generated images. Figure G shows the results of generating im-

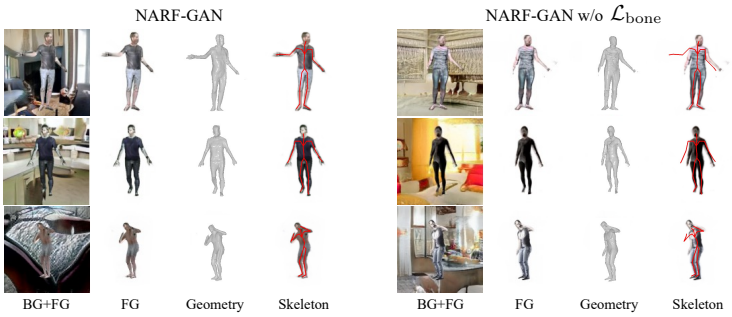

Fig. F. Ablation study on $\mathcal{L}_{\text{bone}}$. From left to right, generated images, foreground images, geometry, and foreground images with input skeletons are visualized.

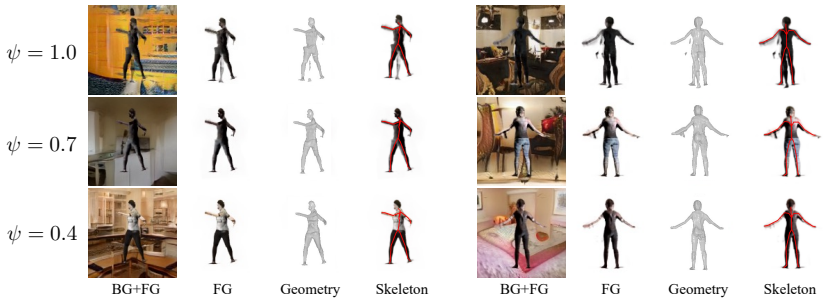

Fig. G. Truncation trick.

ages with the truncation ψ for the tri-plane generator G_{tri} set to 1.0, 0.7, and 0.4. When truncation ψ is set to 1.0, multiple legs and arms will be generated. Smaller ψ helps generate more plausible appearance and shapes of the object.

H StyleNARF (Section 4.2)

StyleNARF is a combination of NARF [14] and StyleNeRF [6]. To reduce the computational complexity, it first generates low-resolution features with NARF and then upsamples the features to a higher resolution with a CNN-based generator. Similar to ENARF, StyleNARF can only generate foreground objects, so a StyleGAN2 based background generator G_{b} is used as in ENARF-GAN. First, we sample latent vectors from a normal distribution, $\mathbf{z} = (\mathbf{z}_{\text{NARF}}, \mathbf{z}_{\text{b}}, \mathbf{z}_{\text{up}}) \sim \mathcal{N}(0, I)$, where $\mathbf{z}_{\text{NARF}}$ is a latent vector for NARF, $\mathbf{z}_{\text{b}}$ is a latent vector for background, and $\mathbf{z}_{\text{up}}$ is a latent vector for the upsampler. Then the NARF model G_{NARF} generates low-resolution foreground feature $\mathbf{F}_f$ and mask $\mathbf{M}_f$, and G_{b} generates background feature $\mathbf{F}_b$.

$$\mathbf{F}_f, \mathbf{M}_f = G_{\text{NARF}}(\mathbf{z}_{\text{ENARF}}, \{l_k, R_k, \mathbf{t}_k\}_{k=1:K}), \mathbf{F}_b = G_{\text{b}}(\mathbf{z}_b). \tag{4}$$

The foreground and background are combined using the generated foreground mask $\mathbf{M}_f$ at low resolution.

$$\mathbf{F} = \mathbf{F}_f + \mathbf{F}_b * (1 - \mathbf{M}_f). \tag{5}$$

The upsampler G_{up} upsamples the feature $\mathbf{F}$ based on the latent vector $\mathbf{z}_{\text{up}}$ and generates the final output $\mathbf{C}$.

$$\mathbf{C} = G_{\text{up}}(\mathbf{F}, \mathbf{z}_{\text{up}}). \tag{6}$$

All layers are implemented with Modulated Convolution [10].

I Datasets

We use images at resolution 128×128 for GAN training.

I.1 SURREAL

We crop the first frame of all videos to 180×180 and resize them to 128×128 so that the pelvis joint is centered. 68033 images are obtained.

I.2 AIST++

The images are cropped to 600×600 so that the pelvis joint is centered and then resized to 128×128. We sample 3000 frames for each subject, resulting in 90000 images in total.

I.3 MSCOCO

First, we select the persons whose entire body is almost visible according to the 2D keypoint annotations in MSCOCO. Each selected person is cropped by a square rectangle that tightly encloses the person and it is resized to 128×128. The number of collected samples is 38727.

J Pose Distribution

Two pose distributions are used in our experiments. One is the CMU pose distribution, which consists of 390k poses collected in the CMU Panoptic dataset. We follow the pose pre-processing steps in the SURREAL dataset, where the distance between the person and the camera is randomly distributed in a normal distribution of mean 7 meters and variance 1 meter. The pose rotation around the z-axis is uniformly distributed between 0 and 2π.

Another pose distribution used in our experiments is a random pose distribution. First, a multivariate normal distribution is fitted to the angle distribution of each joint under the 390k pose samples of the CMU Panoptic dataset. Then, poses are randomly sampled from the learned multivariate normal distribution and randomly rotated so that the pelvis joint is directly above the medial points of the right and left plantar feet.

References

1. Andriluka, M., Pishchulin, L., Gehler, P., Schiele, B.: 2D human pose estimation: New benchmark and state of the art analysis. In: CVPR (2014)
2. Bergman, A.W., Kellnhofer, P., Wang, Y., Chan, E.R., Lindell, D.B., Wetzstein, G.: Generative neural articulated radiance fields. arXiv preprint arXiv:2206.14314 (2022)
3. Bielski, A., Favaro, P.: Emergence of object segmentation in perturbed generative models. In: NeurIPS (2019)
4. Chan, E.R., Lin, C.Z., Chan, M.A., Nagano, K., Pan, B., De Mello, S., Gallo, O., Guibas, L., Tremblay, J., Khamis, S., et al.: Efficient geometry-aware 3D generative adversarial networks. In: CVPR (2022)
5. Contributors, M.: Openmmlab pose estimation toolbox and benchmark. https://github.com/open-mmlab/mmpose (2020)
6. Gu, J., Liu, L., Wang, P., Theobalt, C.: StyleNeRF: A style-based 3D aware generator for high-resolution image synthesis. In: ICLR (2022)
7. Habermann, M., Xu, W., Zollhoefer, M., Pons-Moll, G., Theobalt, C.: Deepcap: Monocular human performance capture using weak supervision. In: CVPR (2020)
8. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of gans for improved quality, stability, and variation. In: ICLR (2018)
9. Karras, T., Laine, S., Aila, T.: A style-based generator architecture for generative adversarial networks. In: CVPR (2019)
10. Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., Aila, T.: Analyzing and improving the image quality of stylegan. In: CVPR (2020)
11. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (2015)
12. Liu, L., Habermann, M., Rudnev, V., Sarkar, K., Gu, J., Theobalt, C.: Neural actor: Neural free-view synthesis of human actors with pose control. In: ACM SIGGRAPH Asia (2021)
13. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: NeRF: Representing scenes as neural radiance fields for view synthesis. In: ECCV (2020)
14. Noguchi, A., Sun, X., Lin, S., Harada, T.: Neural articulated radiance field. In: ICCV (2021)