Supplementary Materials. MoRF: Mobile Realistic Fullbody Avatars from a Monocular Video.

In the supplementary materials, we show our results for all 10 people from our self-captured dataset and one person from PeopleSnapshot. Additionally, we provide the results we acquired with NeuMan [26] on one of our 10 sequences. We also describe implementation of our method's ingredients: texture initialization, automatic frame filtering, optimization procedure and the architecture of all of the system components. Finally, we provide the details of our re-implementation of ANR [52], in particular, the split optimization technique proposed there. Please refer to our supplementary video showing avatars in action to compare temporal stability of MoRF and that of the baselines.

A. Detailed qualitative results

Figure A1 shows an example of the training scenario that we used to capture 10 identities for our dataset. Figures A5, A6, A7, A8, A9 show some of the hold out frames of all the 10 identities. For each of those we provide the results produced by MoRF as well as the results of StylePeople and ANR.

Figure A2 demonstrates the results for NeuMan [26] on one of our self-captured sequences and of MoRF on one more identity from PeopleSnapshot (in the main text we present the results for two others). NeuMan aims to construct a scene model and there is camera movement in their scenario, whereas in our case the camera is static. We suggest that this difference in capture scenarios is the reason for the below par results of NeuMan on our data (see Figure A2 - *left*).

Figure A1. Example of the training scenario that we use to capture all 10 training videos. A person slowly shows hands in front of their torso and makes a full turn in a neutral pose.

B. Detailed method implementation

Neural texture initialization (Section 3.2) Refer to Figure A3 for the visualization of example channels of the spectral texture initialization. By assigning numbers of opposite sign (red and blue indicate positive and negative values correspondingly) to different body parts, all the 16 texture channels together encode unique spectral coordinates for each of the points on a mesh surface. This forces the rendering network to separately learn translations from latent space into color for different body parts. Thus, it slows down overfitting and increases the effectiveness of the texture warping network.

Figure A2. Additional results: NeuMan on our dataset *(left avatar)*, MoRF on PeopleSnapshot *(right avatar)*

Figure A3. Example channels of the spectral neural texture initialization.

Mesh fitting improvements: Automatic frames filtering Here were describe our heuristic to automatically filter out the frames with inaccurate mesh fitting (introduced in Section 4.1). Having finished the mesh estimation process, we compare dot-products of normalized pose vectors $\mathbf{p}$ for all the pairs of subsequent frames i and $i+1$. If the cosine distance falls below a certain threshold ξ_1, we discard the $i+1$-th frame as an outlier. Anticipating that there might be a series of badly estimated frames, we compare subsequent frames starting from the $i+2$-th against the last good frame i, discarding the frames until the dot-product is above ξ_2. In short, a series of frames $\{i+1, ..., i+k\}$ is discarded if:

$$\frac{\mathbf{p}_i}{\|\mathbf{p}_i\|} \cdot \frac{\mathbf{p}_{i+1}}{\|\mathbf{p}_{i+1}\|} \leqslant \xi_1 \quad \frac{\mathbf{p}_i}{\|\mathbf{p}_i\|} \cdot \frac{\mathbf{p}_{i+1+j}}{\|\mathbf{p}_{i+1+j}\|} \leqslant \xi_2 \ , \ \forall j < k \tag{2}$$

The described heuristic was applied for the pose vectors of Smplify-X [46] that specify body joints rotations in an axis-angle format, ξ_1 and ξ_2 were correspondingly set to 0.9 and 0.8.

Optimization We first extract 468 frames from the training one-minute videos and run our modified mesh fitting on them. In particular, we use batch size as large as the whole training set (468, all fit to a single GPU) as we found this important for optimization with respect to the shared body shape parameters and for minimizing such objectives as the silhouette loss and the temporal loss. For avatar training, we use batches of the size 6 and train for 500 epochs, except for the MoRF fine-tuning experiment, where we only train for 75 epochs. We use Adam [28] optimizer for all learned parameters. We, however, modify the Adam algorithm for the neural texture, so that exponential averages do not get updated for the texels that receive near zero gradients. It improves avatar stability, because the texels that correspond to rarely seen body parts (shoes bottom, top of the head) are less affected by Adam's momentum. We use the following weights for our losses:

$$\begin{array}{cc} \lambda_{\text{L1}} = 25 & \lambda_{\text{Percept}} = 20 \\ \lambda_{\text{GAN}} = 1 & \lambda_{\text{TV}_{\text{RGB}}} = 10^{-4} \\ \lambda_{\text{TV}_{\Delta}} = 10^{-4} & \lambda_{\Delta \to 0} = 10 \\ \lambda_{\text{Dice}} = 25 & \end{array} \tag{3}$$

We apply pseudo ground truth segmentation masks to fill the background area of the ground truth images with white color. We center and crop all the training frames to the training image resolution, and apply affine transformation augmentations in 2D image space. This includes slight translation off center, rotations between $\pm 45°$, and scale $\times 0.7$ - $\times 1.2$. We leverage Lanczos filtering during scaling. For the virtual camera that is used for rasterization of SMPL-X mesh, we apply the same transformation, but in 3D space. Unlike rasterization in full-body scale with consequent scaling in the image space, with 3D camera augmentations we achieve higher level of detail conveyed from the neural texture when scaling the ground truth up, and avoid image aliasing when scaling ground truth down.

C. Neural networks design

Further, we provide the exact architectures of the components. On the design graphs, we use notation: Conv(ic, oc, k, s, p) for 2D convolutional layers, MaxPool2d(k, s, p, d) for 2D maximum pooling layers, ConvTranspose2d(ic, oc, k, s, p) for 2D transposed convolutional layers, Linear(ic, oc) for fully-connected layers; where ic and oc are numbers of input and output channels respectively, k is kernel size, s, p and d denote stride, padding and dilation respectively (the same along both spatial dimensions). 2D Batch Normalization [23] is denoted by BatchNorm or BN. Adaptive 2D Instance Normalization [21] is denoted by AdaIn.

Texture warping network The design of the texture warping network is presented in Figure A10. A convolutional encoder-decoder architecture predicts 2-channel warping fields (xy-offsets) conditioned on a pose vector $\mathbf{p}$ of size 63 (21 joints, 3 axis rotation angles) and a frame-specifically learned vector $\mathbf{z}$ of size 128. A Multilayer Perceptron translates the condition vectors into mean μ and standard deviation σ that are applied inside Adaptive Instance Normalization [21] layer AdaIn(μ, σ) (in fact we also add 1 to all predicted values of σ, anticipating that MLP would predict near zero values). The convolutional part leverages as input 2D positional encoding $\mathbf{E}(d, x, y)$, consisting of $d = 128$ channels and spatial dimensions containing sines and cosines of different frequencies computed from meshgrid pixel coordinates x and y ($j \in \mathbb{N}$, $j < d/4$):

$$\begin{aligned} \mathbf{E}(4j, x, y) &= \sin\left(x \cdot 10^{\frac{-4 \cdot 2j}{d}}\right) \\ \mathbf{E}(4j+1, x, y) &= \sin\left(y \cdot 10^{\frac{-4 \cdot 2j}{d}}\right) \\ \mathbf{E}(4j+2, x, y) &= \cos\left(x \cdot 10^{\frac{-4 \cdot 2j}{d}}\right) \\ \mathbf{E}(4j+3, x, y) &= \cos\left(y \cdot 10^{\frac{-4 \cdot 2j}{d}}\right) \end{aligned} \tag{4}$$

All convolutional layers in the warping network have bias, LeakyReLU have slope 0.01. For technical reasons, for a batch of input data we infer warping fields one by one (as with batch size set to one), thus the network design does not include any Batch Normalization layers. As a regularization, for every training sample we project a rasterized SMPL-X mesh into the texture space, to collect texture masks of body area that is seen on the training frames. During training, we apply the masks to predicted warping fields to prevent gradients from affecting warping predictions for unseen body parts. Otherwise, for the unseen parts the warping can be abnormal, because it does not affect the training predictions, yet it decreases warping stability in the novel poses.

Neural texture As in DNR [59], we learn a hierarchy of neural texture levels of sizes: $16 \times T \times T$ pixels, for $T \in \{512, 256, 128, 64, 32, 16, 8\}$. Compared to a single-scale texture, the hierarchy yields better temporal stability, and allows conveying training signal in texels that are rarely observed during training. For each rendering of the avatars we apply bilinear upsampling for all the texture levels to a resolution of 512×512 pixels, then average the levels, infer a warping field, resample all texels using warping offsets, and finally superimpose the warped texture on the mesh.

Discriminator networks We follow PatchGAN [24] architecture. We individually train 3 identical discriminators

with design as shown in Figure A11. Each processes real or predicted images in one of the 3 scales: the original resolution, $\times 0.5$ resolution, $\times 0.25$ resolution. The discriminators output a 1-channel low resolution image of probabilities, that can be interpreted as probability of being real of multiple patches on the input image. We apply masks of pseudo-ground truth segmentation on these probability maps, then compute the Adversarial loss. Across all discriminators, we use Batch Normalization layers with affine learned parameters and momentum 0.1, and LeakyReLU with slope 0.2.

Neural rendering network Figure A12 shows designs of encoder and decoder blocks used in the rendering U-Net [54] network. Figure A13 shows the assembled neural renderer design. The encoder part essentially consists of ResNet18 [18] layers. The decoder part leverages bilinear upsampling, which compared to nearest upsampling solves a rare spontaneous visual artifact of learning highly repetitive noisy patterns on the avatar. The output of the decoder part is translated by two independent shallow convolutional networks that predict the color image and the segmentation mask of the avatar. Everywhere in the rendering network we use Batch Normalization layers with affine learnt parameters and momentum 0.1. Convolutional layers do not have bias.

D. Re-implementation of ANR

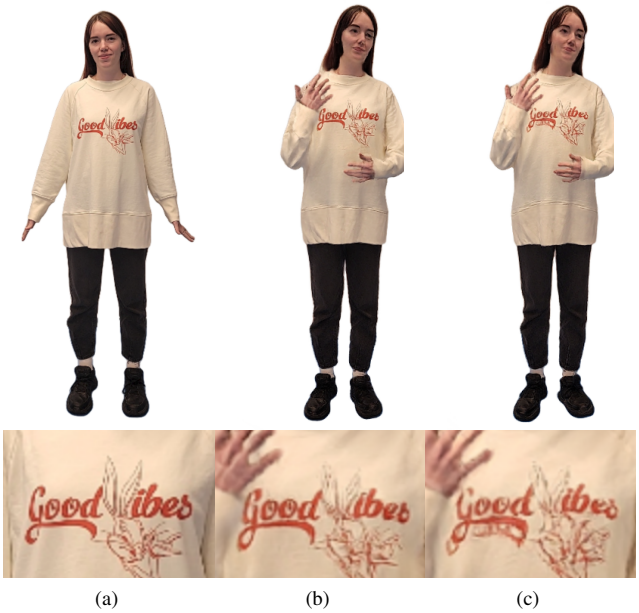
ANR [52] introduces the split optimization scheme to reduce texture averaging artifacts that happen due to ambiguity of correspondence between training images and the human body meshes fit to them. We re-implement this technique to integrate it in our pipeline.

Differently to end-to-end training of a rendering network and a neural texture (as in StylePeople [16]), in ANR the neural texture is only optimized for a sub-sample of training images (*keyframes*). The split optimization process alternates between two types of updates: (a) updates computed from the keyframes (those affect both the neural texture and the neural renderer) and (b) updates computed from all the frames (and affect only the neural renderer).

In our implementation, one minibatch for the step of type (a) is followed by one minibatch for the step of type (b). Both updates optimize the full objective as was described for MoRF in Section 4.3 (as well as for StylePeople). As the result, for the same number of texture updates, the neural renderer in ANR receives twice as large number of updates as in StylePeople optimization procedure. Differently to ANR, we use our rendering network and hierarchy of neural textures (see Section C). We thus compare the pure ability of the approaches to handle mesh fitting misalignment as well as equalize the capacity of the methods. Also notice that ANR originally trains a shared neural renderer for many subjects, while we train it for a single subject.

In Figure A4 we compare our ANR re-implementation against the StylePeople baseline on a training sequence of 864 frames, with meshes acquired via SMPLify-X [46]. The training video was recorded with our scenario as described in Section 5. We greedily picked 87 keyframes (10% of all the training frames) according to the procedure described in ANR. Our implementation of the split optimization strategy exhibits less texture averaging than StylePeople after the same number of texture updates and results in a clearer shirt pattern for the shown subject. We additionally made sure that the shown improvement is not due to the higher number of renderer updates by qualitatively comparing the results obtained after the same number of renderer updates (and, consequently, twice as small number of texture updates) as in StylePeople. We thus claim that our ANR re-implementation is valid and indeed helps compensating SMPLify-X mesh fitting misalignment on comparatively long sequences, as opposed to StylePeople that has no such ability.

To compare ANR, StylePeople and MoRF as shown in Figure 6 and Table 1, we run our ANR implementation on our data described in Section 5. For these experiments we also used 87 keyframes instead of 10% of our training images (this would be just 49 images) since such a small sub-sample size could lead to overfitting to the keyframes. Note that in the experiments shown in Figure 6 and Table 1, meshes were acquired with our mesh fitting procedure described in Section 4.1.



(a) (b) (c)

Figure A4. (a) Ground truth image. (b) Avatar trained with our ANR [52] re-implementation. (c) Avatar trained with baseline StylePeople [16], notice the greater texture averaging.

Figure A5. Hold out images with novel poses for identities 01, 02 from our dataset, and corresponding avatar images of different methods.

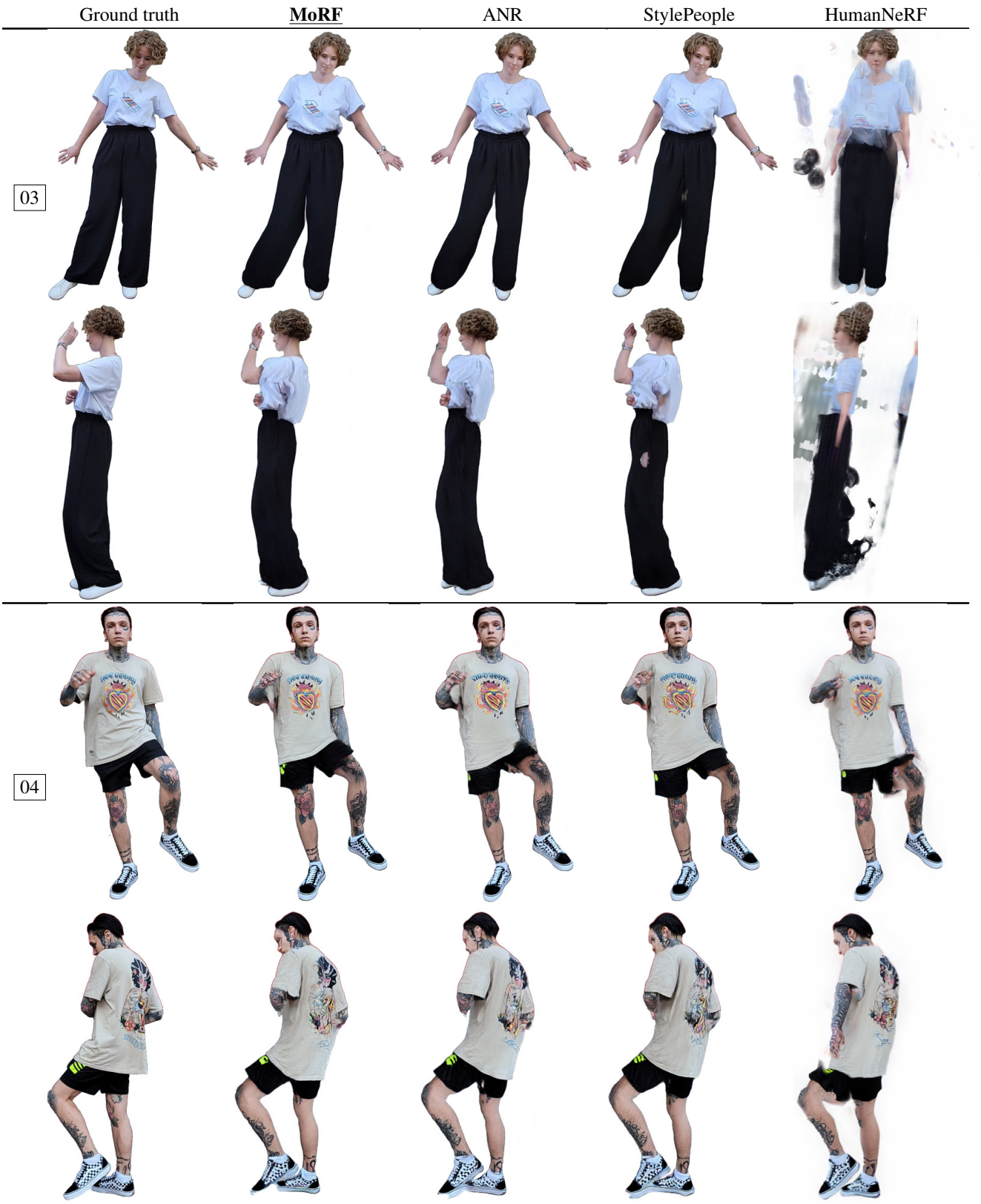


Figure A6. Hold out images with novel poses for identities 03, 04 from our dataset, and corresponding avatar images of different methods.

Figure A7. Hold out images with novel poses for identities 05, 06 from our dataset, and corresponding avatar images of different methods.

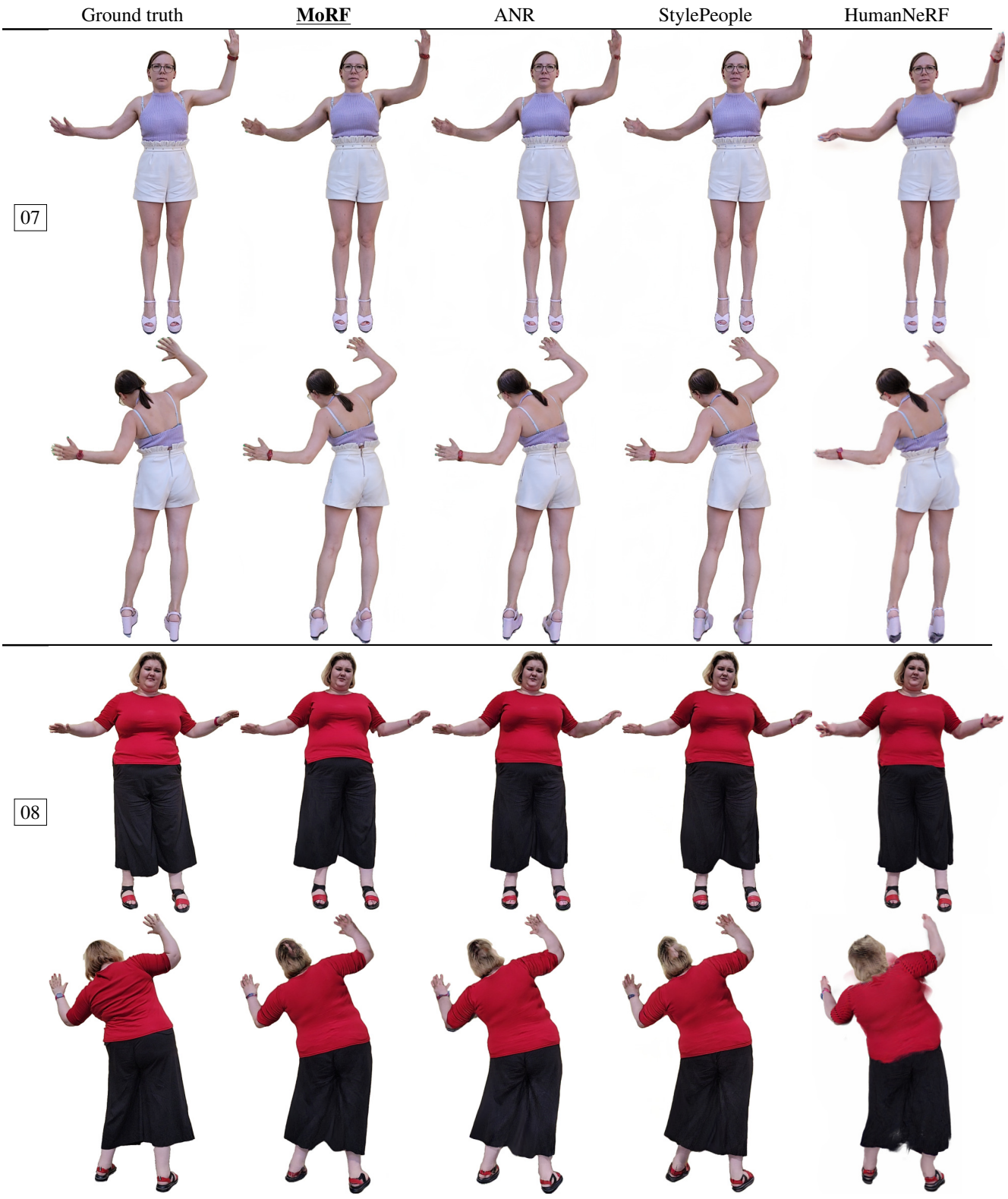


Figure A8. Hold out images with novel poses for identities 07, 08 from our dataset, and corresponding avatar images of different methods.

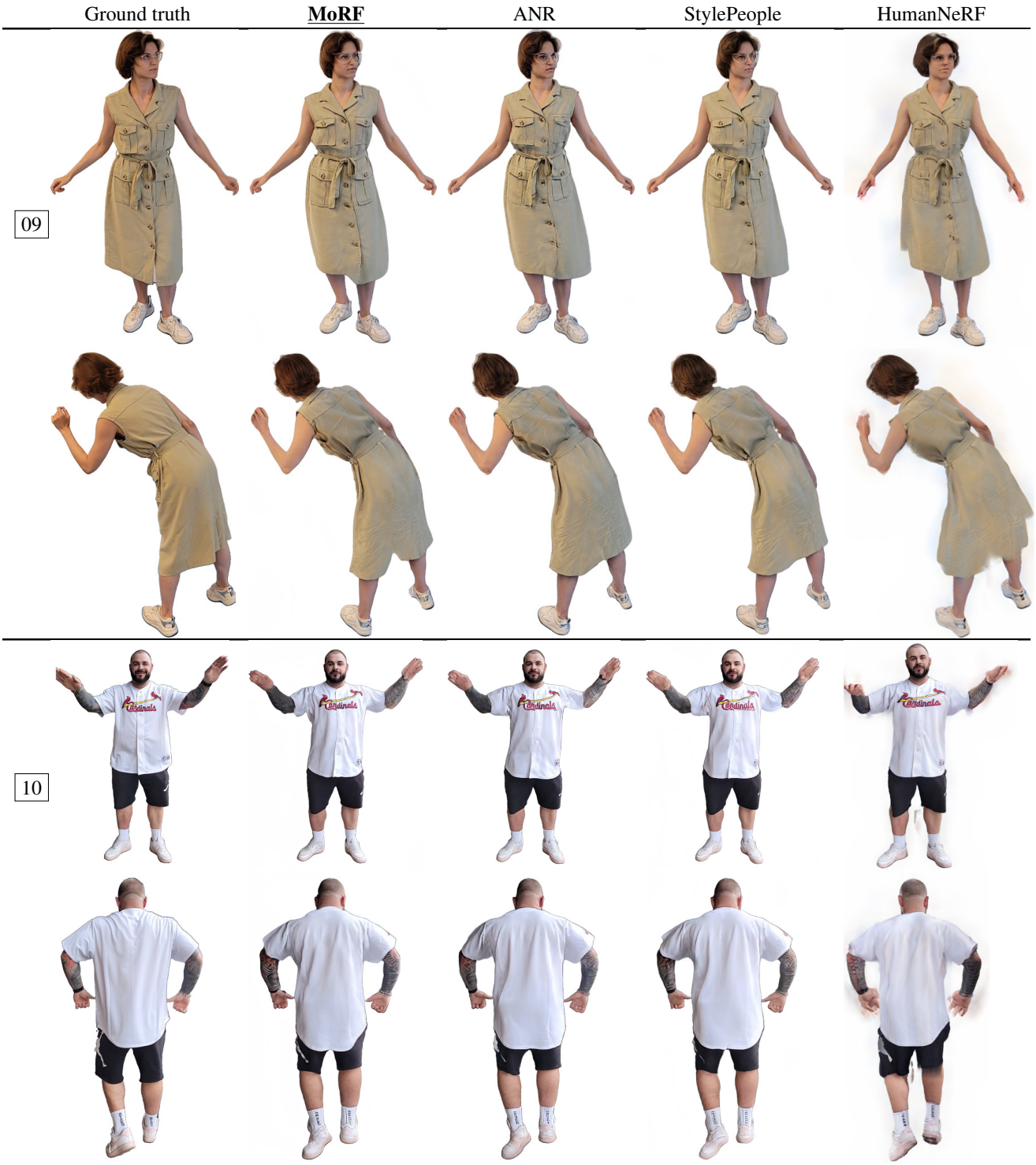


Figure A9. Hold out images with novel poses for identities 09, 10 from our dataset, and corresponding avatar images of different methods.

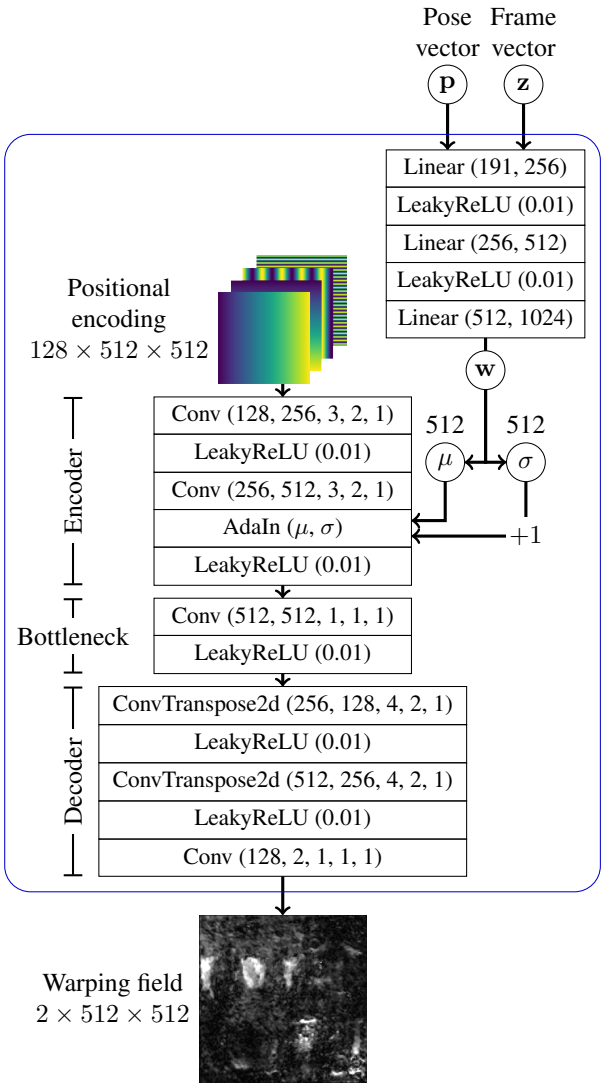


Figure A10. Design of the texture warping network.

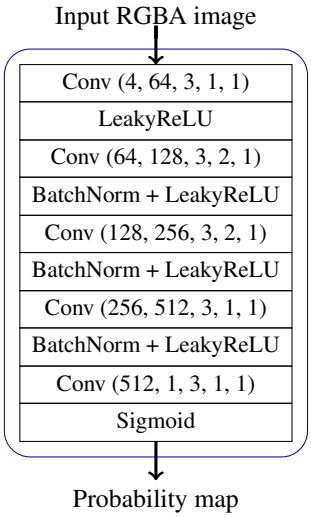


Figure A11. Design of the discriminator (PatchGAN [24]). We train 3 of those identical models to discriminate quality at the original resolution, at 0.5 downscale, and at 0.25 downscale.

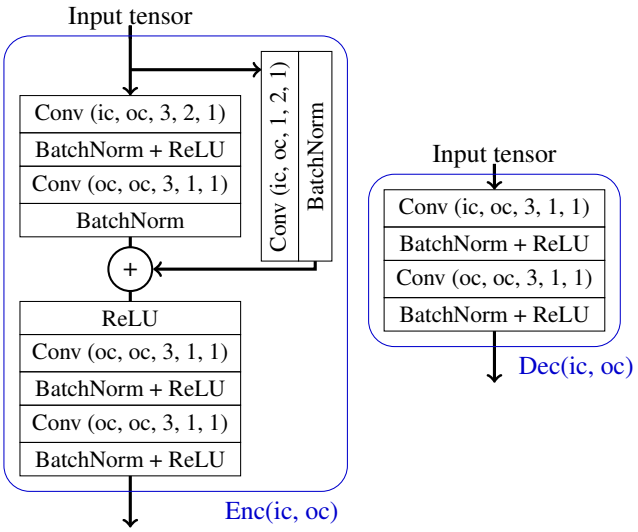


Figure A12. Design of the encoder block (`Enc`, inspired by layers of ResNet18) and decoder block (`Dec`) in the neural rendering network. `ic` and `oc` stand for the number of input and output channels of the block respectively.

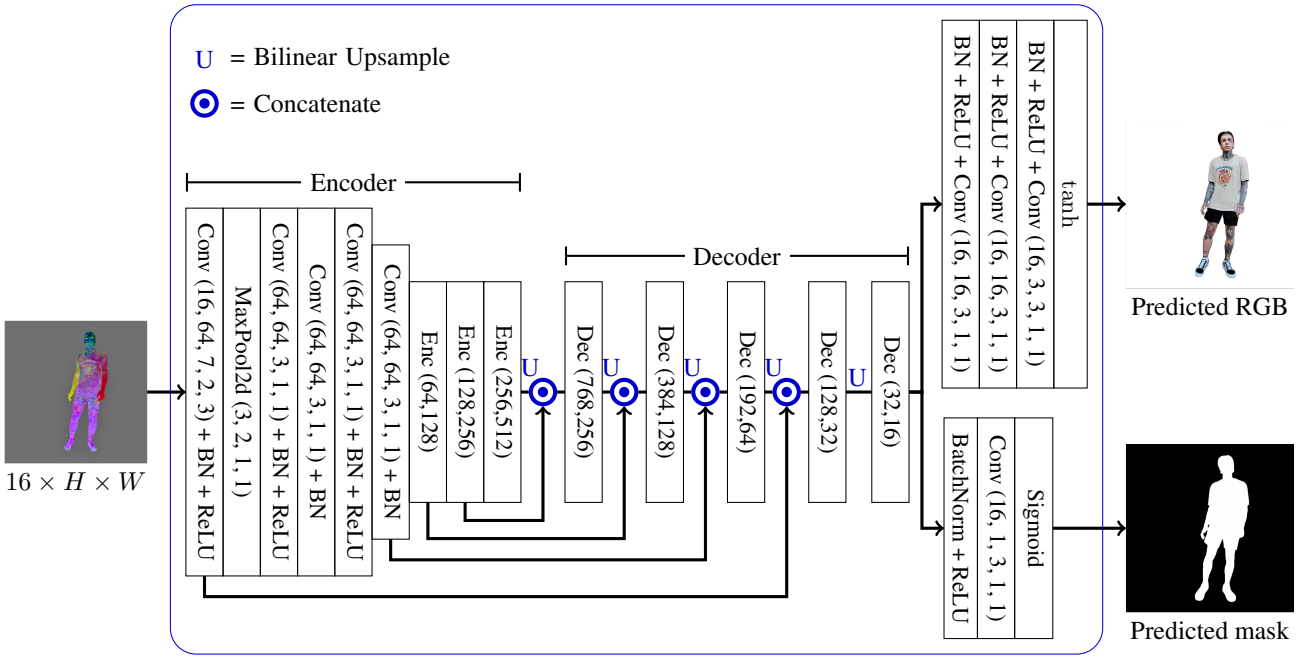


Figure A13. Neural rendering network design, which consists of a U-Net [54] and convolutional heads predicting RGB images and segmentation masks. The definitions of `Enc` and `Dec` blocks are given in Figure A12.