A. RGB sampling algorithm

In our approach, we use the combination of RGB and neural texture. To sample the RGB texture, we use the following algorithm:

Algorithm 1 RGB texture sampling algorithm

Require: $RGB(\text{size} \times \text{size} \times 3)$
Require: $UV(\text{size} \times \text{size} \times 2)$
Initialize texture with zeros
$T \leftarrow zeros(\text{texture\_size} \times \text{texture\_size} \times 3)$
$C \leftarrow zeros(\text{texture\_size} \times \text{texture\_size})$

# Fill texels with mean value of neighbors
for $\forall x, y \in [0..\text{size}]$ **do**
  $(i, j) \leftarrow UV[x, y]$
  for $\forall k, m \in [-1, 0, 1]$ **do**
    $T[i + k, j + m]$ += $RGB[x, y]$
    $C[i + k, j + m]$ += 1
  end for
end for
$T = T/C$

# Fill exact values in texels we don't need to inpaint
for $\forall x, y \in [0..\text{size}]$ **do**
  $(i, j) \leftarrow UV[x, y]$
  $T[i, j] \leftarrow RGB[x, y]$
end for

A simple filling with an average value is needed in order to remove the gaps that appear on the texture due to the discreteness of sampling grid. The described algorithm allows us to fill them taking into account the color of neighboring texels.

In order to avoid sampling errors caused by the inaccuracy of the SMPL-X fitting, we used the occlusion detector described in Section 3.5. We have shown a more thorough diagram of this stage in Figure 9.

B. RGB texture refinement

We have found it beneficial to perform several optimization steps (namely 64) of RGB texture to enhance high frequency details (Fig. 7 (e)) in the inference stage. To achieve this, we use gradients from the neural renderer derived by comparing the rendering result with the input image. Gradients are applied to texels with weights that correspond to the angles between the normal vectors and the camera direction (Fig. 3). This makes sure that only texels that can be seen in the input image are optimized with prioritization of more frontal ones. We employed $L2$ and $LPIPS$ losses to encourage color matching, and $Adversarial$ loss with regularization analogous to the 4 equation to amplify detalization.

We also apply a linear adjustment to the RGB channels of the VQGAN decoding output to improve color matching between front and back views after the inpainting stage:

$$T_{\text{rgb}} = T_{\text{rgb}}\alpha + \beta. \tag{7}$$

In this case, all texels share the trainable parameters alpha and beta. We optimize them with renderer's gradients derived by the pixels visible in the input image. As a result, the RGB channels of the neural texture at the VQGAN output strengthen color matching with the sampled RGB texture. This helps us to minimize the seam after combining textures (Fig. 7 (e)).

C. Architecture details

Encoder network. As an encoder network (Fig. 12a), we have adapted the StyleGAN2 discriminator architecture with a few changes. Namely, three images are fed to the network input: RGB, segmentation mask, and single-channel noise. Noise is introduced to provide additional freedom to the generative model when training the GAN. The efficiency of using noise in generative neural networks has been demonstrated by the authors of StyleGAN.

The images received at the input are concatenated by channels and passed through a feature extractor with an architecture equivalent to the StyleGAN discriminator consisting of ResNet blocks. We modified the model head so that it outputs a vector of length 512. This vector is then used as the input of the StyleGAN2 generator and the proposed encoder is trained end-to-end with the generator and the renderer.

Our model is trained on RGB images at the 512×512 resolution. For each $3 \times 512 \times 512$ input image, we generate a $21 \times 256 \times 256$ neural texture. In the texture, the first 16 channels are generated by the network G, the next three channels are RGB channels and the remaining two channels are the sampling and the inpainting masks.

Renderer network. Here we describe the θ renderer (Fig. 12b). The resulting texture is applied to the SMPL-X model and rasterized. The rasterized image has a size of $21 \times 512 \times 512$ and is fed to a neural renderer. The renderer takes three images as input: a rasterized SMPL-X model with a neural texture, a UV render, and a UV mask. Each input image is passed through a convolutional network consisting of two convolutions with LeakyReLU activation and BatchNorm layers. Output features are concatenated and fed into a U-Net consisting of ResNet blocks. U-Net has 3 levels connected by feature concatenation. The U-Net output is passed through two additional convolutional networks to predict the RGB image of the avatar and its mask.

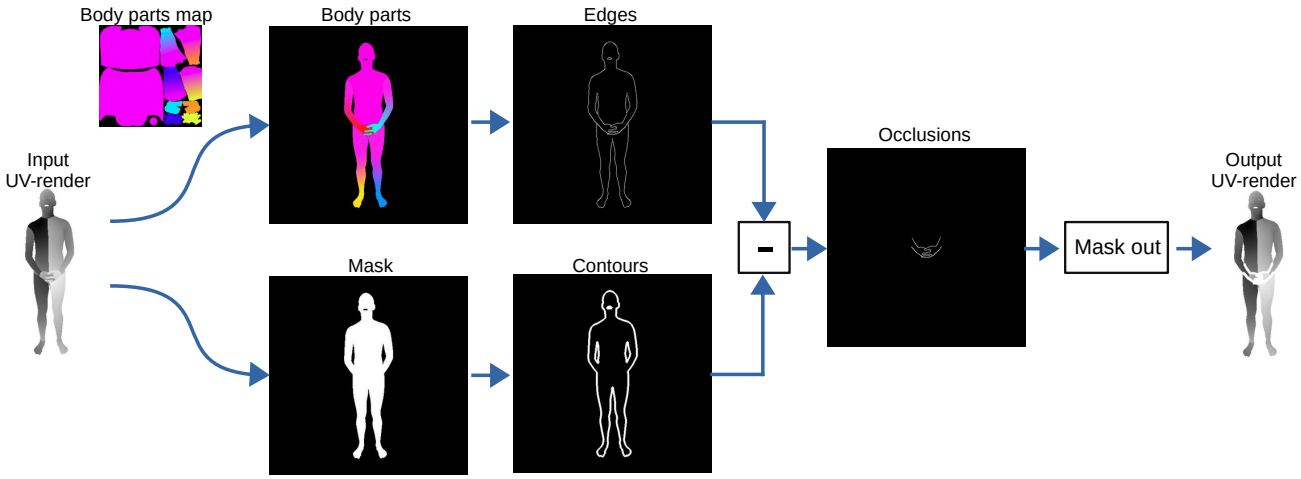


Figure 9: **Occlusions detection.** We use the body parts map as a texture to detect self-occluded areas on the avatar. From UV-render we will get rendered body parts. We get the outer silhouette of the avatar by binarization. We detect the outlines of the avatar with the edge detector. Based on the difference of contours and edges, we determine the outer contour of the occlusion area and remove it from the UV-render.

D. Robustness of the method

In order to assess the robustness of our method, we compared the methods qualitatively (Fig. 10) and quantitatively (Table 3) on the additionally dataset: THuman2.0[57]. This dataset contains 3D scans of people in diverse clothes and complex poses. For the test, we selected 25 random people and used the front-view renders as input for the one-shot methods. Our method obtains the most convincing front and back views and is resistant to complex poses and various datasets (Fig. 10). This is also confirmed by objective metrics (Table 3). Our method allows us to get the best metrics for the novel view on this additional dataset.

We also used THuman2.0 to compare with the recent S3F [9] method for generating one-shot avatars (Fig. 11). Our approach better preserves high-frequency detail in the front view and produces fewer artifacts in the back view. However, their method better restores a regular pattern on the back and allows model relightning.

E. Additional results

We present additional results of our approach on diverse data. On Fig. 14 we show results for input images containing different people. The top row shows an additional example of processing of a person in loose clothing. The next row demonstrates the high-fidelity rendering of an avatar wearing a T-shirt with a complex high-frequency print. The bottom two rows demonstrate the accuracy of avatar reconstruction from images of people in unusual poses. Also, the frames of the animation sequence show the avatars from more varied viewpoints (*e.g.* top and bottom). Invariance to the human pose is achieved through the use of a neural texture framework with a parametric model. All avatar processing, such as restoring the back, is done in canonical texture space.

On Fig. 13 we demonstrate an additional use case for our one-shot approach. We used neural network inpainting to remove the person from the original image and replace it with an animated avatar. In this way we can create the effect of a photo that has come to life.

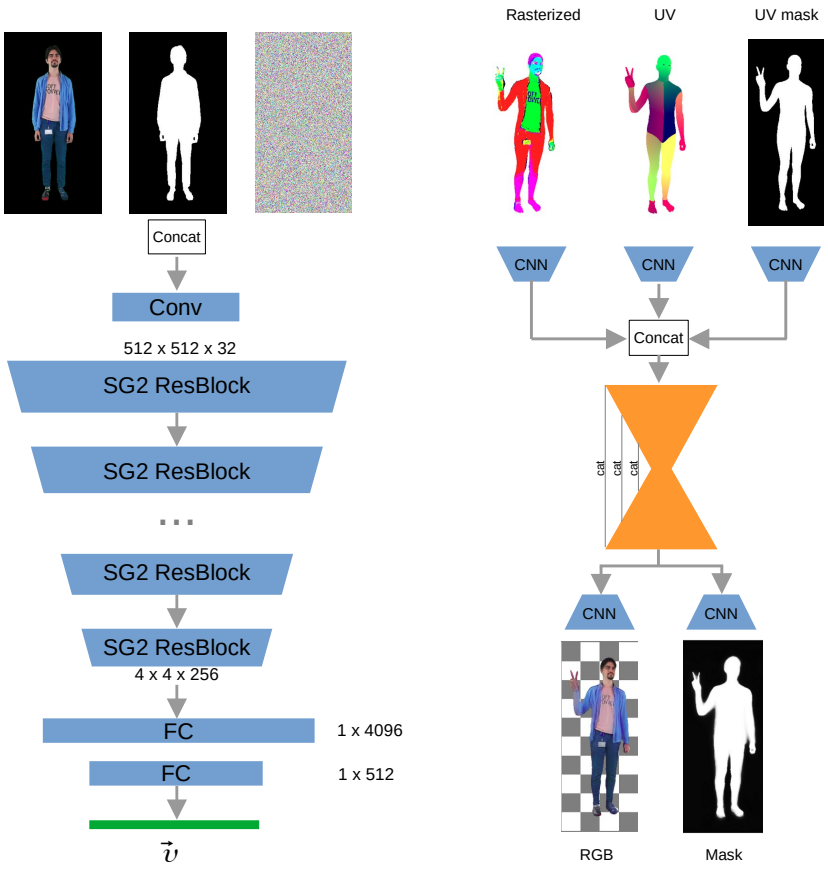
One of the limitations of the current approach is the handling of tissue deformations in the input image. Our method does not modify the textures depending on the pose, which can make the fabric look unrealistic when changing the pose. Another limitation is the insufficient sharpness of the edges of loose clothing. Even though dresses are rendered correctly by our method on most frames, the edges of the dress look unrealistic. In our future research, we would like to focus on addressing these two shortcomings.

Method	Same view				Novel view		
	MS-SSIM ↑	PSNR ↑	LPIPS ↓	DISTS ↓	DISTS ↓	ReID ↓	KID ↓
PIFu	0,9893	28,2686	0,0474	0,0912	0,2213	0.11608	0,0924
Phorhum	0,9566	23,8915	0,0521	0,1249	0,1835	0.12516	0,0389
ARCH	0,9372	21,7770	0,0645	0,1617	0,2082	0.12193	0,1065
ARCH++	0,9577	22,8318	0,0562	0,1067	0,1806	0.10108	0,0408
S3F	0,9706	25,6459	0,0497	0,1152	0,1928	0.11991	0,1108
StylePeople	0,9765	25,8120	0,0588	0,0830	0,1828	0.14212	0,0347
DINAR (Ours)	0,9600	22,8671	0,0568	0,0975	0,1607	0.09999	0,0250

Table 3: **Metrics comparison on the THuman2.0 dataset.** To demonstrate the robustness of our approach, we evaluated the metrics on a second dataset. The table is compiled similarly to Table 1 from the main paper.

Figure 10: **Results on the THuman2.0 dataset.** We compared our method with existing approaches on an additional dataset. Similar to the main article, our method shows the most convincing results for the new dataset.

Figure 11: **Comparison with S3F on the THuman2.0 dataset.** We compared our approach with the most recent one-shot approach: Structured 3D Features.



(a) **Encoder architecture** (b) **Renderer architecture**

Figure 12: **Encoder and renderer architecture.** The encoder architecture is a modified architecture of the StyleGAN2 discriminator. We changed the head to get a vector of length 512. The renderer has a U-Net architecture that predicts the RGB of an avatar from a rasterized model and UV render.

Input Image with a person replaced by an animated avatar

Figure 13: **Making photos come alive.** An additional use case of our approach is to replace the person in the photo with their animated avatar. By doing this, we can achieve the effect of an animated photo.

Figure 14: **More avatar animation examples.** We present more examples of avatar animations, including those obtained from more complex poses. The top two rows demonstrate how the approach works with people in loose clothes and clothes with highly detailed prints.