EUROGRAPHICS 2023 / K. Myszkowski and M. Nießner
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 42 (2023), Number 2

Generating Texture for 3D Human Avatar from a Single Image using Sampling and Refinement Networks

Sihun Cha[1], Kwanggyoon Seo[1], Amirsaman Ashtari[1], Junyong Noh[1]

[1]Visual Media Lab, KAIST, Republic of Korea

1. Training data for the curriculum learning

In order to train *SamplerNet* in a gradual manner, a region-wise augmentation technique is used to approximate the interpolation between the UV-aligned partial texture and the unaligned partial texture.

$T_{DensePose}$ is an unaligned partial texture used during the training when the curriculum step is 3 or higher. $T_{DensePose}$ is produced by mapping the human appearance in the source image into the UV space of the SMPL model [LMR*15] using DensePose [GNK18]. During the mapping process, $M_{DensePose}$, a binary mask that indicates valid pixels in $T_{DensePose}$, is obtained as shown in Figure 1a.

$T_{Augment}$ is used for the UV-aligned and interpolated textures. $T_{Augment}$ is produced by applying region-wise augmentation to T^M_{GT}, where T^M_{GT} is produced by masking the ground truth texture T_{GT} using $M_{DensePose}$. This process is visualized in Figure 1b. The region-wise augmentation is applied by varying control parameter α according to the curriculum step, as explained in Section 3.2.2 of the main paper. By changing α, interpolation from T^M_{GT} to $T_{DensePose}$ is approximated which enables the curriculum learning scheme.

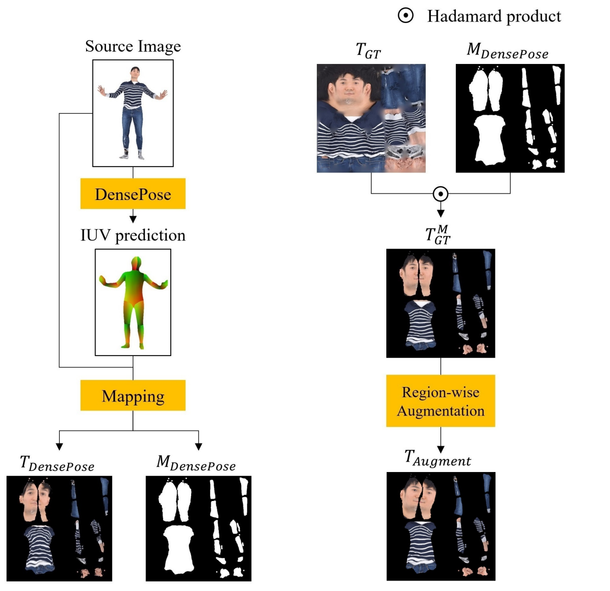
As explained in Section 3.1 of the main paper, we produce the input texture map by combining the partial texture and its mirrored counterpart, which is expressed as follows:

$$T_{input} = T_{source} + (T^{mirror}_{source} \odot (1 - M_{source})), \qquad (1)$$

where T_{source} is the partial texture created from the source image, M_{source} is a binary mask that indicates the valid pixels of T_{source}, and T^{mirror}_{source} is the mirrored counterpart of T_{source}. The training data for *SamplerNet* is produced by replacing T_{source} in Equation 1 with $T_{DensePose}$ or $T_{Augment}$ according to the curriculum step as shown in Figure 2.

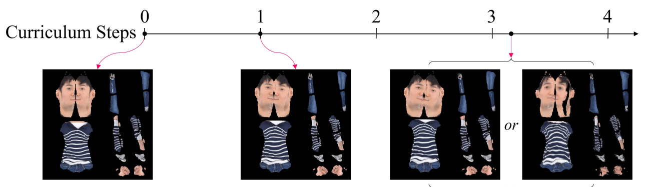
2. Network Architecture

The architecture of *SamplerNet* consists of two encoders and a single decoder. Both encoders in *SamplerNet* consist of one convolutional layer followed by five layers of residual blocks. The decoder contains five residual block layers with up sampling and one convolutional layer at the end. The residual block consists of two convolutional layers with gated convolutions [YLY*19]. One encoder of *SamplerNet* receives a normal map as input and the other encoder



(a) $T_{DensePose}$ (b) $T_{Augment}$

Figure 1: *Illustration of the acquisition process of (a) $T_{DensePose}$ and (b) $T_{Augment}$ that will be used in the curriculum learning process.*



(a) $T_{Augment}$ ($\alpha = 0$) (b) $T_{Augment}$ ($\alpha = 0.1$) (c) $T_{Augment}$ ($\alpha = 0.125$) or $T_{DensePose}$

Figure 2: *Change of T_{input} according to the curriculum step. Input texture is produced using (a) $T_{Augment}$ with $\alpha = 0$ (step=0), (b) $T_{Augment}$ with $\alpha = 0.1$ (step=1), and (c) $T_{Augment}$ with $\alpha = 0.125$ or $T_{DensePose}$ (step≥ 3).*

receives a partial texture map, T_{input}, concatenated with a visibility mask. The decoder outputs a sampling grid that is used to produce a sampled texture, T_{sample}. An overview of the architecture and the details of the network are shown in Figure 3 and Table 1, respectively.

$RefinerNet$ has a U-Net like architecture. The network consists of three down and up sampling convolution layers and 9 residual blocks in the bottleneck. The sampled texture, T_{sample}, and the occlusion mask are concatenated together and given as input to $RefinerNet$. The output of $RefinerNet$ is a blending mask M_{blend} and a refined texture map T_{refine}. The overview of the architecture and the details of the network are shown in Figure 4 and Table 2, respectively.

Both networks use LeakyReLU for activation with instance normalization. We set the negative slope for LeakyReLU as 0.2. The last layer of the decoder of $RefinerNet$ and $SamplerNet$ use sigmoid and tanh for the activation, respectively. Reflection padding was used for all padding operations in the network.

Table 1: *Architecture detail of SamplerNet.*

(a) *Appearance Encoder*

Layer	Kernel	Stride	Padding	Activation	Output
Input					4×256×256
Conv2d	7×7	1	3	LeakyReLU	32×128×128
ResBlock	3×3	2	1	LeakyReLU	64×64×64
ResBlock	3×3	2	1	LeakyReLU	128×32×32
ResBlock	3×3	2	1	LeakyReLU	256×16×16
ResBlock	3×3	2	1	LeakyReLU	512×8×8
ResBlock	3×3	2	1	LeakyReLU	1024×4×4

(b) *Geometry Encoder*

Layer	Kernel	Stride	Padding	Activation	Output
Input					3×256×256
Conv2d	7×7	1	3	LeakyReLU	32×128×128
ResBlock	3×3	2	1	LeakyReLU	64×64×64
ResBlock	3×3	2	1	LeakyReLU	128×32×32
ResBlock	3×3	2	1	LeakyReLU	256×16×16
ResBlock	3×3	2	1	LeakyReLU	512×8×8
ResBlock	3×3	2	1	LeakyReLU	1024×4×4

(c) *Decoder*

Layer	Kernel	Stride	Padding	Activation	Output
Upsample					2048×8×8
ResBlock	3×3	2	1	LeakyReLU	512×8×8
Upsample					1024×16×16
ResBlock	3×3	2	1	LeakyReLU	256×16×16
Upsample					512×32×32
ResBlock	3×3	2	1	LeakyReLU	128×32×32
Upsample					256×64×64
ResBlock	3×3	2	1	LeakyReLU	64×64×64
Upsample					128×128×128
ResBlock	3×3	2	1	LeakyReLU	32×128×128
Conv2d	3×3	1	1	Tanh	2×256×256
Output					2×256×256

Table 2: *Architecture detail of RefinerNet.*

Layer	Kernel	Stride	Padding	Activation	Output
Input					3×256×256
Conv2d	7×7	1	3	LeakyReLU	32×256×256
Conv2d	3×3	2	1	LeakyReLU	64×128×128
Conv2d	3×3	2	1	LeakyReLU	128×64×64
Conv2d	3×3	2	1	LeakyReLU	256×32×32
ResBlock	3×3	1	1	LeakyReLU	256×32×32
ResBlock	3×3	1	1	LeakyReLU	256×32×32
ResBlock	3×3	1	1	LeakyReLU	256×32×32
ResBlock	3×3	1	1	LeakyReLU	256×32×32
ResBlock	3×3	1	1	LeakyReLU	256×32×32
ResBlock	3×3	1	1	LeakyReLU	256×32×32
ResBlock	3×3	1	1	LeakyReLU	256×32×32
ResBlock	3×3	1	1	LeakyReLU	256×32×32
ResBlock	3×3	1	1	LeakyReLU	256×32×32
Upsample					256×64×64
Conv2d	3×3	1	1	LeakyReLU	128×64×64
Upsample					128×128×128
Conv2d	3×3	1	1	LeakyReLU	64×128×128
Upsample					64×256×256
Conv2d	3×3	1	1	LeakyReLU	32×256×256
Conv2d	3×3	1	1	Sigmoid	4×256×256
Output					4×256×256

References

[GNK18] GÜLER, RIZA ALP, NEVEROVA, NATALIA, and KOKKINOS, IASONAS. "Densepose: Dense human pose estimation in the wild". *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, 7297–7306 1.

[LMR*15] LOPER, MATTHEW, MAHMOOD, NAUREEN, ROMERO, JAVIER, et al. "SMPL: A skinned multi-person linear model". *ACM transactions on graphics (TOG)* 34.6 (2015), 1–16 1.

[YLY*19] YU, JIAHUI, LIN, ZHE, YANG, JIMEI, et al. "Free-form image inpainting with gated convolution". *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, 4471–4480 1.

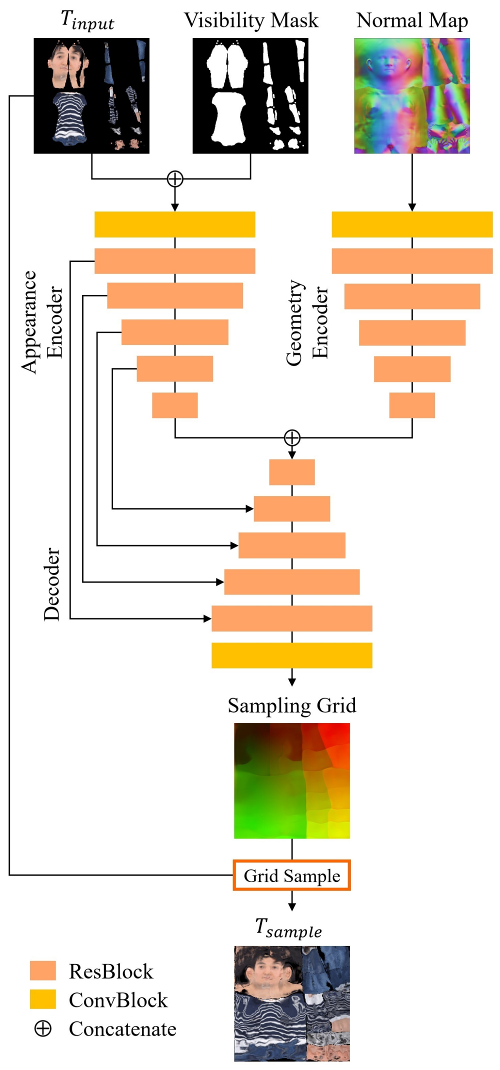


Figure 3: *The architecture of SamplerNet.*

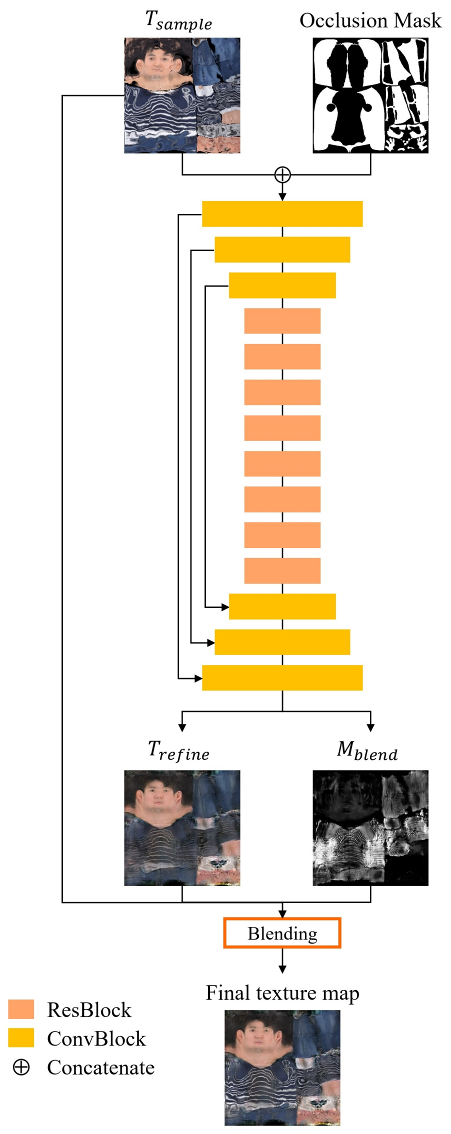


Figure 4: *The architecture of $RefinerNet$.*