TexMesh: Reconstructing Detailed Human Texture and Geometry from RGB-D Video Supplementary Material

Tiancheng Zhi[1]*, Christoph Lassner[2], Tony Tung[2], Carsten Stoll[2], Srinivasa G. Narasimhan[1], and Minh Vo[2]

[1] Carnegie Mellon University, {tzhi,srinivas}@cs.cmu.edu
[2] Facebook Reality Labs, {classner,tony.tung,carsten.stoll,minh.vo}@fb.com

1 UV Mapping

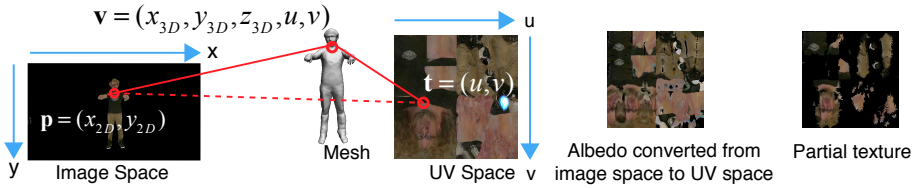


Fig. 1. The mesh bridges image space and UV space. Assume point $\mathbf{v}$ on the mesh is associated with its 3D position (x_{3D}, y_{3D}, z_{3D}) and UV coordinates (u, v). The corresponding point $\mathbf{p} = (x_{2D}, y_{2D})$ in image space can be obtained via camera projection. It also corresponds to point $\mathbf{t} = (u, v)$ in UV space. To convert features from image space to UV space, for each $\mathbf{t}$, we first obtain its 3D position (x_{3D}, y_{3D}, z_{3D}) by barycentric interpolation, project it to image coordinates (x_{2D}, y_{2D}), and finally sample features from the image.

We follow the common practice in graphics where texture is stored using a UV mapping [3]. This map unwraps a mesh into 2D space, called UV space. Each pixel $\mathbf{t} = (u, v)$ in UV space corresponds to a point $\mathbf{v}$ on the mesh. Its 3D position is defined by the barycentric interpolation of the vertices of the face where the point is on. With the 3D position, we can project it to the image space of a calibrated camera. Thus, we can sample image features and convert them into UV space. Fig. 1 shows an example of converting albedo to UV space. It is further converted into a partial texture by masking with visibility.

2 Visibility Map

To calculate visibility, as shown in Fig. 2, we rasterize a image with UV coordinates, then sample that to UV space, and compare with the correct UV

* Work was done during TZ internship at Facebook Reality Labs, Sausalito, CA, USA.

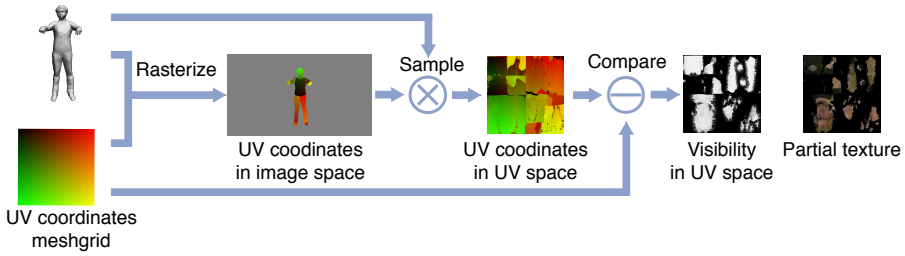


Fig. 2. Visibility and partial texture generation. By rasterizing UV coordinates to image space and sample it back to UV space, we obtain a UV coordinates map where only the visible parts are "correct". By comparing it with the ground truth UV meshgrid, we obtain the visibility map in UV space. We further calculate partial texture by masking the sampled albedo.

coordinates. The pixels whose sampled UVs are consistent with its position in UV space are visible. By masking the sampled albedo with visibility, we obtain the partial texture.

3 Augment MeshRef Features

In addition to VGG [8] features, we can further augment the features by including vertex position information. Specifically, we can rasterize coarse vertex position into both image spcae and UV space, to become a vertex position image I_{vp} and a vertex position map T_{vp}. We append I_{vp} to the input of VGGNet, and append T_{vp} to the input of the UV space CNN.

4 Implementation Details

Network Architecture. All the CNNs are U-Net sharing similar architectures, except for the VGG16 Network [8] in MeshRef module. See Tab. 1 for the shared components and Tab. 2, 3, and 4 for architectures of AlbeNorm, TexGen, and MeshRef CNNs. Specially, the CNN in TexGen predicts the residual between the coarse texture and the fine texture.

Hyperparameters. We use $K = 30$ for the number of selected frames, and $\lambda_a^{an} = 1, \lambda_n^{an} = 1, \lambda_{L1}^{tg} = 20, \lambda_{pct}^{tg} = 1, \lambda_{lap}^{tg} = 10, \lambda_{L1}^{mr1} = 1, \lambda_{ssim}^{mr1} = 1, \lambda_{lap}^{mr1} = 20, \lambda_{pct}^{mr2} = 1, \lambda_{sil}^{mr2} = 100, \lambda_{temp}^{mr2} = 10, \lambda_{pos}^{mr2} = 10, \lambda_{lap}^{mr2} = 10, \lambda_{deform}^{mr2} = 10$ for the loss weights. We use learning rate 10^{-5} for pretraining AlbeNorm, 10^{-4} for pretraining MeshRef, 3×10^{-4} for optimizing TexGen, and 5×10^{-5} for finetuning MeshRef. We use batch size 4 for pretraining AlbeNorm, 1 for pretraining MeshRef, 1 for optimizing TexGen, and 3 for finetuning MeshRef (as a triplet for motion smoothness loss). VGGNet is trained from scratch with MeshRef CNN,

Table 1. Network Components. We use ReLU [6] for activation, and Instance Normalization [9] for normalization

Type	Components
inconv	[Conv3×3 + ReLU + InstanceNorm]×2
down	[Conv3×3 + ReLU + InstanceNorm]×2 + MaxPool2×2
up	Upsample + [Conv3×3 + ReLU + InstanceNorm]×2
outconv	Conv1×1

Table 2. Network Architecture of AlbeNorm CNN

Name	Type	Input	Output Channels
inc	inconv	RGB+SH lighting	64
down1	down	inc	128
down2	down	down1	256
down3	down	down2	512
down4	down	down3	512
up1a	up	down4, down3	256
up2a	up	up1a, down2	128
up3a	up	up2a, down1	64
up4a	up	up3a, inc	64
outca (Normal Output)	up	up4a	3
up1b	up	down4, down3	256
up2b	up	up1b, down2	128
up3b	up	up2b, down1	64
up4b	up	up3b, inc	64
outcb (Albedo Output)	outconv	up4b	3

Table 3. Network Architecture of TexGen CNN

Name	Type	Input	Output Channels
inc	inconv	Coarse Texture	64
down1	down	inc	128
down2	down	down1	256
down3	down	down2	512
down4	down	down3	512
up1	up	down4, down3	256
up2	up	up1, down2	128
up3	up	up2, down1	64
up4	up	up3, inc	64
outc	outconv	up4	3

Table 4. Network Architecture of MeshRef CNN. "feat0", "feat1", "feat2", "feat3", "feat4" are features converted from VGGNet input, conv1_2, conv2_2, conv3_3, and conv4_3 features

Name	Type	Input	Output Channels
inc	inconv	feat0	64
down1	down	inc, feat1	128
down2	down	down1, feat2	256
down3	down	down2, feat3	512
down4	down	down3, feat4	512
up1	up	down4, down3	256
up2	up	up1, down2	128
up3	up	up2, down1	64
up4	up	up3, inc	64
outc	outconv	up4	3

and kept fixed during finetuning. To speed up finetuning, we use a smaller image size 480×270 for photometric losses, but the image features are from the 960×540 original image.

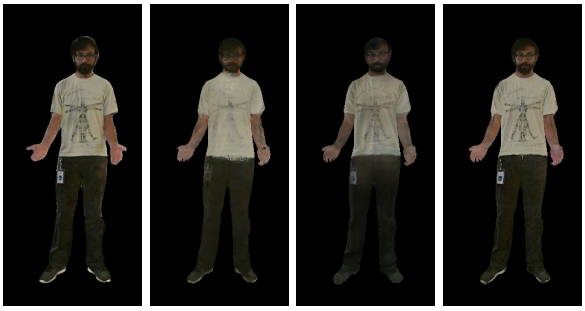
Adaptive Robust Perceptual Loss. We use the adaptive robust loss [2] for perceptual losses [4]. We use VGG16 conv1_2, conv2_2, conv3_3, and conv4_3 features for TexGen, and conv3_3, and conv4_3 features for MeshRef. We use learning rate 3×10^{-4} for the adaptive robust function.

5 Qualitative Texture Generation Results

See Fig. 3 and 4 for qualitative results. Our method provides clearer texture, better contrast, and less baked-in shading.

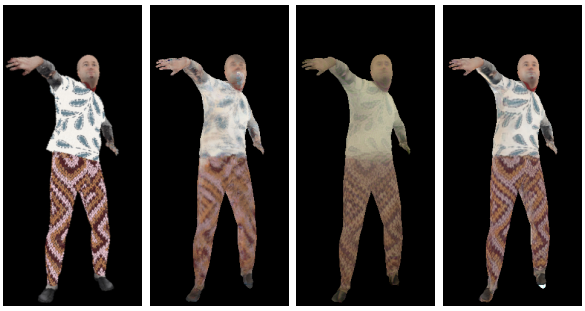
6 Handle Different Human Models and Camera Parameters

We establish a mapping between SMPL [5] and Our Human Model, so that we can replace the initial meshes of DeepHuman [10] and HMD [11] by our coarse mesh. Besides, DeepHuman and HMD use different camera parameters from ours, and the cropping makes it hard to simply transform the mesh from one setting to the other. Thus, we adopt an approximate way to do this: We transform our coarse mesh to match the position and scale of their original initial mesh. Similarly, after obtaining their reconstruction result, we transform the mesh to match the position and scale of our coarse mesh, for DeepHuman, HMD, and their variants.



(a) Albedo (b) SBM [1] (c) TNA [7] (d) Ours

Fig. 3. Comparing texture generation by visualizing no-shading image on real data. (a) is albedo image from AlbeNorm, which can be seen as "ground truth". Our result has clearer texture, and less baked-in shading.



(a) Albedo (b) SBM [1] (c) TNA [7] (d) Ours

Fig. 4. Comparing texture generation by visualizing no-shading image on synthetic data. (a) is ground truth albedo image. Our result has better contrast and clearer texture.

References

1. Alldieck, T., Magnor, M., Xu, W., Theobalt, C., Pons-Moll, G.: Video based reconstruction of 3d people models. In: CVPR (2018)
2. Barron, J.T.: A general and adaptive robust loss function. In: CVPR (2019)
3. Blinn, J.F., Newell, M.E.: Texture and reflection in computer generated images. Communications of the ACM **19**(10), 542–547 (1976)
4. Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: ECCV (2016)
5. Loper, M., Mahmood, N., Romero, J., Pons-Moll, G., Black, M.J.: Smpl: A skinned multi-person linear model. TOG **34**(6), 1–16 (2015)
6. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: ICML (2010)
7. Shysheya, A., Zakharov, E., Aliev, K.A., Bashirov, R., Burkov, E., Iskakov, K., Ivakhnenko, A., Malkov, Y., Pasechnik, I., Ulyanov, D., et al.: Textured neural avatars. In: CVPR (2019)
8. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR (2015)
9. Ulyanov, D., Vedaldi, A., Lempitsky, V.: Instance normalization: The missing ingredient for fast stylization. arXiv preprint arXiv:1607.08022 (2016)
10. Zheng, Z., Yu, T., Wei, Y., Dai, Q., Liu, Y.: Deephuman: 3d human reconstruction from a single image. In: ICCV (2019)
11. Zhu, H., Zuo, X., Wang, S., Cao, X., Yang, R.: Detailed human shape estimation from a single image by hierarchical mesh deformation. In: CVPR (2019)