DoubleField: Bridging the Neural Surface and Radiance Fields for High-fidelity Human Reconstruction and Rendering Supplementary Material

1. More Implementation Details

1.1. Surface-guided Sampling

The surface-guided sampling strategy will determine the intersection points in the surface field at first and then perform fine-grained sampling around the intersected surface. Specifically, given camera parameters of the rendering view and the ray $\mathbf{r} = \mathbf{o} + t\mathbf{d}$, a uniform sampling is firstly applied along the ray in the depth bounds $[t_n, t_f]$:

$$t_i \sim \mathcal{U}\left[t_n + \frac{(i-1)(t_f - t_n)}{N_s}, t_n + \frac{i(t_f - t_n)}{N_s}\right], \quad (1)$$

where N_s is the number of sampling points, and each point is formulated as $\boldsymbol{x}_i = \mathbf{o} + t_i\mathbf{d}$. We query the surface field value of each point and determine the first intersection position t_c on the surface:

$$t_c = \min\left\{t_i | o(\boldsymbol{x}_i) > 0.5\right\}. \quad (2)$$

The intersection positions are then used to guide the sampling at a more fine-grained level by considering the radiance field surrounding the intersected surface:

$$t_j \sim \mathcal{U}\left[t_c - \frac{N_r - 2(j-1)}{N_r}\delta, t_c + \frac{2j - N_r}{N_r}\delta\right], \quad (3)$$

where δ is a pre-defined sampling radius and N_r is the number of sampling points at the fine-grained level. The final color is rendered by the integration as NeRF [4]:

$$\hat{C}(\boldsymbol{r}) = \sum_{j=1}^{N_r} T(j)(1 - \exp(-\sigma(\boldsymbol{x}_j)\delta_j))c(\boldsymbol{x}_j), \quad (4)$$

In practice, we use $N_s = 64$ sampling points used to determine the surface intersection. After determining the first intersection, there are $N_r = 16$ sampling points around the surface within the radius of $\delta = 0.03m$ for the query of the neural radiance field.

1.2. Positional Encoding

Following the transformer [10] and NeRF [4], we apply positional encoding to the direction of the viewing ray, the coordinate of the sampling point and the raw RGB values of the input image. Given a vector $\boldsymbol{v}$, position encoding maps $\boldsymbol{v}$ to a higher dimension space:

$$\begin{aligned}\gamma^*(\boldsymbol{v}) = (&\sin(2^0\pi\boldsymbol{v}), \cos(2^0\pi\boldsymbol{v}), ..., \\ &\sin(2^{L-1}\pi\boldsymbol{v}), \cos(2^{L-1}\pi\boldsymbol{v})),\end{aligned} \quad (5)$$

In practice, we set L to 10 and additionally concatenate the result with the origin vector $\boldsymbol{v}$:

$$\begin{aligned}\gamma(\boldsymbol{v}) = (&\sin(2^0\pi\boldsymbol{v}), \cos(2^0\pi\boldsymbol{v}), ..., \\ &\sin(2^{L-1}\pi\boldsymbol{v}), \cos(2^{L-1}\pi\boldsymbol{v}), \boldsymbol{v}).\end{aligned} \quad (6)$$

1.3. Network Architecture

Our framework contains three main network modules: 1) the image encoder, 2) the view-to-view transformer and 3) the DoubleField network. Here, we provide more details about these three modules.

Image Encoder Following PIFu [8], we design our image encoder based on the Hourglass architecture [5], which has 2 stacked layers and each layer has 4 stacks. The image encoder takes the images with size of $512 \times 512 \times 3$ as inputs and produces the feature maps with the size of $128 \times 128 \times 256$.

View-to-view Transformer We design the view-to-view transformer based on the classical encoder-decoder structure [10]. The number of tokens is equal to the number of views in our transformer. Since only sparse multi-view inputs are involved and the sampling points can be divided into batches during training, our transformer does not suffer from memory issues. 1) The **encoder** has 1 layer with 8 heads. The size of its inner embedding is 256 and the dimension of Q, K, V for each head is 32. The encoder takes the concatenation of multi-view image features and the direction embedding as inputs and produces the fused features with the same dimension of $256 + (60 + 3)$. The fused features are concatenated with the positional encoding of query points and then fed into the double MLP to obtain

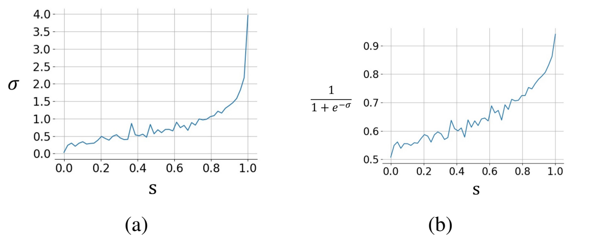


Figure 1. Relationship between the values of the occupancy s in the surface field and the density σ in the radiance field.

the double embedding with the dimension of $256 + (60 + 3)$. 2) The **decoder** consists of 1 layer with 8 heads. The size of its inner embedding is also 256 and the dimension of Q, K, V is 64. The decoder takes the concatenation of the double embedding $\boldsymbol{e}_{db}$, the colored embedding, and the direction embedding of the query view as inputs. To keep the shape of different inputs be the same one for the attention operation, we generate a zero vector with the size of $256 + (60 + 3) + (60 + 3)$ for the query view and assign the its direction embedding to the vector. Finally, the generated features and the double embedding concatenated with the colored embedding are fed into decoder to obtain the texture embedding $\boldsymbol{e}_c$.

DoubleField Network There are 3 MLPs in our DoubleField Network for the prediction: 1) The **double MLP** E_{db} has 8 layers and the numbers of neurons in each layer are $[340, 256, 256, 512, 512, 256]$. 2) The **geometry MLP** E_g consists of 3 layers and the numbers of neurons in each layer are $[319, 128, 2]$. The two channels of the outputs are the occupancy value of the surface field and the density value of the radiance field, respectively. For the prediction of the surface field, a sigmoid activation function is applied to constrain the value within $[0, 1]$. 3) The **texture MLP** E_c contains 4 layers and the numbers of neurons in each layer is $[382, 256, 128, 3]$, which is used to output the color values of the radiance field.

1.4. Dataset Preparation

There are total 1,700 models collected from the Twindom dataset, where 1,500 models are used for training and 200 models are used for testing. For each model, we first normalize its height to 1.8m, then adopt Taichi [1] and its derivative Taichi_three to render images with the size of 512×512 from 360 degrees in yaw axis. To improve the generalization in real-world scenes, we use a perspective camera model and add random rotation of $[-30, 30]$ degrees. The distance between the human model and camera is randomly set within the interval of $[1, 2]$.

For the evaluation on ultra-high resolutions, we render images with the ultra-high resolution of 4096×4096 from 360 degrees in yaw axis. Here, we still adopt the perspective camera model but do not add random rotation.

1.5. Training Phase

The learning of geometry and appearance are simultaneous in the pre-training phase. We adopt the spatial sampling strategy in PIFu [8] for the learning of geometry and the proposed surface-guided sampling strategy for the learning of appearance. For the spatial sampling, the number N_o of the sampling points is set to 20,000 and the standard deviation σ of the random offsets is set to 0.03m. We use 4 random views among 360 degrees as input and 1 random view among 360 degrees as supervision for multi-view training. The learning rate in our training process is 1e-5 and the batch size is 3. We use 3 NVIDIA GeForce RTX 2080 GPUs to train the network for about 4 days.

Loss function As mentioned in the main paper, our loss function can be formulated as:

$$L = \lambda_g L_g + \lambda_r L_r + \lambda_c L_c, \tag{7}$$

where we set $\lambda_g = 1$, $\lambda_r = 0.5$, $\lambda_c = 0.5$ in the training phase.

1.6. Finetuning Phase

During finetuning, we only adopt the surface-guided sampling strategy to refine the geometry and appearance results. For 6 view inputs, we randomly pick 4 of 6 views as input and 1 of 6 views for supervision. The reason that we do not use all views is to avoid overfitting and improve generalization. The setting of the sampling strategy is the same with the pre-training phase. The learning rate of finetuning is set to 1e-6. We first fix the color module and refine the geometry for 2,000 iterations, which takes about 10 minutes. Then we fix the geometry module and refine the appearance results by finetuning the texture MLP and the view-to-view transformer for additional 2,000 iterations, which takes about 10 minutes. The finetuning experiments are conducted on a single NVIDIA GeForce RTX 2080 GPU.

1.6.1 Other methods

PixelNeRF [11] For the finetuning on 6 views input, we randomly pick 4 of 6 views as input and 1 of 6 views for supervision. The input image of PixelNeRF is downsampled to 512×512 but the output is supervised by images with the resolution of 4096×4096. We finetune all modules of PixelNeRF with additional 4000 iterations.

PIFu [8]+DVR [6] Similar to PixelNeRF, the input images of PIFu+DVR are downsampled to 512×512 but its outputs are supervised by images with the resolution of 4096×4096.

Method	Training or finetuning			Inference		
	query points (per image)	maximum ray nums	time	marching cube	query points (per image)	time
NeRF sampling [11]	5×10^7	3×10^2	5.0s	0	5×10^7	5.0s
Surface-guided Sampling	2×10^6(n.g.)+5×10^5	2×10^3	0.4s	7×10^4	5×10^5	0.2s
Surface-guided Sampling (with precomputed depth)	5×10^5	3×10^3	0.2s	0	5×10^5	0.15s

Table 1. Comparison of the NeRF sampling and our surface-guided sampling strategies in the training, finetuning, and inference processes, where n.g. means no gradient.

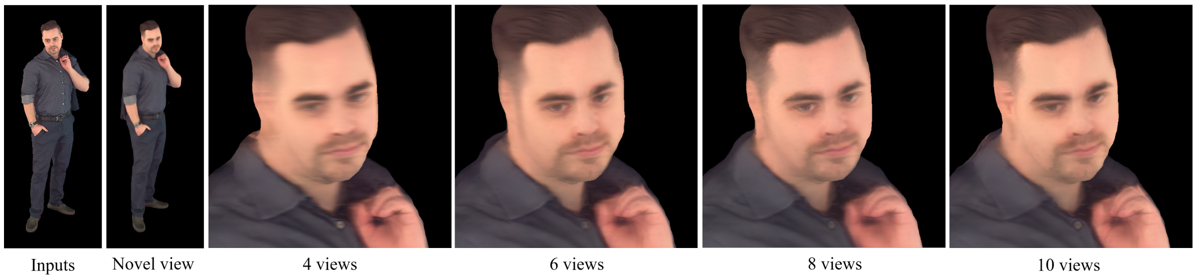


Figure 2. Novel-view rendering results given different numbers of input views.

	Iterations				
	0	500	1k	2k	4k
PSNR	23.04	23.25	23.36	23.46	23.66
Chamfer	0.767	0.744	0.734	0.738	0.732

Table 2. The quantitative results across different finetuning iterations.

We finetune all modules of the network with additional 4000 iterations.

NeuralBody [7] We use the released code of NeuralBody and regard each model is trained on a single frame. The training process is the same with the original NeuralBody and we modify the dataset part to enable it to take ultra-high-resolution images with size of 4096×4096 as input. The training settings such as the learning rate and decay weight are the same with the original implementation. We train each model for about 15 hours on a single NVIDIA GeForce RTX 2080 GPU. In video comparisons, we train NeuralBody on the whole video sequence for about 20 hours on a single NVIDIA GeForce RTX 2080 GPU.

1.7. Evaluation Metrics

Geometry Evaluation For the geometry evaluation, we adopt the Chamfer distance and point-to-surface distance as the evaluation metrics, which are implemented based on the released code of PIFuHD [9]. The number of sampling points for evaluation is 20,000.

Appearance Evaluation For the appearance evaluation, we adopt PSNR and SSIM as the evaluation metrics, which have the same protocol with NeuralBody [7]. For fair comparisons with NeuralBody, we compute PSNR and SSIM on the effective areas of rendering images and ground truth images, which are cropped using the BoundingRect in OpenCV.

2. Additional Ablation Study

2.1. Calibration errors

Connecting the two fields in the feature space is also morerobust to small calibration errors since the density field needn't strictly align with the surface field after finetuning. We give an additional experiment by altering the calibration matrix in synthetic data (randomly add 3cm translation and 3° rotation) and then finetuning our network. As shown in Fig. 3, some artifacts (in eye regions) disappear after finetuning and the rendering results are more photo-realistic.

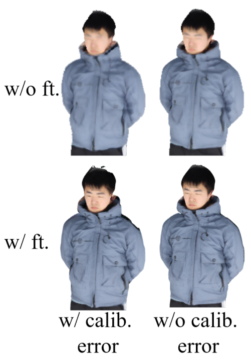


Figure 3. Ablation study for calibration errors.

2.2. Efficiency of Surface-guided Sampling

In Table 1, we compare the numbers of query points and the maximum numbers of ray per batch in different sampling strategies to validate the efficiency of the proposed surface-guided sampling strategy during the training, finetuning, and inference processes. The experiments are conducted on a

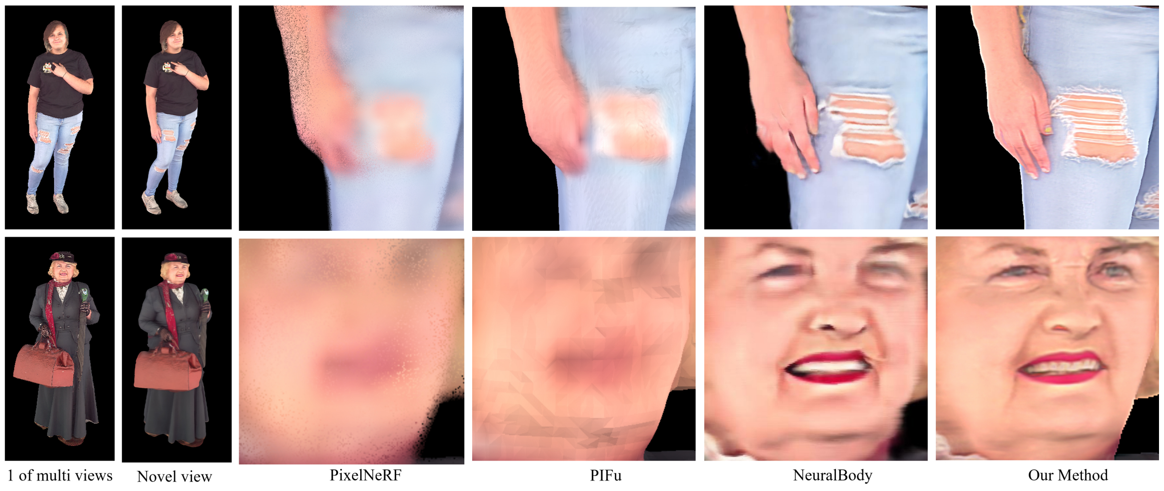


Figure 4. Additional results on the Twindom dataset. Best viewed digitally with zoom-in.

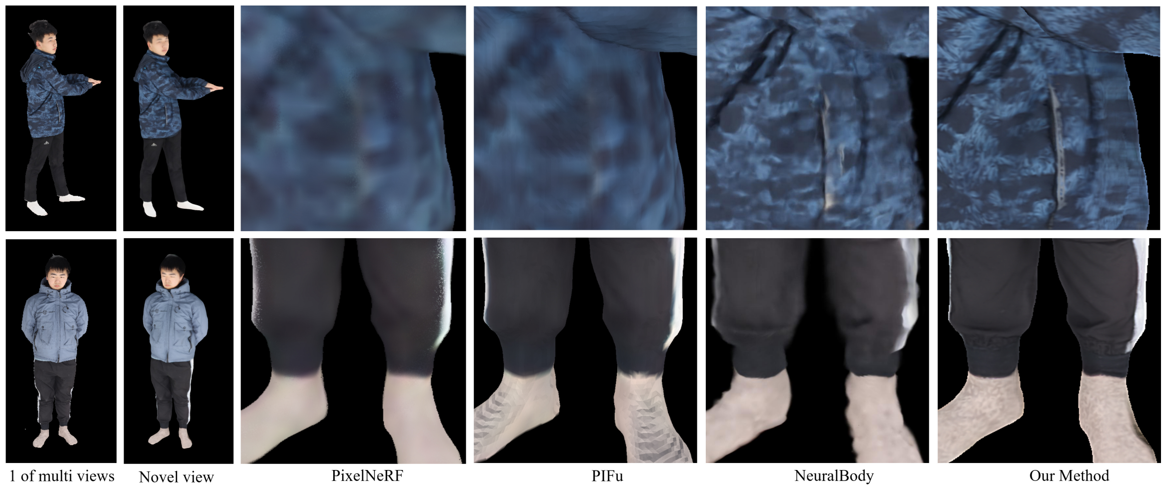


Figure 5. Additional results on the THuman2.0 dataset. Best viewed digitally with zoom-in.

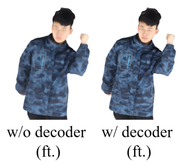
single NVIDIA GeForce RTX 2080 GPU, where 10 models are randomly picked from a dataset to evaluate the average number of query points.

As shown in Table 1, DoubleField with the surface-guided sampling strategy is faster and more efficient on account of using much less query points for rendering and supporting more sampling rays per batch. Moreover, when the depth map is precomputed by Marching Cube, our method can be accelerated a lot and save more memory.

2.3. The decoder of view-to-view transformer

As discussed in L.466-476 of the main paper, our view-to-view transformer has an encoder-decoder structure and differs from DeepMultiCap [12] that only uses an encoder structure. The decoder directly utilizes raw RGB values from ultra-high resolution images, which is essential to achieve high-fidelity human rendering. We also provide additional ablation study for the decoder on Thuman2.0 dataset as follows:

Method	PSNR	SSIM
w/o decoder	23.95	0.871
w/o decoder(Ft.)	24.26	0.880
our method(Ft.)	**25.10**	**0.905**



In this experiment, we remove the decoder and add color encoding to Eq.5, then feed the fused features into the color MLP and concatenate them with the view direction. As shown above, both quantitative and qualitative results validate the necessity of the decoder. Without the decoder, the results are blurry even after finetuning.

2.4. Number of Input Views

Our network is trained using 4 views as inputs and can be generalized to more views. We evaluate our network using different numbers of views and show the novel-view rendering results in Fig. 2. Given more views as inputs, our method can achieve higher quality rendering. As shown in Fig. 2, our method can obtain plausible results with the inputs from only 6 sparse views.

2.5. Finetuning Iterations and Orders

We further evaluate our method with different finetuning iterations. The results are reported in Tab. 2. Our method can refine geometry and achieve high-fidelity rendering by fast finetuning, which takes about 2,000 iterations until the convergence of geometry finetuning and about additional 2,000 iterations until the convergence of appearance finetuning. In addition, the strategy with the first finetuning color followed by geometry (both with 2K iterations) is unstable and leads to higher Chamfer distance 0.772 and lower PSNR 23.35.

2.6. Relationship between the Two Fields

We randomly pick 10 models from the Twindom testing dataset and randomly sample 20,000 points around the surface to query the occupancy value of the surface field and the density of the radiance field. The relationship between the values of the occupancy s and the density σ is depicted in Fig. 1, which indicates the DoubleField representation associates the two fields through the learning process and builds an implicit exponential relationship between the two fields.

3. Additional Results

3.1. Results on Twindom and THuman2.0 Datasets

We show additional results on the Twindom and THuman2.0 datasets in Fig. 4 and Fig. 5, respectively. All figures are best viewed digitally with zoom-in.

3.2. Results on Real-world Data

We adopt 5 Kinect Azure cameras to capture ultra-high-resolution RGB images with size of 4096×3072 and the frame rates is 15 FPS. We shut down the depth sensor to avoid dropping frames but there are still some missing frames. After obtaining multi-view images, we adopt SCHP [3] and MODNet [2] to segment people in each view. Then we finetune our network on the whole sequence for 4000 iterations (2000 iterations for geometry and 2000 iterations for color). please refer to the supplementary video to see additional results on real-world data.

References

[1] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *TOG*, 38(6):201, 2019.

[2] Zhanghan Ke, Kaican Li, Yurou Zhou, Qiuhua Wu, Xiangyu Mao, Qiong Yan, and Rynson WH Lau. Is a green screen really necessary for real-time portrait matting? *arXiv preprint arXiv:2011.11961*, 2020.

[3] Peike Li, Yunqiu Xu, Yunchao Wei, and Yi Yang. Self-correction for human parsing. *arXiv preprint arXiv:1910.09777*, 2019.

[4] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, pages 405–421. Springer, 2020.

[5] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *ECCV*, pages 483–499. Springer, 2016.

[6] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3D representations without 3D supervision. In *CVPR*, pages 3504–3515, 2020.

[7] Sida Peng, Yuanqing Zhang, Yinghao Xu, Qianqian Wang, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Neural body: Implicit neural representations with structured latent codes for novel view synthesis of dynamic humans. In *CVPR*, 2021.

[8] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *ICCV*, October 2019.

[9] Shunsuke Saito, Tomas Simon, Jason Saragih, and Hanbyul Joo. Pifuhd: Multi-level pixel-aligned implicit function for high-resolution 3D human digitization. In *CVPR*, 2020.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[11] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *CVPR*, 2021.

[12] Yang Zheng, Ruizhi Shao, Yuxiang Zhang, Tao Yu, Zerong Zheng, Qionghai Dai, and Yebin Liu. Deepmulticap: Performance capture of multiple characters using sparse multiview cameras. In *ICCV*, 2021.