# SAILOR: Synergizing Radiance and Occupancy Fields for Live Human Performance Capture

ZHENG DONG, State Key Laboratory of CAD&CG, Zhejiang University, China
KE XU, City University of Hong Kong, China
YAOAN GAO, State Key Laboratory of CAD&CG, Zhejiang University, China
QILIN SUN, The Chinese University of Hong Kong, Shenzhen and Point Spread Technology, China
HUJUN BAO, State Key Laboratory of CAD&CG, Zhejiang University, China
WEIWEI XU*, State Key Laboratory of CAD&CG, Zhejiang University, China
RYNSON W.H. LAU, City University of Hong Kong, China

## *Supplementary Material*

This supplementary material provides more information on the method details (Section 1), more results (Section 2), and visualization of our interactive GUI (Section 3).

## 1 MORE METHOD DETAILS

### 1.1 Loss Functions for $\mathcal{F}_d$

***(1)*** *Depth consistency loss:* it penalizes the per-pixel difference between the rendered ground-truth depth map (denoted as $\mathbf{D}_{gt}$) and the denoised depths $\mathbf{D}_{rf}$, which can be written as:

$$L_D = \sum_{i=1}^{N} \mathcal{L}_2(\mathbf{d}^i_{rf}(p), \mathbf{d}^i_{gt}(p)). \tag{1}$$

***(2)*** *Normal consistency loss:* it penalizes the error of the computed normal maps to enhance the details of $\mathbf{D}_{rf}$, which is:

$$L_N = \sum_{i=1}^{N} \mathcal{L}_1(\mathbf{n}^i_{rf}(p) - \mathbf{n}^i_{gt}(p)) + \mathcal{L}_1(1, < \mathbf{n}^i_{rf}(p), \mathbf{n}^i_{gt}(p) >), \tag{2}$$

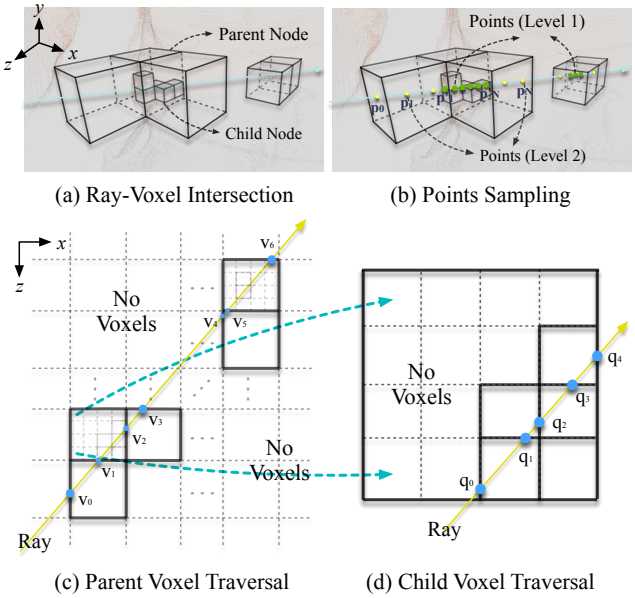
where $\mathbf{d}^i_{rf}(p)$ and $\mathbf{n}^i_{rf}(p)$ denote the predicted depth value of $\mathbf{D}_{rf}$ and normal vector of $\mathbf{N}_{rf}$, respectively, in pixel $p$ of view $i$. $\mathbf{N}_{rf}$ is the normal map computed from $\mathbf{D}_{rf}$. $\mathbf{d}^i_{gt}(p)$ and $\mathbf{n}^i_{gt}(p)$ are the corresponding ground-truth depth value and normal vector. $\mathcal{L}_1$ and $\mathcal{L}_2$ represent the smooth $L_1$ loss and $L_2$ loss, respectively.

***(3)*** *3D consistency loss:* it further constrains the consistency between the point cloud fused from $\mathbf{D}^i_{rf}$ and the ground-truth point cloud, to eliminate the artifacts in the subsequent depth fusion, as:

$$L_P = \mathcal{L}_c(\mathbf{P}, \mathbf{P}_{gt}), \tag{3}$$

where $\mathbf{P} = F(\{\mathbf{D}^i_{rf}, \mathbf{K}^i, \mathbf{RT}^i\}_{i=1,...,N})$ is the fused point cloud from $\mathbf{D}^i_{rf}$, and $F$ represents the depth fusion scheme. $\mathbf{P}_{gt}$ is the sampled point cloud from the ground-truth 3D model. $\mathcal{L}_c$ denotes the chamfer loss.

The complete loss function for depth denoising can be written as: $L = L_D + \lambda_N L_N + \lambda_P L_P$, where $\lambda_N$ and $\lambda_P$ are the balancing hyper-parameters.




(a) Ray-Voxel Intersection (b) Points Sampling

(c) Parent Voxel Traversal (d) Child Voxel Traversal

Fig. 1. The ray-voxel intersection and points sampling processes (a,b). We use the Voxel Traversal Algorithm [Amanatides et al. 1987] to determine the voxels, near and far points ($v_i$) that are intersected with the ray (c). We determine the intersected points ($q_i$) of the child voxels with the ray (d). For adaptive point sampling in rendering, the proportion of sampling points varies at two different scales of voxels (b).

### 1.2 Ray-Voxel Intersection and Points Sampling

After the construction of two-layer $\mathcal{T}$, to render a novel-view image, we project rays from pixels of the target view $t$, and perform *Ray-Voxel Intersection* to identify voxels (in both levels of $\mathcal{T}$) that are intersected with the rays, and perform *Points Sampling* to sample the points inside the intersected voxels on the rays.

*Ray-Voxel Intersection.* As shown in Fig. 1(a), given an emitted ray $\mathbf{l}$, we detect the parent and child voxels intersected by $\mathbf{l}$, as follows: (1) Use the voxel traversal algorithm [Amanatides et al. 1987] to detect the valid parent voxels along the ray $\mathbf{l}$, according to the index volume $\mathbf{V}^2_{idx}$, and record the depth values of the ray-voxel intersected points (near and far) at the target view $t$. In the case shown in Fig. 1(c), there are a total of 4 valid parent voxels intersected with $\mathbf{l}$, and the recorded near and far depths are:

*Corresponding author

$\mathbf{D}_{near} = d(\{\mathbf{v}_i\}_{i=0,1,2,4,5}, \mathbf{RT}^t)$ and $\mathbf{D}_{far} = d(\{\mathbf{v}_i\}_{i=1,2,3,5,6}, \mathbf{RT}^t)$, where $d(\cdot, \mathbf{RT}^t)$ is a projection function to obtain the depth value of a 3D point. (2) Continue to use the voxel traversal algorithm to detect all valid child voxels and record their near and far depth lists, noted as $\mathbf{D}'_{near}$ and $\mathbf{D}'_{far}$, as shown in Fig. 1(d). (3) Merge all near and far depth values into lists $\mathbf{D}^{all}_{near} = [\mathbf{D}_{near}, \mathbf{D}'_{near}, ...]$ and $\mathbf{D}^{all}_{far} = [\mathbf{D}_{far}, \mathbf{D}'_{far}, ...]$, respectively, as the final ray-voxel intersection results, to determine the points sampling range.

*Points Sampling.* We sample points between all near and far points (*e.g.*, $\{\mathbf{v}_0, \mathbf{v}_1\}$, $\{\mathbf{q}_0, \mathbf{q}_1\}$) along the ray $\mathbf{l}$ for rendering. To allocate the number of sampling points for each valid voxel, we first compute the assigned weight for the $i$-th voxel as:

$$w_i = \frac{(\mathbf{d}_{far}(i) - \mathbf{d}_{near}(i)) \cdot \mathbf{s}_i}{\sum_k^{N_v} (\mathbf{d}_{far}(k) - \mathbf{d}_{near}(k)) \cdot \mathbf{s}_k}, \tag{4}$$

where $N_v = len(\mathbf{D}^{all}_{near})$ is the number of valid voxels. $\mathbf{d}_{far}(i)$ and $\mathbf{d}_{near}(i)$ represent the far and near depths of $\mathbf{D}^{all}_{far}$ and $\mathbf{D}^{all}_{near}$ at voxel $i$. $\mathbf{s}_i$ is the size scale of voxel $i$. In order to allocate more sampling points for the child voxels, we set the scale $\mathbf{s}_i$ to 4, so that more points near the surface will be sampled, as shown in Fig. 1(b). Given the total number $M$ of sampling points for ray $\mathbf{l}$, the number of sampled points in voxel $i$ is then computed as:

$$m_i = \max(\lfloor w_i \cdot M \rfloor, 1). \tag{5}$$

We allocate points along the ray direction for each voxel to ensure that voxels on the surfaces closest to the source of the ray are always sampled. If $\sum_i m_i < M$, we reassign the remaining points $M - \sum_i m_i$ according to the previous allocating order. Finally, we determine depth $d_j$ of the $j$-th sampling point within the voxel $i$ via:

$$d_j = \mathbf{d}_{near}(i) + \frac{\mathbf{d}_{far}(i) - \mathbf{d}_{near}(i)}{m_i + 1} \cdot (j + 1), \tag{6}$$

where $j$ starts from 0, and $m_i$ sampled points are evenly distributed in voxel $i$.

The whole process has been implemented with CUDA acceleration. Given $512^2$ rays, it takes around 10ms and 2ms for ray-voxel intersection and points sampling, respectively.

## 1.3 Loss Functions for $\mathcal{F}_b$

To train the blending network $\mathcal{F}_b$, we penalize the per-ray error between the final predicted color patch $\hat{\mathbf{P}}_\mathbf{r}$ and the ground-truth color patch $\mathbf{P}^*_\mathbf{r}$ using both L1 and SSIM [Wang et al. 2004] losses as:

$$L_B = \sum_{\mathbf{r} \in R} \mu_1 \cdot \mathcal{L}_1(\hat{\mathbf{P}}_\mathbf{r}, \mathbf{P}^*_\mathbf{r}) + \mu_2 \cdot (1 - \mathbf{SSIM}(\hat{\mathbf{P}}_\mathbf{r}, \mathbf{P}^*_\mathbf{r})), \tag{7}$$

where $\hat{\mathbf{P}}_\mathbf{r} = \{\hat{\mathbf{C}}_{\mathbf{r}(i,j)} | 1 < i, j < S_{patch}\} \in \mathbb{R}^{S_{patch} \cdot S_{patch} \cdot 3}$ is the color patch with size of $S_{patch} \cdot S_{patch}$ (set to 128). $(i, j)$ indicates the pixel index of the sub-rays inside the patch. $\mu_1$ and $\mu_2$ are the balancing terms. Compared to the ray-independent color loss function (Eq. 5 in our main paper), $L_B$ used for $\mathcal{F}_b$ can better learn the relationship between rays within a patch. Besides the loss for measuring the per-ray error, we also incorporate a feature loss (denoted as $L_{ft}$) based on a pre-trained VGG-16 network [Simonyan and Zisserman 2014], to further improve the rendered quality, as:

$$L_{ft} = \sum_{\mathbf{r} \in R} \mathcal{L}_{VGG}(\hat{\mathbf{P}}_\mathbf{r}, \mathbf{P}^*_\mathbf{r}), \tag{8}$$

where $\mathcal{L}_{VGG}$ denotes the L1 loss between VGG features, and we use the three features fed to the first three *MaxPool2d* layers to compute this loss. The overall training loss for $\mathcal{F}_b$ becomes: $L_B + \mu_{vgg} \cdot L_{ft}$, where the balancing term $\mu_{vgg}$ is set to 0.01 empirically in our experiments. We train $\mathcal{F}_b$ independently without updating the parameters of our SRONet.

## 1.4 Our Capturing System

The capturing system comprises 8 Kinect cameras, which are evenly placed around the performers in a circle at 45-degree intervals, and each camera is approximately 1.5m away from the actor. The lighting system contains 4 bottom spotlights and 4 ceiling lights, evenly placed around the performers for global illumination. Before capturing, the optical axis of each camera faces the actor, enabling actors to perform a series of actions at the center. The 8 Kinect cameras are pre-calibrated using the iterative closest point (ICP) method, and synchronized at 15 frame-per-seconds.

For novel-view rendering evaluation, we use RGBD images of 4 fixed perspective views (the interval between two adjacent views is 90 degrees, and the indexes of cameras are 0,4,6,7, respectively) as inputs. RGBD images of the other four views (*i.e.*, indexes of 1,2,3,5) are used to evaluate rendering quality.

According to the multi-camera-sync provided by Microsoft Kinect-dk, we can prevent the interference between lasers by setting the capture offset to above 160$\mu$s. For input depths, we simulate the depth noise based on z-Distance by choosing a function (*e.g.*, $1.5z^2 - 1.5z + 1.375$) according to the Kinect paper [Fankhauser et al. 2015]. We also add larger Gaussian noise (average scale is 1.5cm) and holes (width 3 pixels on average) to cover the possible depth noise cases.

## 1.5 Implementation Details

We have implemented SAILOR with PyTorch [Paszke et al. 2017] and CUDA acceleration. We train our model (*i.e.*, depth denoising network $\mathcal{F}_d$, SRONet, and neural blending module $\mathcal{F}_b$) on 2 NVIDIA RTX3090 GPUs using the ADAM optimizer [Kingma and Ba 2014]. The $\beta_1$, $\beta_2$ in ADAM optimizer are set to 0.5 and 0.99, respectively. We train $\mathcal{F}_d$ for 10 epochs with a batch size of 6 and a learning rate of $2e^{-4}$. We train the SRONet for 20 epochs with a batch size of 4 and a learning rate starting from $1e^{-4}$ and decaying by half every 5 epochs. We train the $\mathcal{F}_b$ for 10 epochs with a batch size of 2 and a learning rate starting from $1e^{-4}$ and decaying by half every 5 epochs. We set the weight balancing parameters, *i.e.*, ($\lambda_N$ and $\lambda_P$) in $\mathcal{F}_d$ to (0.5 and 0.01); ($\mu_o$, $\mu_c$, and $\lambda_{D'}$) in SRONet to (0.5, 1.0, and 1.0); ($\mu_1$, $\mu_2$, and $\mu_{vgg}$) in $\mathcal{F}_b$ to (0.4, 0.6 and 0.01), respectively. We set the hyper-parameters, $\alpha$ and $\gamma$ in two-layer tree $\mathcal{T}$ to 40 and 0.01, respectively. The number of sampling points $M$ is set to 48 for training and evaluation. The truncated PSDF value $\delta_p$ is set to 0.01. $\sigma_v$ is set to 200. *All the rendering results are of* $1K(1024 \times 1024)$ *resolution.*

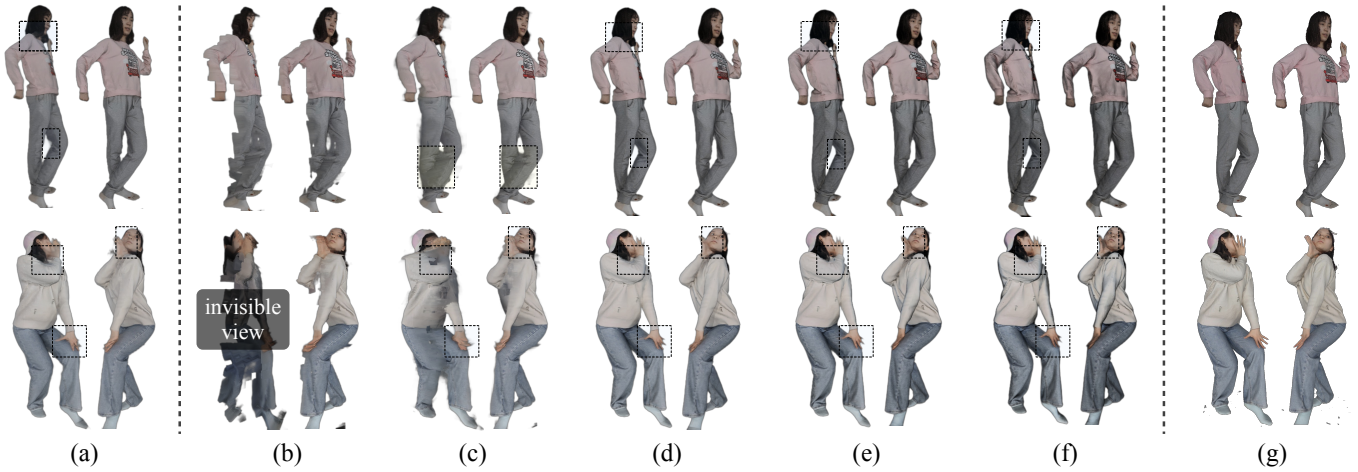



Fig. 2. Comparisons on the number of views during inference. 4 views (our setting) (a). Single view (b). 2 views (c). 6 views (d). 8 views (e). 10 views (f). GT (g).

### 1.6 Training and Inference Details

The training of SAILOR can be separated into three stages, as follows:

(1) We first train our depth denoising network $\mathcal{F}_d$. For the images data, we sample the image tuples $\{\mathbf{I}^i, \mathbf{D}^i, \mathbf{D}^i_{gt}\}_{i=1,...,\mathcal{N}}$ at 90-degree intervals from our training set (this sampling manner is related to our capture setting, where $\mathcal{N} = 4$ and the RGBDs are downsampled to a resolution of $512^2$). For 3D consistency loss, we randomly sample the point cloud $\mathbf{P}_{gt}$ (approx. $\sim 2 \times 10^5$ points) of the performer. $\mathcal{F}_d$ is trained for 10 epochs with a batch size of 6.

(2) We train our SRONet after building the two-layer tree for the target performer. Given the sampled image tuples, we randomly select two novel views $\{\mathbf{I}^{n_0}, \mathbf{I}^{n_1}\}$ from the remaining $60-\mathcal{N}$ views, and sample 1680 rays in each novel view $n_i (i = 1, 2)$ for each mini-batch, where 60 is the number of the images rendered for each 3D scan in our training set. We obtain the ground-truth color vector $\mathbf{C}^*(\mathbf{l})$ and the depth value $\mathbf{D}^*(\mathbf{l})$ for each emitted ray $\mathbf{l}$ to train our SRONet. Besides, we sample 8000 3D points around the performer according to the sampling strategy of PIFu [Saito et al. 2019] and obtain the corresponding GT occupancy value $\mathbf{o}*$ for synergically training our OccNet. The SRONet is trained for 20 epochs with a batch size of 4, while the learning rate is reduced by a factor of 2 after every 5 epochs.

(3) At last, we train our neural blending module $\mathcal{F}_b$, where the parameters of SRONet are fixed during training. We sample $128^2$ rays to form a ground-truth color patch $\mathbf{P}^*_\mathbf{r}$ from a novel view $n_i$ to train $\mathcal{F}_b$. $\mathcal{F}_b$ is trained for 10 epochs with a batch size of 2, while the learning rate is reduced by a factor of 2 after every 5 epochs.

For inference, we use 4 RGBD images as input. The interval between two adjacent views is set to 90 degrees, and the distance between each camera and the performer is approximately 1.5m. When evaluating our method on the THuman2.0 test set, we generate noise of 5 different degrees (*i.e.*, 0.25cm, 0.5cm, 1.0cm, 1.5cm of Gaussian standard deviation) on the ground-truth depth of the 3D meshes for both rendering and reconstruction. In our real-caputred data, the performer stands at the center of the circle that is enclosed by four cameras. We use RGBD images of 4 fixed perspective views as input to SAILOR, of which the indexes of cameras are 0,4,6,7, respectively. RGBD images of the other four views (i.e., indexes of 1,2,3,5) are used to evaluate rendering quality. We crop and down-sample the captured RGBD images to a resolution of $1k^2$, as the inputs for our method. For the depth denoising, two-layer tree building, and SRONet, the input RGBD images continue to be downsampled to a resolution of $512^2$ for acceleration.

### 1.7 Network Structures

(1) Our depth denoising network $\mathcal{F}_d$ is Unet based, similar to the network structure in GTPIFu [Dong et al. 2022]. As shown in Fig. 3, our $\mathcal{F}_d$ adopts two individual *HRNetV2-W18-Small-v2* networks to process the RGB and depth data, separately. Here, we only use the first three stages of HRNet, which leads to a lighter backbone network design. In addition, we remove the CAM and GAM modules that are proposed in GTPIFu for processing the geometric features. Only the ASPP [Chen et al. 2017] and ResCBAM [Woo et al. 2018] modules are retained to fuse the RGB and depth features, and the fused features are sent to the previous stages of the backbone to form a UNet-like structure. At last, the output map is masked by the full-body binary mask to only preserve the body part. For the RGBDs inputs of our $\mathcal{F}_d$, we normalize the RGB data to $[-1, 1]$ via $\mathbf{2} * \mathbf{I}^i - \mathbf{1}$, and the depth data to $[-1, 1]$ using the maximum and minimum values (*i.e.*, $d_{max}$, $d_{min}$) of the effective depth values. We denote the normalized depth map as $\mathbf{D}^i_n$ for view $i$, and this process can be formulated as:

$$\mathbf{d}^i_n(\mathbf{p}) = \begin{cases} \frac{\mathbf{d}^i(\mathbf{p}) - d_{mid}}{d_{max} - d_{min}} & \mathbf{d}^i(\mathbf{p}) > \tau \\ -1 & else \end{cases}, \tag{9}$$

where $\mathbf{d}^i_n(\mathbf{p})$ is the normalized depth value of pixel $\mathbf{p}$ in the normalized depth map $\mathbf{D}^i_n$, and $\mathbf{d}^i(\mathbf{p})$ is the corresponding raw depth value. The depth middle value $d_{mid} = (d_{min} + d_{max})/2.0$. The depth threshold $\tau$ is set to 0.1 to distinguish the invalid depth values. The output value $\mathbf{o}^i_n(\mathbf{p})$ of $\mathcal{F}_d$ falls in $[-1, 1]$, and we unnormalize this value via $(\mathbf{o}^i_n(\mathbf{p}) * (d_{max} - d_{min}) + d_{mid}) * \mathbf{m}(\mathbf{p})$ to obtain the denoised depth value $\mathbf{d}^i_{rf}(\mathbf{p})$, where $\mathbf{m}(\mathbf{p})$ is the mask value of pixel $\mathbf{p}$ of the full-body binary mask.

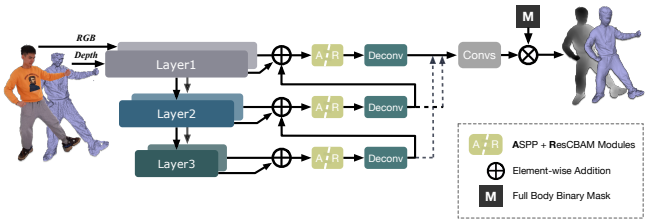
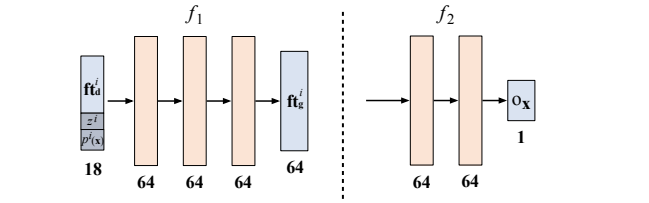
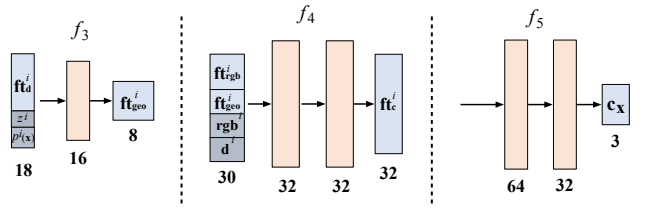



Fig. 3. The structure of our depth denoising network. Layer 1-3 correspond to the first three stages of *HRNetV2-W18-Small-v2* [Wang et al. 2021a].

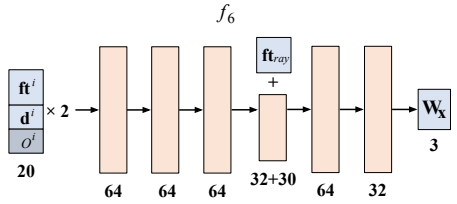
(2) In SRONet, we adopt two individual *HRNetV2-W18-Small-v2* to encode the RGB and depth images. The channel number of output RGB and depth features are set to 16, respectively. For all implicit functions $f_i (i = 1, ..., 5)$, we visualize their structures in Fig. 4(first two rows). For the hydra attention block $\mathcal{H}$, the number of input feature channels is 32.




(a) Implicit functions in *OCCNet*




(b) Implicit functions in *ColorNet*




(c) Implicit function in *Neural Blending Operation*

Fig. 4. Network architecture of our implicit functions in *OCCNet* (a), *ColorNet* (b), and *Neural Blending Operation* (c). All networks are implemented as MLPs , and each inner fully connected layer in the MLP is followed by ReLU activation. The final activation functions of $f_2$, $f_5$ and $f_6$ are the *Sigmoid* functions.

(3) In *Neural Blending Operation* $\mathcal{F}_b$, the high-resolution RGB encoder is a UNet-like network composed of $5 \times 5$, $3 \times 3$ convolutional encoders, bilinear interpolation sampling function, and skip connections. The channel number of the output high-resolution RGB feature is set to 16. Structure of the implicit function (*i.e.*, $f_6$) used in $\mathcal{F}_b$ is visualized in Fig. 4(3rd row).

| Models | *THuman2.0* [Yu et al. 2021b] Dataset | | | |
|---|---|---|---|---|
| | **PSNR** ↑ | **SSIM** ↑ | **LPIPS** $\times 10^{-1}$ ↓ | **MAE** $\times 10^{-2}$ ↓ |
| Hydra Att. → Self Att. | 34.801 | 0.964 | **0.317** | 0.465 |
| **Ours** | **34.882** | **0.969** | 0.354 | **0.392** |
| Models | *Our Real Captured Dataset* | | | |
| | **PSNR** ↑ | **SSIM** ↑ | **LPIPS** $\times 10^{-1}$ ↓ | **MAE** $\times 10^{-2}$ ↓ |
| Hydra Att. → Self Att. | 29.845 | 0.954 | 0.482 | 0.816 |
| **Ours** | **30.228** | **0.968** | **0.454** | **0.634** |

Table 1. Ablation Study on the *THuman*2.0 dataset [Yu et al. 2021b] (upper part) and our real captured data (lower part). The best results are marked in **bold**.

## 2 MORE RESULTS

### 2.1 More Experimental Results

*Comparisons of the view number.* We conduct an experiment to evaluate the view number (Fig. 2) during inference, by changing the number of input RGBD images. As the number of views increases, the texture details gradually improve, especially for the invisible or some small regions (*e.g.*, black dashed boxes). When the number of inputs is 1 or 2, these regions cannot be well handled, or our model can only work near the input views. For the single-Kinect setup (front view), the PSNR/SSIM/LPIPS of two adjacent test views (45 degrees) is 25.356/0.939/0.0588 on real-captured data in Fig.12 (in the main paper). When the number of inputs is greater than 8, there is no significant improvement in texture details, but the shadow of rendered results becomes heavier. Here, we choose to adopt the 4-view capture setting.

*Quantitative comparisons on our real-captured dataset.* Tab. 2 reports the detailed rendering comparisons for 10 independent parts (of different actions) of our real-captured dataset. It shows that our method generally outperforms 6 existing methods in terms of the PSNR and SSIM metrics.

*Hydra Attention for Feature Fusion: $\mathcal{H}$.* We evaluate the effectiveness of Hydra attention [Bolya et al. 2023] for RGB features fusion, by replacing it with the popular self-attention [Vaswani et al. 2017] (denoted as "Hydra Att.→Self Att."). The above Table shows that the quantitative performance slightly drops. Fig.14(f) (in the main paper) shows that the colors may also distort slightly. Considering that Hydra Att. contains much fewer parameters, and is an operation with linear computational complexity compared to Self Att. ($O(TD)$ *v.s.*, $O(T^2D)$), leading to a faster feature fusion, we incorporate it into our method.

### 2.2 Portrait Capturing Setting and More Results

*Settings of Portrait Reconstruction and Rendering.* We use 3 Azure Kinect-V4 sensors to capture the portrait RGBD images. These three cameras are placed in front, left-front, and right-front of the subject, respectively, on the same horizontal line. The distances between the two adjacent cameras are approximately 40cm. The subject is invited to sit in front of and face the middle camera to make facial expressions for capture. We set the number of input views of SAILOR to 3 and apply it to process the inputs directly (*i.e.*, without re-training or fine-tuning).

We provide more of our portrait reconstruction and rendering results, using Point Spread Okulo P1S camera. Fig. 5 illustrates the

| Dataset (size) | Index | *Methods* | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | PixelNeRF | IBRNet | MPSNeRF | NHP | NPBG++ | PIFu(RGBD) | **Ours** |
| **Rocking & Walking** (230) | **PSNR** ↑ | 27.012 | 28.344 | 27.930 | 28.486 | 27.769 | 28.448 | **30.920** |
| | **SSIM** ↑ | 0.909 | 0.934 | 0.928 | 0.938 | 0.932 | 0.948 | **0.965** |
| | **LPIPS** ↓ | 0.146 | 0.121 | 0.114 | 0.113 | 0.0964 | 0.0688 | **0.0451** |
| **Kung Fu** (230) | **PSNR** ↑ | 22.655 | 24.131 | 22.760 | 22.709 | 24.148 | 27.147 | **28.624** |
| | **SSIM** ↑ | 0.903 | 0.926 | 0.913 | 0.927 | 0.913 | 0.953 | **0.958** |
| | **LPIPS** ↓ | 0.140 | 0.110 | 0.113 | 0.110 | 0.0879 | 0.0422 | **0.0350** |
| **Rocking & Undressing** (150) | **PSNR** ↑ | 27.038 | 29.026 | 29.265 | 29.325 | 28.898 | 28.133 | **32.139** |
| | **SSIM** ↑ | 0.915 | 0.942 | 0.942 | 0.946 | 0.943 | 0.955 | **0.968** |
| | **LPIPS** ↓ | 0.152 | 0.126 | 0.111 | 0.116 | 0.0976 | 0.0808 | **0.0453** |
| **Swinging_1** (110) | **PSNR** ↑ | 20.797 | 23.706 | 22.350 | 20.363 | 23.879 | 27.833 | **28.389** |
| | **SSIM** ↑ | 0.905 | 0.925 | 0.921 | 0.930 | 0.900 | 0.954 | **0.962** |
| | **LPIPS** ↓ | 0.157 | 0.109 | 0.122 | 0.118 | 0.0833 | **0.0252** | 0.0259 |
| **Swinging_2** (120) | **PSNR** ↑ | 21.938 | 24.669 | 24.055 | 22.986 | 24.665 | 27.434 | **29.065** |
| | **SSIM** ↑ | 0.890 | 0.913 | 0.910 | 0.919 | 0.915 | 0.938 | **0.948** |
| | **LPIPS** ↓ | 0.151 | 0.103 | 0.108 | 0.0993 | 0.0693 | **0.0317** | 0.0344 |
| **Punching** (120) | **PSNR** ↑ | 25.009 | 26.737 | 26.054 | 25.440 | 27.256 | 29.338 | **29.931** |
| | **SSIM** ↑ | 0.916 | 0.933 | 0.926 | 0.936 | 0.935 | 0.947 | **0.966** |
| | **LPIPS** ↓ | 0.128 | 0.098 | 0.102 | 0.0961 | 0.0677 | 0.0320 | **0.0294** |
| **Swinging & Walking** (126) | **PSNR** ↑ | 20.669 | 23.640 | 22.266 | 21.021 | 24.025 | 27.302 | **30.036** |
| | **SSIM** ↑ | 0.916 | 0.932 | 0.928 | 0.937 | 0.918 | 0.954 | **0.968** |
| | **LPIPS** ↓ | 0.153 | 0.104 | 0.117 | 0.106 | 0.0677 | 0.0280 | **0.0275** |
| **Lifting Legs** (120) | **PSNR** ↑ | 22.023 | 24.387 | 23.906 | 22.612 | 24.960 | 27.572 | **29.060** |
| | **SSIM** ↑ | 0.898 | 0.917 | 0.915 | 0.921 | 0.915 | 0.938 | **0.955** |
| | **LPIPS** ↓ | 0.156 | 0.107 | 0.111 | 0.108 | 0.0741 | **0.0319** | 0.0348 |
| **Stretching_1** (106) | **PSNR** ↑ | 23.950 | 25.853 | 25.259 | 24.173 | 25.508 | 29.062 | **30.224** |
| | **SSIM** ↑ | 0.905 | 0.924 | 0.922 | 0.928 | 0.918 | 0.953 | **0.956** |
| | **LPIPS** ↓ | 0.143 | 0.106 | 0.105 | 0.109 | 0.0760 | **0.0298** | 0.0360 |
| **Stretching_2** (200) | **PSNR** ↑ | 24.568 | 26.927 | 25.939 | 24.991 | 26.879 | 30.050 | **30.597** |
| | **SSIM** ↑ | 0.917 | 0.936 | 0.938 | 0.941 | 0.935 | 0.953 | **0.967** |
| | **LPIPS** ↓ | 0.140 | 0.102 | 0.102 | 0.104 | 0.0707 | **0.0308** | 0.0353 |

Table 2. Comparisons of rendering results on our captured dataset, produced by our method and existing methods, *i.e.*, PixelNeRF [Yu et al. 2021a], IBRNet [Wang et al. 2021b], MPSNeRF [Gao et al. 2022], NHP [Kwon et al. 2021], NPBG++ [Rakhimov et al. 2022] and PIFu(RGBD) [Saito et al. 2019]. The best and second best results are marked in **bold** and underline, respectively.

results. This demonstrates the robustness of our method for different capturing cameras, and the ability to reconstruct some exaggerated expressions and some attached objects (*e.g.*, pillow).

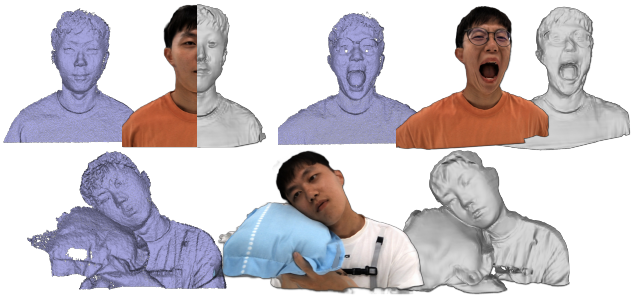


Fig. 5. We apply SAILOR to portrait rendering and reconstruction using Point Spread Okulo P1S camera. Note that our method can handle some exaggerated expressions and some attached objects (*e.g.*, pillow). The displayed images are the input depth maps, our rendering, and reconstruction results, respectively. The rendering results are selected from novel views.

## 3 THE INTERACTIVE GUI

We design an interactive GUI for the usage of SAILOR, as shown in Fig. 6. In our GUI, SAILOR can generate free-view rendering results with a latency of less than 100ms. When the camera-target distance in the novel view is less than 80% of the average distance in the input views, exceeding the original sampling rate, we will perform bilinear interpolation to obtain the zoomed rendering results.

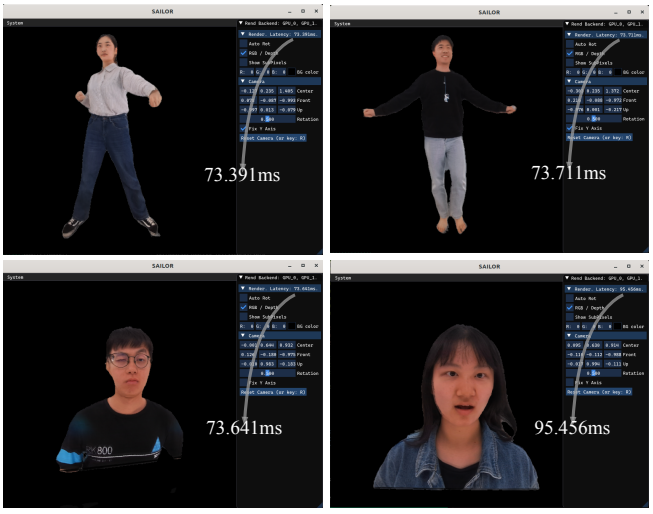



Fig. 6. Visualization of our interactive GUI. Our system can render full-body (upper row) and portrait (bottom row) videos in novel views within 100ms per frame.

## REFERENCES

John Amanatides, Andrew Woo, et al. 1987. A fast voxel traversal algorithm for ray tracing.. In *Eurographics*.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, and Judy Hoffman. 2023. Hydra attention: Efficient attention with many heads. In *Eur. Conf. Comput. Vis.*

Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. 2017. Rethinking atrous convolution for semantic image segmentation. *arXiv:1706.05587* (2017).

Zheng Dong, Ke Xu, Ziheng Duan, Hujun Bao, Weiwei Xu, and Rynson Lau. 2022. Geometry-aware Two-scale PIFu Representation for Human Reconstruction. In *Adv. Neural Inform. Process. Syst.*

Péter Fankhauser, Michael Bloesch, Diego Rodriguez, Ralf Kaestner, Marco Hutter, and Roland Siegwart. 2015. Kinect v2 for mobile robot navigation: Evaluation and modeling. In *ICAR*.

Xiangjun Gao, Jiao long Yang, Jongyoo Kim, Sida Peng, Zicheng Liu, and Xin Tong. 2022. MPS-NeRF: Generalizable 3D Human Rendering from Multiview Images. *IEEE Trans. Pattern Anal. Mach. Intell.* (2022).

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. arXiv:1412.6980

Youngjoong Kwon, Dahun Kim, Duygu Ceylan, and Henry Fuchs. 2021. Neural human performer: Learning generalizable radiance fields for human performance rendering. In *Adv. Neural Inform. Process. Syst.*

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in Pytorch. In *Adv. Neural Inform. Process. Syst.*

Ruslan Rakhimov, Andrei-Timotei Ardelean, Victor Lempitsky, and Evgeny Burnaev. 2022. NPBG++: Accelerating Neural Point-Based Graphics. In *IEEE Conf. Comput. Vis. Pattern Recog.*

Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. 2019. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *Int. Conf. Comput. Vis.*

Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Adv. Neural Inform. Process. Syst.*

Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, and Bin Xiao. 2021a. Deep High-Resolution Representation Learning for Visual Recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* (2021).

Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P Srinivasan, Howard Zhou, Jonathan T Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. 2021b. Ibrnet: Learning multi-view image-based rendering. In *IEEE Conf. Comput. Vis. Pattern Recog.*

Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. on Image Processing* (2004).

Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. 2018. Cbam: Convolutional block attention module. In *Eur. Conf. Comput. Vis.*

Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. 2021a. pixelnerf: Neural radiance fields from one or few images. In *IEEE Conf. Comput. Vis. Pattern Recog.*

Tao Yu, Zerong Zheng, Kaiwen Guo, Pengpeng Liu, Qionghai Dai, and Yebin Liu. 2021b. Function4D: Real-time Human Volumetric Capture from Very Sparse Consumer RGBD Sensors. In *IEEE Conf. Comput. Vis. Pattern Recog.*