Real-Time Neural Rendering in VR Environment

Shengqu Cai Weirong Chen Mingyang Song Tianfu Wang
ETH Zürich
{shecai, weirchen, misong, tianfwang}@ethz.ch

Supervised by Dr. Sergey Prokudin

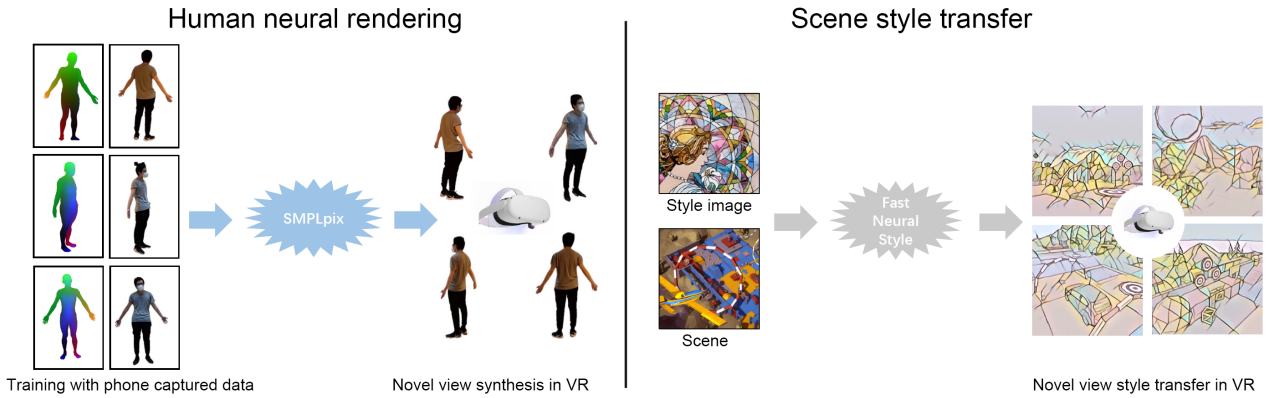

Figure 1. General real-time neural rendering pipeline for VR. Left: human neural rendering. Right: scene style transfer.

Abstract

*Neural rendering, the data-driven approach of using neural networks to synthesize imagery, has achieved photorealistic results for generating images of scenes and humans. However, there has not been a deep investigation on integrating these neural rendering models in an interactive virtual/augmented reality (VR/AR) setting. For this project, we aim to build a pipeline capable of displaying photorealistic scenes in VR devices using neural rendering in real-time. Our work consists of **1)** Collecting data and training models such as SMPLpix specifically for VR camera settings. **2)** Integration of human neural rendering model into Unity & VR device. **3)** Optimization of the pipeline for potential issues running with VR devices. Our results show interactive, real-time, and good quality neural rendering effects for scene style transfer and human rendering on an Oculus Quest 2.*

1. Introduction

There have been increasingly popular demands for generating high quality, photorealistic, and interactive 3D scenes for mixed reality applications such as realistic avatars [5], gaming [3], and immersive communications [30, 31]. Currently, most existing methods use traditional computer graphics and rendering pipelines. These pipelines, including rasterization and ray tracing, rely on explicit, often physically-based, modeling for geometry, material, and light transport [33, 51]. These explicit representations allow parametrized and editable scenes, giving more flexibility for development. However, these models are not effective in terms of rendering photorealistic scenes. Firstly, obtaining a high-quality geometry model, including its 3D data, material profile, and color texture, requires expensive tracking and complex photogrammetry modeling. Additionally, accurately simulating light transport, especially indirect lighting and global illumination, are slow to converge and computationally intensive. The problem of achieving photorealism becomes even more challenging when rendering human models. This is due to the "uncanny valley" effect [12], where small distinctions in the generated rendering result in negative user experiences. Therefore, although rendering photorealistic scenes using traditional pipelines can be obtainable, it becomes very challenging under the constraint of real-time, interactive applications.

Recent years were marked by incredible progress by the alternative approach of neural rendering [42, 43]. Neural

rendering does not necessarily use an explicit scene representation, and instead, tries to generate imagery using a data-driven approach. Current work in neural rendering has already achieved photorealism in areas challenging for traditional graphics pipelines. For rendering general outdoor scenes, convolutional networks [38] are being employed to enhance the quality of video games renderings to photorealistic results, adding subtle details in illumination and texture. In the realm of photorealistic human rendering, SIMPLpix [36] builds on pix2pix [18,47], a generative adversarial network (GAN) [13] driven image-to-image translation model, to achieve realistic texture coloring of the human body, while maintaining the flexibility and user controllability of using an explicit 3D human mesh. However, despite the tremendous advancements in neural rendering, there has not been, to the best of our knowledge, a study on implementing these neural rendering models in a VR/AR setting. This is partly because current neural rendering frameworks are often computationally intensive and thus face challenges integrating to the VR pipeline. Our project aims to explore this direction.

Our goal is to build a pipeline capable of presenting interactive virtual scenes and photorealistic human avatars in a VR setting and investigate the bottlenecks and challenges when building such pipelines. Our pipeline integrates the VR display and user interaction platforms, established VR development platforms, and state-of-the-art neural rendering algorithms for humans and scenes. For rendering humans, we built on the work of SIMPLpix [36], making crucial adjustments and optimizations to its network architecture and objective functions tailored for the VR setup. For rendering virtual scenes, we present a style transfer neural rendering pipeline that allows the virtual scene to be transformed based on the visual style of a given style image. We use Unity, a popular development platform for graphics and VR applications, to set up our virtual scenes and run our neural rendering models as a post-processing pipeline. Our final results show neural style transfer and human neural rendering running in interactive frame rates ($>$ 20 FPS) in an Oculus Quest 2 VR headset. The VR application we built using this pipeline demonstrates discernible and good quality style transfer for virtual scenes and human animations in novel views. Our work shows the feasibility of deploying lightweight neural rendering models in a VR setting.

2. Related work

2.1. Image to image neural translation

In traditional computer graphics, creating a photorealistic scene requires expensive content creation, i.e., meshes, light sources, and physically based materials. Moreover, rendering global illumination scenes with raytracing or photon mapping is time consuming and difficult to converge. To solve this problem, many CNN-based models, e.g. deep render denoising models [46, 53] have been proposed to post-process the rendered images. Machine learning based rendering can bypass the explicit scene modeling and synthesize novel images with deep generative models. One direction of neural rendering is the image-to-image translation. Semantic photo synthesis [8, 19, 48] maps a user specified image with semantic labels to a photorealistic image. These methods are underconstrained, since the semantic maps are label silhouettes without geometry information and ambiguous. Instead of synthesizing images from silhouettes, some papers [17,38] enhance the rendered images. Stephan et al. uses more representative scene features, i.e., G-buffers, to enforce the consistency between the input and output images [38]. Another direction of image-to-image neural rendering is style transfer, which converts the style of one image to another style. Early methods require training one network per style [11, 44, 45]. Most recently, some papers [9, 25] propose to train a single network that can transfer images to multiple styles.

To infer a model in Unity, all networks have to be converted to Open Neural Network Exchange (ONNX) format, which only supports limited kinds of operators. Moreover, the requirement of efficiency constrains the depth of the network. For the style transfer part of our project, we use one network for each style to perform multi-modal real-time image-to-image translation.

2.2. Human centric neural rendering

Unlike general objects or scenes, rendering photorealistic humans is especially challenging due to the so-called "uncanny valley" effects, which refers to the phenomenon that looking at humanoid objects imperfectly resembled actual human beings will create strangely familiar feelings of eeriness and revulsion. Traditional computer graphics pipelines rely on explicit, parameterized representation with the control of pose, lighting, material, and shape. However, such classical methods are usually built on top of the complex systems for physical and optical modeling. Recently, there is a new trend of rendering photorealistic digital avatars, which adopts the idea of implicit representation [28, 30, 40, 50]. While these methods can achieve photorealistic results without computationally intensive modeling, they are usually optimized for a single object only and require retraining or finetuning when applied to the novel mesh. SMPLpix [36] extends the flexibility of neural avatars by integrating the deformable 3D human models SMPL [29], allowing customized control of body pose and shape.

Another direction of achieving photorealistic human rendering is to make the mesh rendering pipeline fully differentiable so that it can be optimized end-to-end and learn the proper implicit representation [22, 27]. Neural point-

based rendering [4] has shown the superiority of a fully differentiable rendering pipeline over image-to-image post-processing. With the recent success of NeRF [32], Neural Actor [26] further combines differential rendering pipeline with deformable 3D human models to control the human pose.

3. Method

3.1. Data acquisition

We capture our data using a conventional smartphone monocular camera, and our data consists of video recordings of a person standing. Although we capture our data in the wild, we apply some constraints to our scenes to ensure good quality data for training. First, we require the person standing in the scene to be static in the world coordinates while keeping a neutral A-pose. This simplified scene ensures that all body parts of the person are visible by the camera throughout the capture process, and ambiguities caused by occlusion are minimized. Second, during the capture process, we move the camera to capture the body at viewpoints similar to the viewpoint of a VR user interacting with a virtual human model. We try to keep close viewing distances to the captured subject. Additionally, to simulate the viewpoint of a VR user looking up and down, we diversify the elevation and azimuth of the camera as we move around the subject while keeping the subject at the center of the frame. Thirdly, we make sure that we have relatively uniform lighting in the scene environment, no distracting objects in the background, and that the captured person is wearing clothes with uniform colors with no shiny or reflective materials. This constraint in the scene lighting makes it easier for the model to register body parts to the correct texture while also giving us accurate segmentation of the captured human from the images.

Once we have captured our video, we process the frames to obtain the training data that is compatible with our VR framework (Figure 2). We first do a segmentation pass for each frame of the video so that we are only left with the person being captured, while other parts of the video are replaced with white background. We then use the tool kit provided by the authors of SMPLpix [36] to transform the video into an SMPL mesh dataset. For each frame in the video, we first use the image-based pose estimation model OpenPose [7] to extract the 2D body key points of the person in the frame. We can then infer the 3D human model using the SMPLify-X method [34], which uses an SMPL-X 3D human model to fit the 2D body key points identified. Finally, for each frame, we get the deformed 3D SMPL-X human model inferred as a 3D mesh and the camera projection data that allows us to project the mesh back to the same location on the image.

Our last step to generating our dataset is to generate the SMPL-X human model images for each video frame using Unity. The SMPL-X model uses vertex colors to identify the body location of the human. However, Unity only supports texture maps instead of vertex colors to the best of our knowledge. As a workaround, we implemented custom mesh processing scripts in the 3D graphics software Blender that allows us to bake the vertex colors of the SMPL-X mesh into texture maps. We also implemented automation scripts in Unity to project the textured meshes in the correct image location and capture the SMPL-X images. By capturing the SMPL-X human model in Unity, we ensure that the neural rendering model input and the input in the VR application have the same color profile.

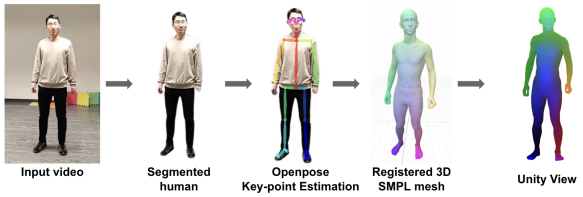

Figure 2. Processing capture video to VR compatible training data

3.2. Model training

We build upon the open-sourced SMPLpix [36] implementation and optimize the method specifically for VR usage. We describe our modified SMPLpix [36] pipeline in the following sections.

3.2.1 Network architecture

Due to the operator constraints of ONXX and Barracuda, the original open-sourced SMPLpix [36] network cannot be imported to Unity. Additionally, the real-time objective requires a light-weight network design. We therefore design a simple U-Net architecture and verify its usability on SMPLpix [36]'s open sourced data. We provide details regarding the network architecture in Table 1.

3.2.2 Data augmentation

An VR environment offers diverse viewpoint variations. To train a model capable of synthesizing novel views similar to those in VR, we enhance the data augmentations used in SMPLpix [36]. Namely, we use camera rotation of $\pm$ 60°, camera translation of $\pm$ 100 pixels and random scaling factors from 0.2 to 2.

3.2.3 Training objectives

To better accommodate VR settings, we use a hybrid set of training objectives including **VGG feature reconstruction**,

input	layer	output
x	ReLUInstanceNormConv(filter=$9 \times 9 \times 8$, stride=1)	conv1_y
conv1_y	ReLUInstanceNormConv(filter=$3 \times 3 \times 16$, stride=2)	conv2_y
conv2_y	ReLUInstanceNormConv(filter=$3 \times 3 \times 32$, stride=1)	conv3_y
conv3_y	ResidualBlock(n_channels=32)	res1_y
res1_y+conv3_y	ResidualBlock(n_channels=32)	res2_y
res2_y+res1_y+conv3_y	ResidualBlock(n_channels=32)	res3_y
res3_y+res2_y+res1_y+conv3_y	ResidualBlock(n_channels=32)	res4_y
res4_y+res3_y+res2_y+res1_y+conv3_y	ResidualBlock(n_channels=32)	res5_y
res5_y+conv3_y	ReLUInstanceNormUpsampleConv(filter=$3 \times 3 \times 16$, stride=1, upsample=2)	deconv1_y
deconv1_y+conv2_y	ReLUInstanceNormUpsampleConv(filter=$3 \times 3 \times 8$, stride=1, upsample=2)	deconv2_y
deconv2_y+conv1_y	Conv(filter=$9 \times 9 \times 3$, stride=1)	RGB

Table 1. Network architecture.

LPIPS reconstruction, **pix2pixHD GAN loss** and **Head loss**, elaborated as follows.

VGG reconstruction loss. Following recent advances in content reconstruction [21, 36], instead of using standard $\mathcal{L}1$ or $\mathcal{L}2$ reconstruction in the RGB domain, we use $\mathcal{L}1$ reconstruction of features from VGG16 pre-trained on ImageNet [10], calculated by $\mathcal{L}1$ distances of the activations. This gives similar effects as using a GAN loss [14], without employing adversarial training [8]. Following SMPLpix [36], the VGG reconstruction loss is computed as:

$$\mathcal{L}_{\mathrm{VGG}}(I_{\mathrm{gt}}, I_{\mathrm{recon}}) = \sum_{i=0}^{5} \frac{1}{2^{(5-i)}} \left\| f_{\mathrm{VGG}}^{(i)}(I_{\mathrm{gt}}) - f_{\mathrm{VGG}}^{(i)}(I_{\mathrm{recon}}) \right\| \quad (1)$$

where $f_{\mathrm{VGG}}^{(i)}(I)$ refers to activations at layer i of the pre-trained VGG network, I_{gt} and I_{recon} refers to the ground truth image and the reconstructed image respectively.

LPIPS reconstruction loss. VGG network used to compute VGG loss is trained on ImageNet [10] for classification task, which indicates that its learned features may not be entirely suitable for reconstruction tasks. To address this, Learned Perceptual Image Patch Similarity (LPIPS) [52] was proposed initially as an evaluation metric. Instead of training for ImageNet [10] classification, deep networks used to extract features in LPIPS [52] are trained specifically with a dataset of human perceptual similarity judgments, which more appropriately reflects human perception preferences. Similar to VGG features, LPIPS [52] can be used as a reconstruction objective, and is generally a stronger objective than standard VGG loss as mentioned in [20, 37]. LPIPS [52] loss is computed as:

$$\mathcal{L}_{\mathrm{LPIPS}}(I_{\mathrm{gt}}, I_{\mathrm{recon}}) = \sum_{i} \tau^{(i)}(\phi^{(i)}(I_{\mathrm{gt}}) - \phi^{(i)}(I_{\mathrm{recon}})) \quad (2)$$

where ϕ is the feature extractor network(for which we use pre-trained AlexNet [24] provided by LPIPS [52] authors) and τ is an operator normalizing and transforming extracted deep features across the ith layer to scalar score.

pix2pixHD [47] GAN loss. Even though not provided in SMPLpix [36]'s open sourced implementation, we found adding pix2pixHD [47] style GAN training to be helpful. Define the neural renderer as the generator G, and corresponding discriminators as D_1, D_2, D_3 operating under different resolutions, pix2pixHD [47] GAN objective can be written as:

$$\min_{G} \max_{D_1,D_2,D_3} \mathcal{L}_{\mathrm{GAN}}(G, D_1, D_2, D_3), \text{where}$$
$$\mathcal{L}_{\mathrm{GAN}}(G, D_1, D_2, D_3) = \sum_{i=1,2,3} \Big[\mathop{\mathbb{E}}_{\mathrm{x,y}} \left[\log D_i\left(\mathrm{x}, \mathrm{y}\right) \right] + \mathop{\mathbb{E}}_{\mathrm{x}} \left[\log\left(1 - D_i\left(\mathrm{x}, G\left(\mathrm{x}\right)\right)\right) \right] \Big] \quad (3)$$

Head loss. With the previous methods, we are able to reconstruct the human body with high fidelity. However, the multi-scale settings in VR and mis-alignment between mesh and image makes it extremely difficult to capture the fine details around face areas. Inspired by LookinGood [30], we attempt to resolve this issue by introducing a head loss into our system. Due to the lack of ground truth segmentation mask, we apply a color filter on the projected mesh to obtain a rough location of the head, and crop a bounding box around it. It is then resized to 512×512 and fed into the network for reconstruction.

3.3. Framework

The goal of our project is to develop an application that can apply human/scene neural rendering to the VR setting. To achieve this, we need to integrate the game engine, the neural rendering module, and the VR display altogether. With the rich functionalities and wide community support,

Unity [15] is selected as our developing engine, which provides the 3D platform and VR toolkits for developing general VR applications. Neural rendering is achieved by the Barracuda package developed by the Unity-Technologies, allowing users to perform model inference using a pre-trained network via Unity. In addition, we are given the Oculus Quest 2 as the VR hardware, which has wide popularity among the young generation and it is very easy to use. It supports both all-in-one (compute on VR) mode and oculus-link mode (compute on PC), and we use the latter for this project to achieve a higher frame rate.

3.3.1 Unity

Unity [15] is one of the most widely used video game engines for 3D game development, architecture, engineering, construction, etc. With cross-platform support, the app can be easily deployed to a variety of desktop, mobile, console, and virtual reality platforms. For developing VR applications, Unity contains the XR Interaction Toolkit [2] which provides the high-level, component-based, interaction system. It contains the basic framework that integrates 3D and UI with the Unity events under the VR setting so that we can easily obtain the camera pose and controllers' actions from the Oculus Quest 2 in real-time. The remaining components, including events and UI, are similar to developing a general 3D game using Unity.

3.3.2 Barracuda

Another reason that we choose Unity as the development platform is that it contains the Barracuda package [1] for neural inference. The Barracuda package is a lightweight cross-platform neural network inference library for Unity, supporting both CPU and GPU network inference. Given a pre-trained neural rendering model in PyTorch, the first step is to first convert it to ONNX format, which can be recognized by the Barracuda package. In our application, Barracuda is used as a post-processing module that takes the raw rendered texture from the Unity built-in render pipeline, post-processes it with the pre-trained network inference, and copies the result back to the output texture. As the result, users can see the neural rendered result from the current camera view.

3.3.3 Pre-processing

Before passing the raw rendered results into the network, we need to perform pre-processing steps of down-sampling and handling the distortion. The original view obtained from Oculus is of resolution 1600×1600, much higher than the resolution of our training data (512×512). We need to down-sample the resolution to 512 as the pre-trained human rendering model is very sensitive to the input size. Besides, the raw rendered result from Oculus also contains a distortion effect that has an irregular round and non-empty boundary. Without transferring the background to a regular square shape or filling it with the white color as the same in the trainset, the model's output will become very blurry. Since we do not have the distortion parameters, the background is handled by cropping the central 720×720 part of the whole image, which directly results in a square and white sub-figure.

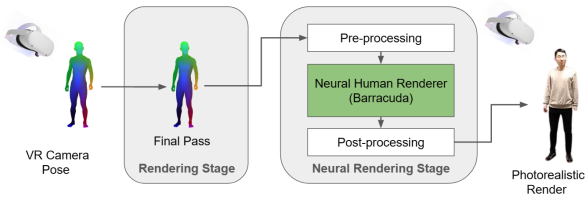

Figure 3. Human Neural Rendering Pipeline

3.3.4 Pipeline

The whole pipeline for human neural rendering is shown in Figure 3. Given the real-time VR camera pose and SMPL human mesh, we use the Unity multi-pass built-in render pipeline to obtain the raw rendered view from the corresponding camera pose and store it in the input texture. In the neural rendering stage, we prepare a pre-trained human neural rendering network in ONNX format and load it to the Barracuda. The input texture is firstly pre-processed with the cropping and down-sampling, then is passed to the neural rendering module for inference, and is finally post-processed up-sampling and copied back to the output texture. As the result, the user can see the photorealistic human in the VR from multiple views.

Besides the human neural rendering, the proposed pipeline can also be generalized to other neural post-processing tasks such as scene style transfer. The only modification needed is to replace the human mesh with the scene mesh and provide a pre-trained style transfer model in ONNX format.

3.4. Implementation

We make this project into an application on the Windows platform. Users can use a USB-C cable or air link of Oculus to receive the image from the computer and interact with the virtual environment. Our UI allows users to apply any neural network model to any meshes. The application has good visual quality in the style transfer part and successfully generates novel views in the neural human rendering part. The program achieves real-time performance with 20 FPS on NVIDIA GeForce RTX 2060 Max-Q.

Figure 4. Human neural rendering of team members

Figure 5. Human neural rendering of the dynamic mesh

4. Experiment

4.1. Training detail

Following SMPLpix [36], we use a two-stage training strategy. In stage 1, we train the network with only VGG loss and LPIPS loss. In stage 2, we additionally introduce pix2pixHD GAN loss. Since pix2pixHD GAN loss is not provided in SMPLpix [36]'s open-sourced implementation, we re-implement this part following pix2pixHD [47], while use settings from original SMPLpix [36] paper. Numerically, we can write the overall loss as:

$$\begin{aligned}\mathcal{L}(I_{\text{gt}},I_{\text{recon}}) &= \mathcal{L}_{\text{VGG}}(I_{\text{gt}},I_{\text{recon}}) \\ &+ \lambda_{\text{LPIPS}}\mathcal{L}_{\text{LPIPS}}(I_{\text{gt}},I_{\text{recon}}) \\ &+ \lambda_{\text{adv}}\sum_{k=1,2,3} D_i(I_{\text{recon}}) \\ &+ \lambda_{\text{head}}\mathcal{L}_{\text{head}}(I_{\text{gt}},I_{\text{recon}})\end{aligned} \tag{4}$$

where in stage 1, we use $\lambda_{\text{LPIPS}} = 0.1$ and $\lambda_{\text{adv}} = 0$; In stage 2, we use $\lambda_{\text{LPIPS}} = 0.1$, $\lambda_{\text{adv}} = 0.1$ and $\lambda_{\text{head}} = 0.1$. Models are trained with Adam optimizer [23] with an initial learning rate of 1.0e-3 using ReduceLROnPlateau scheduler with patience 50. We run each stage for 48 hours on a single NVIDIA GTX TITAN XP with batch size 6.

4.2. Qualitative results

Our pipeline can be generalized to arbitrary neural post-processing. We apply such a pipeline to human neural rendering and style transfer. Both applications can run in real time and have a promising visual quality.

4.2.1 Human rendering

We trained multiple models for each person in our group, which can be seen in Figure 4. While most of the training pairs are captured in the standing poses, Figure 5 shows the generalization ability of our networks to arbitrary poses. Our pipeline can not only be utilized to static human mesh but also dynamic human mesh.

As can be seen in Figure 6, our networks can restore the wrinkles on cloth and the pattern on shoes. When rendering some poses of the dynamic avatar, the texture of arms may disappear. As has been described before, our training set does not contain these poses, and this failure can be alleviated by capturing a new training set with more kinds of poses. Since in our pipeline, we down-sample the input and up-sample the output, the face part is blurry. Another reason for such an artifact is that, to eliminate the effect of the light source and surface material, our input only contains the color information, the geometry information is lost.

4.2.2 Scene style transfer

Considering the requirement for efficiency and some limitations of ONNX, we use a simple U-net described in [21] as our style transfer model. The input images, output images and corresponding style images are shown in Figure 7a and Figure 7b.

The style transfer part of our pipeline supports multiple resolution input. Using higher resolution as input can preserve more patterns and edges of the input images.

4.3. Quantitative results

For quantitative results, we compare our modified model with vanilla open-sourced SMPLpix [36] implementation. We capture data and train a model for each of our four team members.

4.3.1 Data split

To ensure a fair comparison, we select 100 frames of diverse viewpoints for each instance as the test set. Each frame is then randomly augmented using the same transformation parameters for training. Data samples for each instance are repeatedly augmented for 50 times to ensure a small bias, yielding 5000 test images per instance. We train and evaluate our model on resolution 512×512.

4.3.2 Evaluation metric

Follow prior works [35, 36], we report PSNR, SSIM [49] and LPIPS [52] for novel view synthesis results. Since as a human rendering model, certain body parts, e.g. faces, will play an important role in human perception which cannot be fully captures with reconstruction metrics, we additionally report fidelity results using GAN metrics including FID [16], KID [6] and IS [39]. We briefly introduce these

Figure 6. Rendered human from various camera viewpoints

(a) Park scene

(b) Building scene

Figure 7. Scene style transfer

evaluation metrics here and refer our readers to the corresponding papers for details.

IS: Before FID and KID, Inception Score [39] was generally used as the metric for evaluating generated sample quality and diversity. This is done by applying a Inception-V3 network [41] pre-trained on ImageNet [10] to the generated images and calculating the IS via the conditional label distribution $p(y|\mathbf{x})$ with the marginal label distribution $p(y) = \int p(y|\mathbf{x})p\left(\mathbf{x} = G(z)\right) dz$ as follows:

$$\mathrm{IS}(\mathbf{x}) = \exp\left(\mathbb{E}_{\mathbf{x}}\left[D_{KL}\left(p(y|\mathbf{x}) \,||\, p(y)\right)\right]\right).$$

The conditional label distribution $p(y|x)$ should have low entropy (so that the generated images contain meaningful objects) and marginal label distribution $p(y)$ should have high entropy (so that the generated images are diverse), thus leading to a higher score value.

FID: Instead of evaluating on the generated images alone, Frechet Inception Distance additionally considers the real images used for training by comparing their distributions. It utilizes the Inception-V3 network [41] pre-trained on Imagenet [10] to extract image features too, and then compares the multidimensional Gaussian distributions $\mathbb{N}(\boldsymbol{\mu}_r, \boldsymbol{\Sigma}_r)$ and $\mathbb{N}(\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g)$ of the features from real and generated images respectively. FID is then calculated as follows:

$$\mathrm{FID} = |\mu_g - \mu_r|^2 + \mathrm{tr}(\Sigma_g + \Sigma_r - 2(\Sigma_g\Sigma_r)^{1/2})$$

KID: Similarly to FID, Kernel Inception Distance is calculated by squared Maximum Mean Discrepancy (MMD) with a cubic polynomial kernel between Inception representations to achieve unbiased estimation.

4.3.3 Result

We show evaluation results on the four instances in Table 2. We can conclude that our modified model outperforms the vanilla open-sourced model for most metrics on our four instances.

Instance	Method	reconstruction metrics			GAN metrics		
		PSNR ↑	SSIM [49] ↑	LPIPS [52] ↓	FID [16] ↓	KID [6] ↓	IS [39] ↑
Instance 1	Vanilla	26.170	**0.964**	0.028	156.079	0.146	2.830
	Modified	**26.235**	0.963	**0.018**	**152.828**	**0.142**	**2.891**
Instance 2	Vanilla	27.798	0.961	0.039	**168.427**	**0.167**	2.768
	Modified	**28.644**	**0.966**	**0.013**	181.967	**0.167**	**2.983**
Instance 3	Vanilla	28.310	0.971	0.021	153.073	0.144	3.275
	Modified	**28.962**	**0.976**	**0.013**	**149.920**	**0.141**	**3.471**
Instance 4	Vanilla	**31.762**	**0.984**	0.016	170.880	0.244	3.090
	Modified	30.692	0.981	**0.009**	**164.245**	**0.150**	**3.234**

Table 2. Quantitative results on the four instances.

Method	reconstruction metrics			GAN metrics		
	PSNR ↑	SSIM [49] ↑	LPIPS [52] ↑	FID [16] ↓	KID [6] ↓	IS [39] ↑
without LPIPS [52] loss	**27.942**	0.964	0.042	170.233	0.160	2.700
without head loss	26.101	0.963	0.020	156.347	**0.137**	**3.093**
without GAN loss	27.769	**0.972**	0.032	172.776	0.174	2.759
full model	26.235	0.963	**0.018**	**152.828**	0.142	2.891

Table 3. Ablation study results.

Figure 8. Learned face rendering (top) and body rendering (bottom) with head loss.

4.4. Ablation studies

We perform a thorough ablation study to verify our design choices by removing the key components one by one and training models under identical settings as the full model, shown in Table 3. We see that as expected, VGG loss and LPIPS [52] loss simulates part of GAN training results. Therefore, the improvement achieved by utilizing GAN training is less significant. In addition, using LPIPS [52] and GAN loss naturally implies trade-off from reconstruction and the capture of more fine details. We can also observe that unlike in LookinGood [30], head loss does not benefit the model. Figure 8 shows that even though the head avatar itself can be jointly learned, the effect is not successfully transferred to whole body rendering. This is potentially due to the lack of supervision given only the colored mesh 2D projection, which is lack of connection with the whole body model.

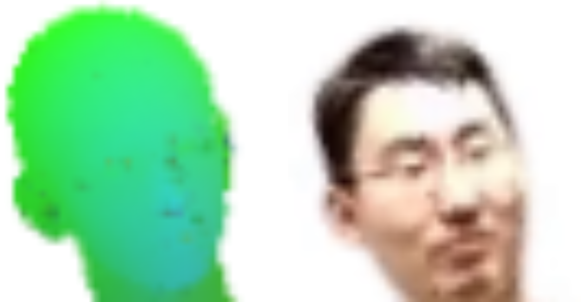

Figure 9. Mis-alignment between mesh (left) and ground truth image (right), zoomed in face area.

5. Conclusion and Future work

In this project, we integrate the neural rendering to the Unity render pipeline and VR device. Our pipeline can use an arbitrary neural network as a kind of post-processing before the images are sent to the VR headset. We implement neural human rendering and style transfer under our pipeline. One of the characteristics of VR applications is that, the camera position varies a lot when the user is walking or shaking head. To make sure the rendered human is correct and consistent, we create a training data collection pipeline inside Unity and add corresponding augmentations during training to model potential various viewpoint changes. Additionally, we add LPIPS loss and GAN loss to restore more details of humans.

Our next step is to improve the quality of neural human rendering in VR, to solve the problem that the texture of rendered humans would be incorrect when the camera is too close or far away. We empirically find that large scale variance dramatically harms the performance of our model. One way to resolve this is to train multiple networks to take care of different scale range separately. During inference stage, corresponding network can be automatically selected based on user's distance to the mesh.

Another issue is the mis-alignment of the extracted mesh and RGB image due to using simple colored mesh, shown in Figure 9. This further increases the difficulty of accurately render fine details, e.g. face details. This is solvable by providing more guidance for the neural renderer other than only colored mesh, e.g. translate a textured human mesh or colored sparse points cloud [36] to photorealistic human.

Acknowledgement

We thank ETH Computer Vision and Geometry group for providing the materials and hardware for this project. We also specially thank our supervisor, Dr. Sergey Prokudin for proposing this interesting topic and offering us guidance and additional device for our project.

References

[1] Introduction to barracuda: Barracuda: 1.0.4. 5

[2] Xr interaction toolkit: Xr interaction toolkit: 0.9.4-preview. 5

[3] Tomas Akenine-Mo, Eric Haines, Naty Hoffman, et al. Real-time rendering. 2018. 1

[4] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16*, pages 696–712. Springer, 2020. 3

[5] Thiemo Alldieck, Marcus Magnor, Weipeng Xu, Christian Theobalt, and Gerard Pons-Moll. Detailed human avatars from monocular video. In *2018 International Conference on 3D Vision (3DV)*, pages 98–109. IEEE, 2018. 1

[6] Mikołaj Bińkowski, Danica J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying mmd gans, 2021. 6, 8

[7] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019. 3

[8] Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1511–1520, 2017. 2, 4

[9] Tian Qi Chen and Mark Schmidt. Fast patch-based style transfer of arbitrary style. *arXiv preprint arXiv:1612.04337*, 2016. 2

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 4, 7

[11] Leon A Gatys, Alexander S Ecker, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Controlling perceptual factors in neural style transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3985–3993, 2017. 2

[12] Tom Geller. Overcoming the uncanny valley. *IEEE computer graphics and applications*, 28(4):11–17, 2008. 1

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 2

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Y. Bengio. Generative adversarial networks. *Advances in Neural Information Processing Systems*, 3, 06 2014. 4

[15] John K Haas. A history of the unity game engine. 2014. 5

[16] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017. 6, 8

[17] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *International conference on machine learning*, pages 1989–1998. PMLR, 2018. 2

[18] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017. 2

[19] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017. 2

[20] Younghyun Jo, Sejong Yang, and Seon Joo Kim. Investigating loss functions for extreme super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020. 4

[21] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016. 4, 6

[22] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3907–3916, 2018. 2

[23] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014. 6

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. 4

[25] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. *arXiv preprint arXiv:1705.08086*, 2017. 2

[26] Lingjie Liu, Marc Habermann, Viktor Rudnev, Kripasindhu Sarkar, Jiatao Gu, and Christian Theobalt. Neural actor: Neural free-view synthesis of human actors with pose control. *arXiv preprint arXiv:2106.02019*, 2021. 3

[27] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7708–7717, 2019. 2

[28] Stephen Lombardi, Jason Saragih, Tomas Simon, and Yaser Sheikh. Deep appearance models for face rendering. *ACM Transactions on Graphics (TOG)*, 37(4):1–13, 2018. 2

[29] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J Black. Smpl: A skinned multi-person linear model. *ACM transactions on graphics (TOG)*, 34(6):1–16, 2015. 2

[30] Ricardo Martin-Brualla, Rohit Pandey, Shuoran Yang, Pavel Pidlypenskyi, Jonathan Taylor, Julien Valentin, Sameh Khamis, Philip Davidson, Anastasia Tkach, Peter Lincoln, et al. Lookingood: Enhancing performance capture with real-time neural re-rendering. *arXiv preprint arXiv:1811.05029*, 2018. 1, 2, 4, 8

[31] Oscar Meruvia-Pastor. Enhancing 3d capture with multiple depth camera systems: A state-of-the-art report. In *RGB-D Image Analysis and Processing*, pages 145–166. Springer, 2019. 1

[32] 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 *European conference on computer vision*, pages 405–421. Springer, 2020. 3

[33] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Transactions on Graphics (TOG)*, 38(6):1–17, 2019. 1

[34] Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed A. A. Osman, Dimitrios Tzionas, and Michael J. Black. Expressive body capture: 3D hands, face, and body from a single image. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 10975–10985, 2019. 3

[35] 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. 6

[36] Sergey Prokudin, Michael J Black, and Javier Romero. Smplpix: Neural avatars from 3d human models. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1810–1819, 2021. 2, 3, 4, 6, 8

[37] Elad Richardson, Yuval Alaluf, Or Patashnik, Yotam Nitzan, Yaniv Azar, Stav Shapiro, and Daniel Cohen-Or. Encoding in style: a stylegan encoder for image-to-image translation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021. 4

[38] Stephan R Richter, Hassan Abu AlHaija, and Vladlen Koltun. Enhancing photorealism enhancement. *arXiv preprint arXiv:2105.04619*, 2021. 2

[39] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016. 6, 7, 8

[40] Aliaksandra Shysheya, Egor Zakharov, Kara-Ali Aliev, Renat Bashirov, Egor Burkov, Karim Iskakov, Aleksei Ivakhnenko, Yury Malkov, Igor Pasechnik, Dmitry Ulyanov, et al. Textured neural avatars. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2387–2397, 2019. 2

[41] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. 7

[42] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, et al. State of the art on neural rendering. In *Computer Graphics Forum*, volume 39, pages 701–727. Wiley Online Library, 2020. 1

[43] Ayush Tewari, Justus Thies, Ben Mildenhall, Pratul Srinivasan, Edgar Tretschk, Yifan Wang, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi, Tomas Simon, Christian Theobalt, Matthias Niessner, Jonathan T. Barron, Gordon Wetzstein, Michael Zollhoefer, and Vladislav Golyanik. Advances in neural rendering, 2021. 1

[44] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, volume 1, page 4, 2016. 2

[45] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 2

[46] Delio Vicini, David Adler, Jan Novák, Fabrice Rousselle, and Brent Burley. Denoising deep monte carlo renderings. *Computer Graphics Forum*, 38(1):316–327, Feb. 2019. 2

[47] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018. 2, 4, 6

[48] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8798–8807, 2018. 2

[49] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. 6, 8

[50] Shih-En Wei, Jason Saragih, Tomas Simon, Adam W Harley, Stephen Lombardi, Michal Perdoch, Alexander Hypes, Dawei Wang, Hernan Badino, and Yaser Sheikh. Vr facial animation via multiview image translation. *ACM Transactions on Graphics (TOG)*, 38(4):1–16, 2019. 2

[51] Tim Weyrich, Jason Lawrence, Hendrik PA Lensch, Szymon Rusinkiewicz, and Todd Zickler. *Principles of appearance acquisition and representation*. Now Publishers Inc, 2009. 1

[52] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. 4, 6, 8

[53] Xianyao Zhang, Marco Manzi, Thijs Vogels, Henrik Dahlberg, Markus Gross, and Marios Papas. Deep compositional denoising for high-quality monte carlo rendering. In *Computer Graphics Forum*, volume 40, pages 1–13. Wiley Online Library, 2021. 2