SqueezeMe: Efficient Gaussian Avatars for VR

SHUNSUKE SAITO, Meta Reality Labs, USA
STANISLAV PIDHORSKYI, Meta Reality Labs, USA
IGOR SANTESTEBAN, Meta Reality Labs, USA
FORREST IANDOLA, Meta Reality Labs, USA
DIVAM GUPTA, Meta Reality Labs, USA
ANUJ PAHUJA, Meta Reality Labs, USA
NEMANJA BARTOLOVIC, Meta Reality Labs, Switzerland
FRANK YU, Meta Reality Labs, USA
EMANUEL GARBIN, Meta Reality Labs, Israel
TOMAS SIMON, Meta Reality Labs, USA

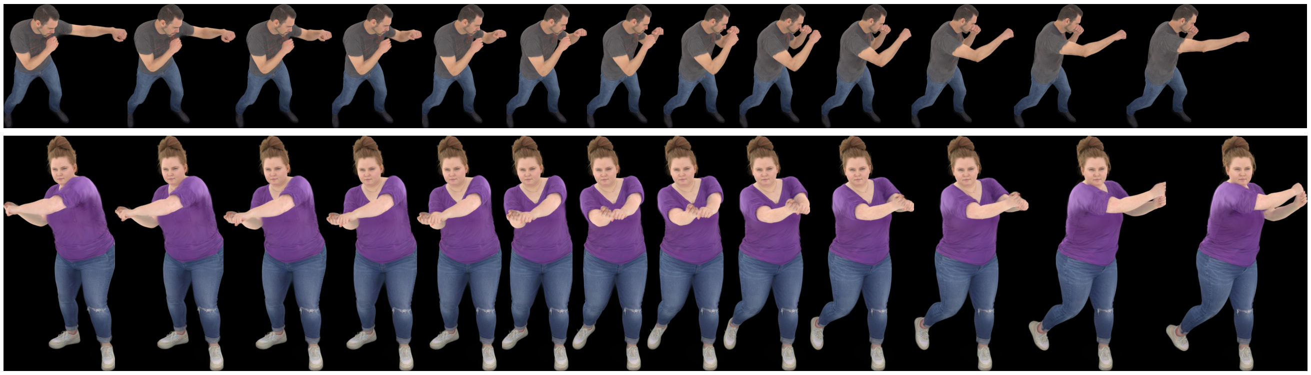

Fig. 1. **With SqueezeMe, we simultaneously run 3 Gaussian avatars locally on a Meta Quest 3 VR headset.** This figure shows frame sequences of two SqueezeMe avatars. Top: Boxing. Bottom: Miming hitting a baseball. Later in the paper, we refer to this model as "4k correctives, linearized." Each avatar has approximately 60,000 Gaussian Splats and is drivable from video input.

Gaussian Splatting has enabled real-time 3D human avatars with unprecedented levels of visual quality. While previous methods require a desktop GPU for real-time inference of a single avatar, we aim to squeeze multiple Gaussian avatars onto a portable virtual reality headset with real-time drivable inference. We begin by training a previous work, Animatable Gaussians, on a high quality dataset captured with 512 cameras. The Gaussians are animated by controlling base set of Gaussians with linear blend skinning (LBS) motion and then further adjusting the Gaussians with a neural network decoder to correct their appearance. When deploying the model on a Meta Quest 3 VR headset, we find two major computational bottlenecks: the decoder and the rendering. To accelerate the decoder, we train the Gaussians in UV-space instead of pixel-space, and we distill the decoder to a single neural network layer. Further, we discover that neighborhoods of Gaussians can share a single corrective from the decoder, which provides an additional speedup. To accelerate the rendering, we develop a custom pipeline in Vulkan that runs on the mobile GPU. Putting it all together, we run 3 Gaussian avatars concurrently at 72 FPS on a VR headset. Demo videos are at forresti.github.io/squeezeme.

1 Introduction

Since the inception of Virtual Reality (VR), a fundamental goal has been to faithfully simulate the real world in an immersive environment [22]. To simulate human interactions, we need realistic avatars that mimic the appearance of real people. This would enable people to be themselves in VR, happily and productively interacting with other people in meetings, games, adventures, and shared experiences. For the best experience, avatars must be drivable from camera input data and runnable in real-time. To avoid high cloud-computation costs and internet latencies, it would be ideal run the avatar inference and visualization locally on VR headsets.

For graphical simulation of scenes and objects, three common approaches are mesh, NeRF [12], and Gaussian Splatting (GS) [4]. Meshes, with textures and materials overlayed, are cheap to render, but they are difficult to train from data. Further, representing fine details such as hair can be difficult [21]. NeRF offers higher quality than mesh, and NeRF is easier to train from data, but rendering NeRF is prohibitively slow (even with optimizations such as Instant-NGP [14]) on a VR headset. 3D Gaussian Splatting [4] has emerged as a particularly effective approach for graphical simulation of human avatars. GS offers dramatic improvements in the quality

.

of human hair and clothing visualization, compared to previous mesh-based methods [20]. Further, recent work such as Animatable Gaussians has shown that human avatars can be "driven" by using a decoder and/or linear blend skinning to update the avatar's shape and appearance [6] (see Section 2.2 for details). And, while Animatable Gaussians method runs at 10 FPS on a desktop GPU, we must dramatically improve the algorithm's efficiency before it can run in real-time on the modest computational budget of a Meta Quest 3 VR headset.

In this work, our goal is to redesign a drivable Gaussian splatting avatar algorithm – specifically Animatable Gaussians [6] – to be fast enough to run multiple avatars locally on a VR headset, while preserving nearly all the quality of the baseline model.

2 Related Work

Since the introduction of Gaussian Splatting (GS) [4], the community has developed solutions for animating people, objects, and scenes with GS. For instance, Luiten et al show that Gaussians can be extended into the time domain, enabling 4D video animation [9]. Li et al introduced Animatable Gaussians, which enables driving Gaussian avatars to mimic input videos [6]. This allows a user to move in front of a webcam and control a Gaussian-enabled character in a video game or virtual reality environment. Several other works also animate human figures with Gaussians [3, 13, 17, 21, 26]. To reduce the computational costs associated with visualizing Gaussians, Svitov et al propose HAHA, which represents drivable avatars with a hybrid of Gaussians and mesh [23]. Next, we review Gaussian Splatting and Animatable Gaussians in more detail.

2.1 Gaussian Splatting

3D Gaussian Splatting [4] is a powerful representation for computer graphics. Each Gaussian splat is parameterized by several terms including rotation, translation μ, scale σ, and a set of spherical harmonics terms for view-dependent color. The splat's rotation and scale construct a covariance matrix Σ. Each splat also has a density term δ, which defines the opacity at the center of the splat.

Let us view the splats from a specific camera c, defined with focal length (f_x, f_y), translation $t_c = (x_c, y_c, z_c)$, and rotation matrix R_c. The Jacobian of this camera is

$$J = \begin{bmatrix} \frac{f_x}{z_c} & 0 & -\frac{f_x x_c}{z_c^2} \\ 0 & \frac{f_y}{z_c} & -\frac{f_y y_c}{z_c^2} \end{bmatrix}$$

From the camera perspective, the 2D covariance of the splat is $\Sigma_{proj} = J R_c \Sigma R_c^T J^T$. Let μ_c represent the translation of a splat in pixel space.

After sorting the splats based on their depth and their visibility to each pixel, the splats are rasterized onto an image as follows. The opacity α of a splat k to a pixel located at $p = (p_x, p_y)$ units from the image center is

$$\alpha_k = \delta\ exp(-\frac{1}{2}(\mu_c - p)^T \Sigma_{proj}^{-1}(\mu_c - p))$$

The final color of pixel p is computed as

$$C = \sum_{i=1}^{N} \alpha_i \prod_{j=1}^{i-1}(1 - \alpha_j) c_i$$

where N is the number of splats visible to pixel p, and the view-dependent color c_i is computed using spherical harmonics.

2.2 Animatable Gaussians

We now summarize the Animatable Gaussians [6] method for representing human motion with Gaussian Splatting. Let G_i represent the initial "base" Gaussians of a human body before animation. Let G_f represent the final Gaussians that are posed to form one frame of animation. Let p be the desired pose of the avatar for one frame of animation. Let LBS be linear blend skinning [11]. Let C be a neural network that computes correctives. For each frame, the animated Gaussian is computed as:

$$G_f = LBS(G_i + C(p), p) \tag{1}$$

The intuition is that G_i is the set of Gaussians in a canonical pose (typically an A-pose with legs together and the arms raised slightly), and the large motions are produced using LBS. And, the neural network C captures more subtle effects such as the stretching and wrinkling of skin and fabric.

C produces a grid of 1024x1024 Gaussian correctives in pixel-space. While C produces approximately 1 million correctives, there are only 300,000 Gaussians, so a binary mask is used to remove approximately two-thirds of the correctives produced by C. For inference, we find this is highly inefficient; it takes 50 ms (20 FPS) to run a quantized version of model C on the Hexagon Tensor Processor (HTP) on a Meta Quest 3 VR headset.

2.3 VR Hardware

Our target platform is the Meta Quest 3 virtual reality headset. It uses around 10 Watts of power, which is far less than an NVIDIA H100 GPU's 700 Watts. The computational system-on-chip (SOC) in Quest 3 is the Qualcomm XR2G2 [19]. Within XR2G2 are multiple processing devices, notably a cluster of ARM CPUs, the Hexagon Tensor Processor (HTP), and the Adreno GPU [18]. The HTP has two engines: the matrix engine, which has fixed-function support to efficiently compute convolutions and linear layers, and the vector engine, which can be programmed more flexibly for other operations such as activation functions, transposes, and image scaling. The HTP is optimized for integer math with 8-bit weights, and 8-bit or 16-bit activations, therefore we need to quantize our models to run on the HTP. In our final implementation, we will run the corrective model C on the HTP, and we will render[1] the them on the GPU.

In the mobile GPU implementation that we will describe in Section 3.5, we can visualize approximately 180,000 Gaussians per frame, at a frame rate of 72 frames per second (FPS), which is the refresh rate of the VR display. Our goal is to run many avatars in parallel at 72 FPS, so we will reduce the number of Gaussians and also use a

[1] In this paper, we use the terms "visualize" and "render" interchangeably. What we are referring to is the process of projecting Gaussian splats into image-space and rasterizing them.

level of detail system to further reduce the Gaussians on avatars on far-away avatars.

2.4 Codec Avatars

The goal of Codec Avatars (CA) is to produce realistic 3D human avatars that are drivable in real-time and can be used in VR environments. CA can be driven from full-body camera(s) directed at a human. CA can also be driven from cameras mounted on a VR headset [1, 2]. In CA, there is a notion of a *sender* and a *receiver.* A sender provides live updates of their face and body pose to other users. A receiver takes others' pose and visualizes them in real-time in a VR environment. Typically, every user is both a sender and a receiver. In a video game or large meeting, a receiver may need to visualize dozens of avatars at once.

Multiple representations of Codec Avatars have been developed, including tracked mesh [10], volumetric primitives [7], and Gaussian Splatting [20]. For the present work, we focus full-body Codec Avatars based on Gaussian Splatting. For our experiments, we drive the avatars from full-body cameras, though in the future our work can be extended to drive the avatars from VR head-mounted cameras.

3 Methodology

3.1 UV-space Animatable Gaussians

We now aim to produce quality similar to Animatable Gaussians, with fewer Gaussians. Our goal is to reduce the number of Gaussians from 300,000 (too slow to run even one avatar at 72 FPS), to 60,000 or less. If there are 60,000 Gaussians per avatar, we could visualize 3 avatars in parallel.

In our preliminary experiments, we tried simply training with 5× fewer Gaussians (reducing from 300k Gaussians to 60k Gaussians), but we found that it degrades the avatar quality dramatically. In Animatable Gaussians, we observe two areas where Gaussians may be wasted. First, there are separate sets of Gaussians for the front of the human body and the back of the body. Second, the Gaussians are not initialized to align with the body. To address both of these inefficiencies, we use the training images to reconstruct the avatars and fit a mesh that can be controlled with LBS, similar to SMPL [8]. The mesh includes a predetermined UV mapping that we use for all identities. Then, before we begin Gaussian splat training, we initialize one splat at each pixel location in UV-space. We refer to our approach as "UV-space Animatable Gaussians," and in Table 1 we observe it produces results comparable or superior quality to the original Animatable Gaussians with 5× fewer Gaussians.

Compared to the original Animatable Gaussians model, an other change we make is to reformulate the corrective computation with an encoder-decoder network. So, we replace the corrective computation network C with an encoder E and a decoder D, so the corrective computation. The input to the encoder is a set of face keypoints and a set of body keypoints. The encoder has two parts: a face encoder, and a body encoder. The face encoder and the body encoder each consist of two linear layers, and they each output a 32d vector. We concatenate the output of the face encoder and the body encoder into one 64d vector, which is the input to the decoder. The network architecture of D is a stack of convolution, hardswish, and bilinear upsampling. We illustrate the model architecture in Figure 2. The output of D is a 256x256x37 grid of correctives, each element of the 256x256 is one Gaussian corrective, and each Gaussian corrective has 37 values, of which 27 values represent the spherical harmonics[2], and the remaining 10 values represent correctives for the rotation, translation, and scale.

While our UV-space of size 256x256 gives an upper bound of 65536 Gaussians, there are some empty regions of the UV-mapping, so there are only 60381 Gaussians. To discard unnecessary correctives, we apply a binary mask M to the 65536 correctives to reduce it to 60381.

For the UV-space Gaussian method, we now show how to compute the Gaussians for one frame of animation:

$$G_f = LBS(G_i + M(D(E(p))), p) \tag{2}$$

For compactness in the previous equation, describe the computation of final Gaussians as a sum of the base Gaussians G_i and the correctives $D(E(p))$. But, in reality, there is no corrective for δ, and there are no base Gaussian attributes in G_i for the 1- and 2-degree spherical harmonics.

Our loss function is a weighted-sum of several terms: $\mathcal{L}_{photo}$ and $\mathcal{L}_{lpips}$ are the photometric (L1) and LPIPS [28] distance between the ground-truth and predicted image. $\mathcal{L}_{kpt}$ is the L1 loss, and the body pose keypoints. To reduce "runaway" Gaussians that drift far from their initial position, $\mathcal{L}_{offset}$ is an L1 loss that is maximized when the mean μ of each Gaussian in G_f is unchanged from μ where the Gaussian was initialized at the start of training. To reduce unwanted holes in the avatar, we introduce $\mathcal{L}_{\alpha}$, which is an L1 loss that is maximized when the α-transparency map generated by the Gaussian splat model matches the Sapiens [5] segmentation mask; the intuition is that the avatar should be opaque and background pixels should be transparent. When rendering Gaussian splats on a mobile, it is cheaper if each Gaussian is visible from fewer pixels [16]. To reduce the number of pixels to which Gaussian is applied during rendering, we adopt the loss $\mathcal{L}_{\sigma}$, which uses L1 to minimize the average size of the splats, and we adopt $\mathcal{L}_{opacity}$, which uses L1 to minimize the average opacity of the splats. The weights for the sum are set to $\lambda_{photo} = 1$, $\lambda_{lpips} = 0.1$, $\lambda_{opacity} = 0.01$, $\lambda_{scale} = 1$, $\lambda_{offset} = 1$, $\lambda_{alpha} = 0.1$, and $\lambda_{keypoint} = 0.1$. The loss is computed as

$$\begin{aligned}\mathcal{L} = &\lambda_{photo}\mathcal{L}_{photo} + \lambda_{lpips}\mathcal{L}_{lpips} + \lambda_{\alpha}\mathcal{L}_{\alpha} + \lambda_{kpt}\mathcal{L}_{kpt} \\ &+ \lambda_{opacity}\mathcal{L}_{opacity} + \lambda_{scale}\mathcal{L}_{scale} + \lambda_{offset}\mathcal{L}_{offset}\end{aligned}$$

3.2 Linear distillation

We now turn our attention distilling the encoder E and decoder D into a single linear layer using principal component analysis (PCA). For short, we refer to this process as *linearization.* To distill a decoder for one identity, we begin by collecting a dataset of inputs and outputs to the encoder-decoder. In this section, we continue to use the face-encoder, but we replace the body-encoder with LBS data. So, our input space is a set of poses, and the output space is the output of the decoder from Equation 2.

[2] For all experiments on UV-space Gaussians, we use 2 degrees of spherical harmonics.

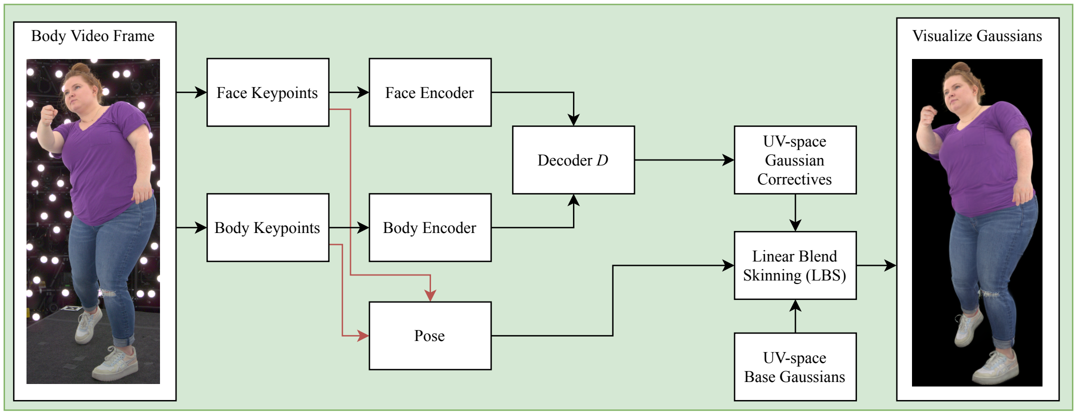

Fig. 2. **System Diagram during training.** This is the configuration we use for training the model in Section 3.1.

For one frame, the linearized decoder's input is the concatenation of a 280d vector of LBS poses and a 32d vector of facial keypoint embeddings. The decoder's output is a 60381x37 tensor. We use PCA to compress the input vector to a total of 32d. As an additional optimization, we do a second stage of PCA to compress the spherical harmonics from 27 values to 6 values. We invert the dataset matrix and use least-squares to compute a linear layer that maps the input to 60381x16 output.

For a given pose p, the output of the distilled decoder $D_L(p)$ is computed as:

$$D_L(p) = C \cdot B_c \tag{3}$$

Where:

- C (Pose Codes) the PCA-compressed representation of the pose, calculated as $(p - \bar{p}) \cdot B_p$. The pose basis matrix B_p is obtained by performing PCA on the centered poses $(p - \bar{p})$ and retaining d principal components. We add a bias term to get $C = \begin{bmatrix} 1 & (p - \bar{p}) \cdot B_p \end{bmatrix}$.
- B_c (Correctives Basis Matrix) is the analytical solution to the least squares problem, calculated using the normal equations $B_c = (C^T C)^{-1} C^T \cdot M(D(E(p)))$, where $M(D(E(p)))$ are the masked correctives.

The distilled decoder is a 2-layer model. The first layer is a linear layer that takes a 32d compressed vector and outputs a 60381x16 vector, and the layer has 32x60381x16 parameters. The second layer takes 6 of the 16 values from the first layer and expands them into 27 spherical harmonics, and the layer has 6x27 parameters. By far, the dominant cost is the first layer, which we find takes 4 ms to run (with quantization) on the Hexagon Tensor Processor (HTP) of the XR2G2 chip in the Meta Quest 3 VR headset.

With D_L representing the distilled decoder, we compute the Gaussians for one frame of animation as follows.

$$G_f = LBS(G_i + D_L(p), p) \tag{4}$$

Our linear distillation draws inspiration from Gaussian Eigen Models [29], which distills a decoder linear for head-only human animation to a single linear layer. One difference between our work and [29] is that we do full-body instead of head-only avatars. An additional difference is that we do an additional stage of PCA to compress the spherical harmonics space from 27 dimensions to 6, further reducing the computational cost of our linear decoder.

3.3 Gaussian Corrective Sharing

In the human body, nearby particles of skin, hair, and clothing move together. So far in this paper, we have produced one corrective for each Gaussian, which allows Gaussians to move independently. If we can leverage the correlation among nearby Gaussians to reduce the number of correctives, this will reduce the number of outputs in the decoder, yielding a speedup. In other words, we wish to decouple the number of Gaussians from the number of correctives, and we want to reduce the number of correctives.

While the decoder from Equation 2 produces 256x256x37 output, we modify the decoder to produce a smaller 64x64x37 output. For notation, we call the modified decoder D_{GCS}, where GCS is short for Gaussian corrective sharing. This reduces the number of correctives from 65536 to 4096. During training, we take the output of D_{GCS} and use `Nearest` interpolation to upscale it from 64x64 to 256x256 = 65536 correctives. This has the effect of "sharing" corrective for each 4x4 neighborhood of Gaussians in UV-space. For notation, we abbreviate `Nearest` upscaling as Up. And, similar to Equation 2, we use a mask M to reduce the number of correctives from 65536 to 60381. During training, we compute the Gaussians for one frame of animation as

$$G_f = LBS(G_i + M(Up(D_{GCS}(E(p)))), p) \tag{5}$$

3.4 Combining Linear Distillation and Corrective Sharing

We now combine linear distillation and Gaussian corrective sharing. The key idea is to distill D_{GCS} without upsampling, so the linearized model only needs to produce 4096 correctives instead of 60381 correctives. But, how to map a flat array of 4096 correctives to a flat array of 60381 Gaussians? We address this by building a lookup table as follows. Let A be an array of size 64x64, where the values of A go from 0 to 4095 in contiguous order. Let Up represent `Nearest` upscaling. And, M is the binary mask that removes unused

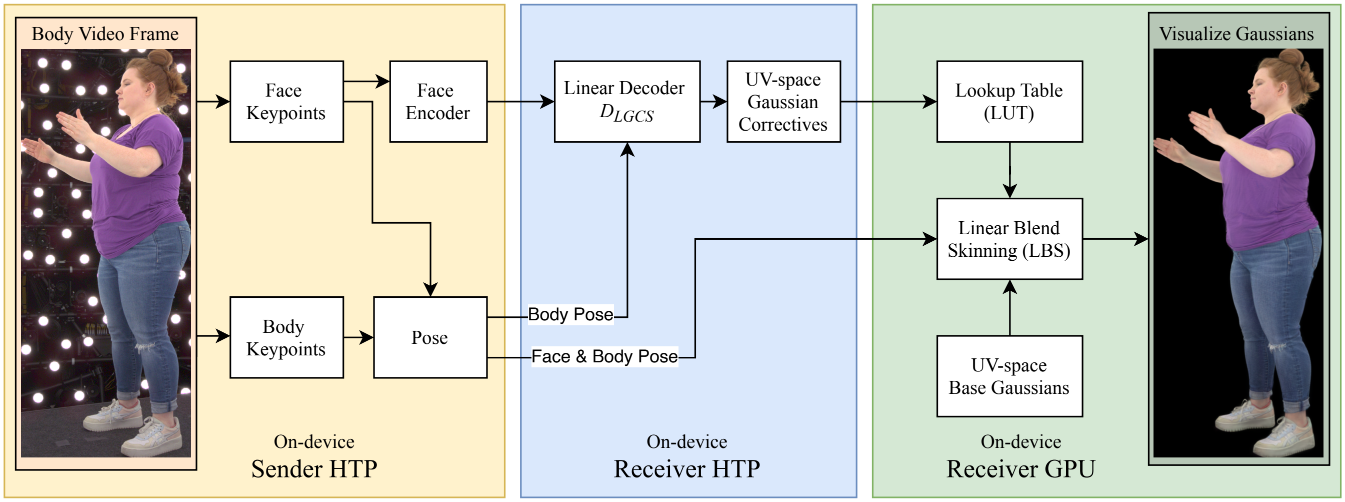

Fig. 3. **System Diagram with optimizations.** Here, we show the end-to-end optimized system, including the techniques from Sections 3.1–3.4. For details on HTP and GPU, see Section 2.3. For details on the sender and receiver, see Section 2.4.

correctives from a 256x256 grid. We define the lookup table LUT as

$$LUT = M(Up(A)) \tag{6}$$

where LUT is a vector of length 60381. For any input to LUT in the range of [0, 60380], returns the appropriate corrective sharing index, in the range of [0, 4095].

We apply linear distillation to the corrective sharing decoder to produce a new decoder D_{LGCS}. Let x be the index of one Gaussian in the range of [0, 60380]. For one frame of animation, we compute Gaussian x as

$$Corr = D_{LGCS}(p) \tag{7}$$

$$G_f[x] = LBS(G_i[x] + Corr[LUT[x]], p) \tag{8}$$

When using corrective sharing with 4096 correctives, the latency to compute the quantized decoder D_{LGCS} takes just 0.45 ms on the Quest 3 HTP. We illustrate the end-to-end system with D_{LGCS} and on-device inference in Figure 3.

3.5 Vulkan Visualizer

While the Qualcomm system-on-chip in the Meta Quest 3 does not support CUDA, it does support Vulkan. With Vulkan, we can program both compute shaders (which use the programmable portions of the mobile GPU) and graphics shaders (which use the fixed-function hardware rasterizer). The key steps in Gaussian Splatting visualization are: projecting 3D Gaussians camera space; applying spherical harmonics to determine the color of each Gaussian; sorting Gaussians based on their depth; and rasterization. We implement a compute shader that takes a camera angle and performs the projection and calculates the color of each Gaussian. Inspired by [15, 27], the compute shader outputs a set of quads, where each quad has the size, color, and opacity-function of one gaussian. To sort and rasterize the quads into an image, we implement graphics shaders that take advantage of the hardware support for depth-sorting and rasterization.

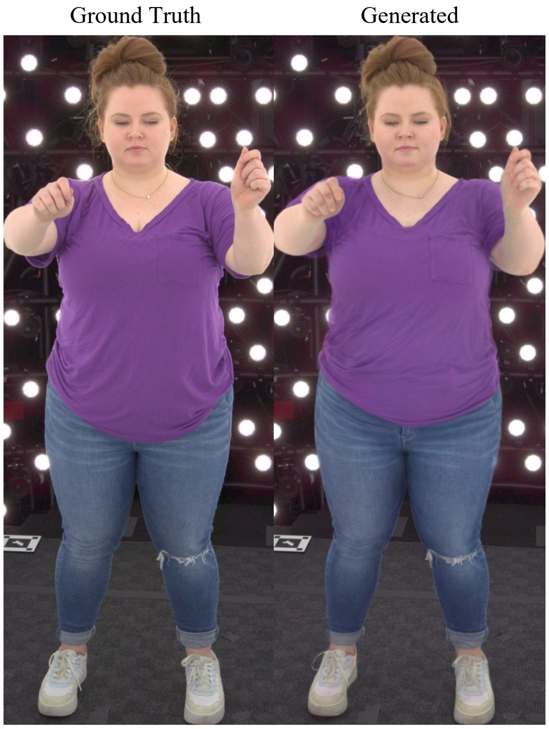

Fig. 4. This is what the images look like when we evaluate them. Left: ground-truth image. Right: generated image, with the avatar rendered in front of a static image of the studio. Both images are cropped to a bounding-box of a human segmentation mask.

Table 1. **Main results.** We evaluate the impact of linearization and number of correctives on quality. Results are at the original image resolution, with images cropped to rectangles based on segmentation of the avatar. Results are averaged over 4 identities. Latency is for the decoder only.

Model	# Gaussians	# Correctives	Linearized	L1 ↓	LPIPS ↓	PSNR ↑	SSIM ↑	Decoder Latency on Quest 3
Animatable Gaussians [6]	300k	300k		0.043	0.158	23.349	0.602	
SqueezeMe	60k	60k		0.037	0.145	24.744	0.625	
SqueezeMe	60k	4k		0.037	0.147	24.778	0.623	
SqueezeMe	60k	60k	✓	0.040	0.150	24.159	0.618	5.0 ms
SqueezeMe	60k	4k	✓	0.040	0.151	24.134	0.615	0.45 ms
SqueezeMe (no decoder)	60k	0		0.041	0.165	23.632	0.607	0

4 Experimental Setup

We train and evaluate on data that was collected on an internal capture dome with the users' permission to use in published research. Our capture dome is a 3 meter diameter dome with 512, 25 megapixel cameras streaming at 90 FPS. The dome also has 1024 individually controllable lights.

For evaluating the results, we render the avatars in front of an image of the multi-camera, multi-light studio environment where the ground-truth data was captured, and we show examples of ground-truth and generated images in Figure 4. A significant portion of the image is background, so we crop the rendered images to the maximum width and height of Sapiens [5] segmentation masks on the ground-truth data. It is important to note that without this cropping, all models would benefit from artificially inflated accuracy scores due to the large background regions that do not contain the avatar. We use LPIPS [28], L1, PSNR, and SSIM [25] to evaluate the cropped images against ground-truth.

We choose Animatable Gaussians as a baseline for the following reasons. Works such as [26] focus on the human head, while our goal is to animate the full body. 3DGS Avatar [17] and Animatable Gaussians [6] both develop Gaussian full-body avatars, but Animatable Gaussians produces the biggest gains over the non-Gaussian baseline of ARAH [24]. Further, while other works train Gaussian avatars from multi-camera datasets, GaussianAvatar [3] and HAHA [23] each train on a single-camera video sequence, which is quite impressive, but limits the quality of the final avatar. To the best of our knowledge, Animatable Gaussians is the leading method for Gaussian full-body animation, particularly when a multi-camera training dataset is available.

5 Results

In Table 1, we compare several versions of our method with the baseline of Animatable Gaussians. In all of our evaluations, we are performing novel-view synthesis, i.e. we are using camera positions that were not in training set. All results are averaged across 4 identities and 5 cameras per identity. The test set is held-out from the dataset that is used for training, linearization, and quantization. In Table 1, we observe that non-linearized models with 65k or 4k correctives produce virtually the same results, with 65k narrowly winning on LPIPS and SSIM; 4k winning on PSNR, and a tie on L1. Thus, corrective sharing with 4k correctives does not seem degrade quality according to these numerical measurements. Further, we observe a modest drop in numerical quality when linearizing the 65k and 4k models. But, the linearized models still produce higher quality than a baseline trained with no decoder.

We evaluated the linearized models with and without quantization, and we got identical results. In other words, quantizing with 8-bit weights and 16-bit activations did not harm the L1, LPIPS, PSNR, and SSIM. Therefore, the "Linearized" results in Table 1 are for both quantized and unquantized models.

We now consider the qualitative results. In Figure 5, we show representative examples of the different models across different identities, poses, and camera views. We observe that using corrective-sharing to reduce the number of correctives from 65k to 4k and linearizing the model produce very little degradation in the avatar quality. However, by reducing the number of correctives and linearizing the model, we are able to squeeze the decoder onto a VR headset with just 0.45 ms of latency per inference. When running at 72 FPS, and with half of the VR headset's Hexagon Tensor Processor (HTP) core dedicated to the decoder, this allows to decode 15 avatars in parallel.

However, there are some problems that arise with corrective-sharing and linearization. For instance, in Figure 6(b, e) we find that corrective-sharing and linearization can both cause degradation at on the arms, particularly at the armpit and the point where a t-shirt sleeve meets the skin. Further, in Figure 6(a), we observe that for certain identities and certain poses, all the models struggle with blurriness on the hands. These problems may be resolved in the future with more adaptive methods of distributing Gaussians and correctives across a human avatar.

5.1 End-to-End Results

Our analysis has focused on improving the latency of the decoder, which provides correctives to the Gaussians in every frame. Now that we have improved the decoder latency, and we have implemented an efficient visualizer in Vulkan, we can run 3 avatars at full resolution concurrently on a Meta Quest 3 VR headset at 72 FPS. The optimized linear decoder with 4k correctives uses minimal computation on the HTP, but the mobile GPU visualization limits latency. In the future, a level of detail (LOD) system could allow more avatars to be decoded and visualized on a VR headset.

6 Conclusions

Gaussian Splatting based avatars offer high quality, but until the present work they were too slow to run locally on a VR headset. We propose several techniques to improve the efficiency of Gaussian

Splatting in animatable human avatars, including UV-space Gaussians, linear distillation and corrective-sharing. This improves the latency of the Gaussian corrective decoder from a baseline of 50 ms to just 0.45 ms. Further, we can run 3 avatars at 72 frames per second on a Quest 3 VR headset. We believe this work will usher in a new phase of lifelike human interactions in virtual reality.

References

[1] Shaojie Bai, Te-Li Wang, Chenghui Li, Akshay Venkatesh, Tomas Simon, Chen Cao, Gabriel Schwartz, Jason Saragih, Yaser Sheikh, and Shih-En Wei. Universal facial encoding of codec avatars from vr headsets. *ACM Trans. Graph.*, 43(4), 2024.

[2] Hang Chu, Shugao Ma, Fernando De la Torre, Sanja Fidler, and Yaser Sheikh. Expressive telepresence via modular codec avatars. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XII 16*, pages 330–345. Springer, 2020.

[3] Liangxiao Hu, Hongwen Zhang, Yuxiang Zhang, Boyao Zhou, Boning Liu, Shengping Zhang, and Liqiang Nie. GaussianAvatar: Towards realistic human avatar modeling from a single video via animatable 3d gaussians. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[4] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.*, 42(4), 2023.

[5] Rawal Khirodkar, Timur Bagautdinov, Julieta Martinez, Su Zhaoen, Austin James, Peter Selednik, Stuart Anderson, and Shunsuke Saito. Sapiens: Foundation for human vision models. In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part IV*, page 206–228, Berlin, Heidelberg, 2024. Springer-Verlag.

[6] Zhe Li, Zerong Zheng, Lizhen Wang, and Yebin Liu. Animatable Gaussians: Learning pose-dependent gaussian maps for high-fidelity human avatar modeling. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19711–19722, 2024.

[7] Stephen Lombardi, Tomas Simon, Gabriel Schwartz, Michael Zollhoefer, Yaser Sheikh, and Jason Saragih. Mixture of volumetric primitives for efficient neural rendering. *ACM Trans. Graph.*, 40(4), 2021.

[8] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16, 2015.

[9] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3D Gaussians: Tracking by Persistent Dynamic View Synthesis . In *2024 International Conference on 3D Vision (3DV)*, pages 800–809, Los Alamitos, CA, USA, 2024. IEEE Computer Society.

[10] Shugao Ma, Tomas Simon, Jason M. Saragih, Dawei Wang, Yuecheng Li, Fernando De la Torre, and Yaser Sheikh. Pixel codec avatars. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 64–73, 2021.

[11] Thalmann Magnenat, Richard Laperrière, and Daniel Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings of Graphics Interface'88*, pages 26–33. Canadian Inf. Process. Soc, 1988.

[12] 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. *Commun. ACM*, 65(1):99–106, 2021.

[13] Arthur Moreau, Jifei Song, Helisa Dhamo, Richard Shaw, Yiren Zhou, and Eduardo Perez-Pellitero. Human Gaussian Splatting: Real-Time Rendering of Animatable Avatars . In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 788–798, Los Alamitos, CA, USA, 2024. IEEE Computer Society.

[14] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, 2022.

[15] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3D Gaussian Splatting for accelerated novel view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10349–10358, 2024.

[16] Michael Niemeyer, Fabian Manhardt, Marie-Julie Rakotosaona, Michael Oechsle, Daniel Duckworth, Rama Gosula, Keisuke Tateno, John Bates, Dominik Kaeser, and Federico Tombari. Radsplat: Radiance field-informed gaussian splatting for robust real-time rendering with 900+ fps. *arXiv:2403.13806*, 2024.

[17] Zhiyin Qian, Shaofei Wang, Marko Mihajlovic, Andreas Geiger, and Siyu Tang. 3DGS-Avatar: Animatable avatars via deformable 3d gaussian splatting. In *CVPR*, 2024.

[18] Qualcomm. Snapdragon xr2 gen 2 platform. https://www.qualcomm.com/products/mobile/snapdragon/xr-vr-ar/snapdragon-xr2-gen-2-platform, .

[19] Qualcomm. Meta quest 3. https://www.qualcomm.com/products/mobile/snapdragon/xr-vr-ar/xr-vr-ar-device-finder/meta-quest-3, .

[20] Shunsuke Saito, Gabriel Schwartz, Tomas Simon, Junxuan Li, and Giljoo Nam. Relightable gaussian codec avatars. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 130–141, 2024.

[21] Zhijing Shao, Zhaolong Wang, Zhuang Li, Duotun Wang, Xiangru Lin, Yu Zhang, Mingming Fan, and Zeyu Wang. SplattingAvatar: Realistic Real-Time Human Avatars with Mesh-Embedded Gaussian Splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[22] Ivan E Sutherland. The ultimate display. In *Proceedings of the IFIP Congress*, pages 506–508. New York, 1965.

[23] David Svitov, Pietro Morerio, Lourdes Agapito, and Alessio Del Bue. Haha: Highly articulated gaussian human avatars with textured mesh prior. *ACCV*, 2024.

[24] Shaofei Wang, Katja Schwarz, Andreas Geiger, and Siyu Tang. Arah: Animatable volume rendering of articulated human sdfs. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII*, page 1–19, Berlin, Heidelberg, 2022. Springer-Verlag.

[25] 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.

[26] Yuelang Xu, Bengwang Chen, Zhe Li, Hongwen Zhang, Lizhen Wang, Zerong Zheng, and Yebin Liu. Gaussian head avatar: Ultra high-fidelity head avatar via dynamic gaussians. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1931–1941, 2024.

[27] Zhen Xu and Zhiyuan Yu. Fast gaussian rasterization. https://github.com/dendenxu/fast-gaussian-rasterization, 2024.

[28] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018.

[29] Wojciech Zielonka, Timo Bolkart, Thabo Beeler, and Justus Thies. Gaussian Eigen Models for Human Heads. *arXiv:2407.04545*, 2024.

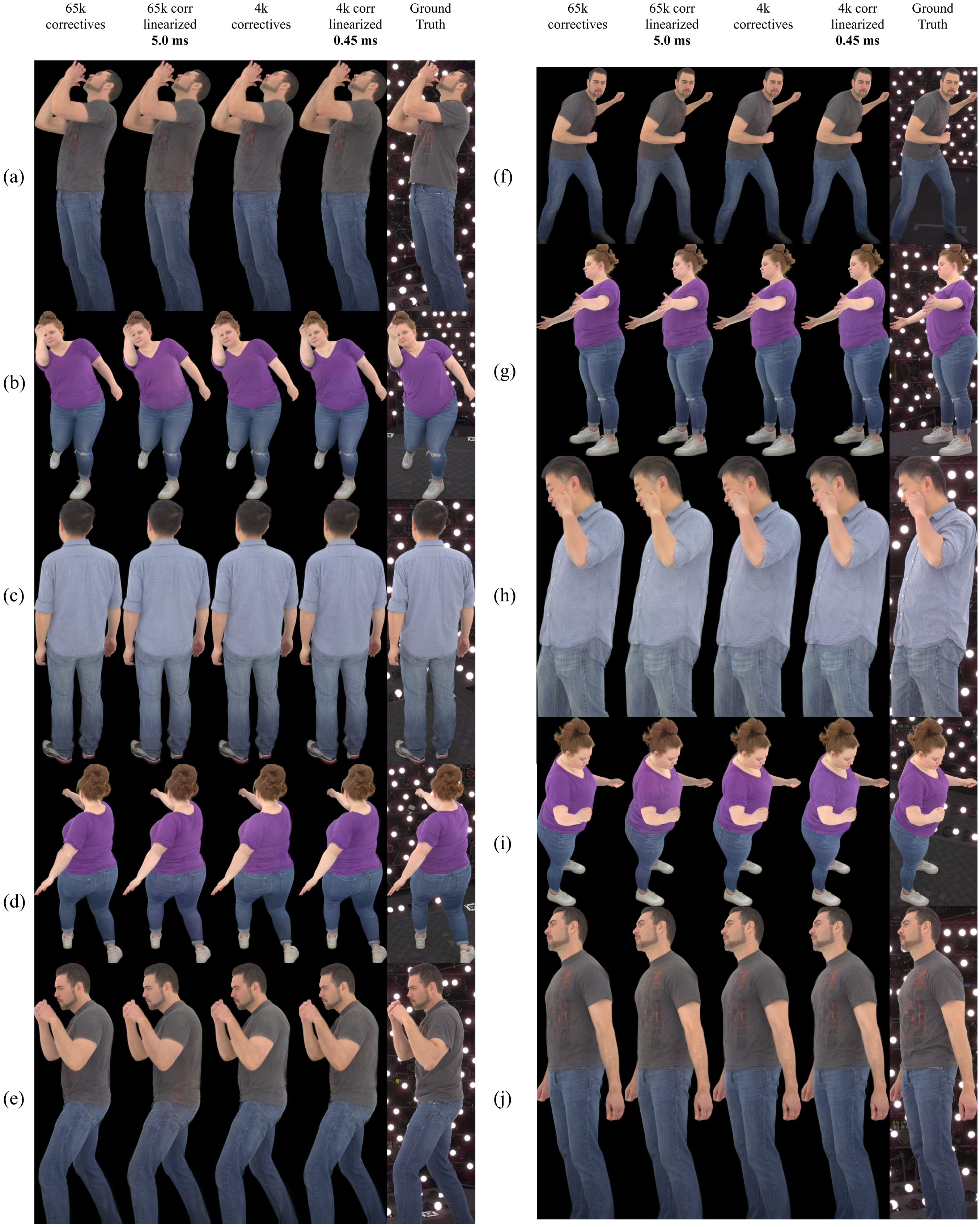

Fig. 5. **Qualitative results.** Our 0.45 ms model produces results that are competitive with far more expensive models.

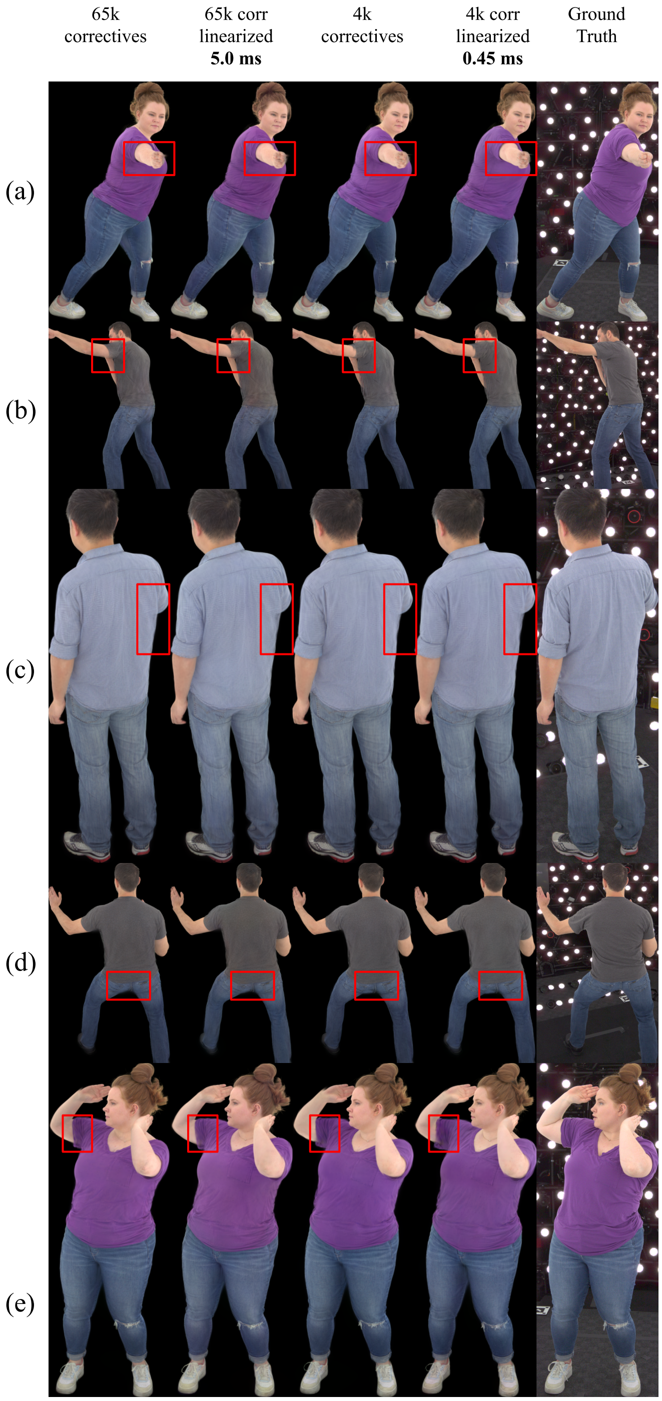
65k correctives
65k corr linearized **5.0 ms**
4k correctives
4k corr linearized **0.45 ms**
Ground Truth
(a)
(b)
(c)
(d)
(e)

Fig. 6. **Failure cases.** (a) In all models and identities, hands are sometimes blurry. (b) The 4k and linearized models struggle with the edge of a t-shirt sleeve. (c) All models have unwanted transparency under the arm for this identity's avatar, but it is worse in 4k and linearized models. (d) All models struggle with the seat of the pants for this identity, but the 4k and linearized models struggle more. (e) The 4k and linear models suffer more degradation in the underarm for this identity.