Pixel Codec Avatars

Shugao Ma Tomas Simon Jason Saragih Dawei Wang
Yuecheng Li Fernando De La Torre Yaser Sheikh
Facebook Reality Labs Research
{shugao, tsimon, jsaragih, dawei.wang, yuecheng.li, ftorre, yasers}@fb.com

Abstract

Telecommunication with photorealistic avatars in virtual or augmented reality is a promising path for achieving authentic face-to-face communication in 3D over remote physical distances. In this work, we present the Pixel Codec Avatars (PiCA): a deep generative model of 3D human faces that achieves state of the art reconstruction performance while being computationally efficient and adaptive to the rendering conditions during execution. Our model combines two core ideas: (1) a fully convolutional architecture for decoding spatially varying features, and (2) a rendering-adaptive per-pixel decoder. Both techniques are integrated via a dense surface representation that is learned in a weakly-supervised manner from low-topology mesh tracking over training images. We demonstrate that PiCA improves reconstruction over existing techniques across testing expressions and views on persons of different gender and skin tone. Importantly, we show that the PiCA model is much smaller than the state-of-art baseline model, and makes multi-person telecommunicaiton possible: on a single Oculus Quest 2 mobile VR headset, 5 avatars are rendered in realtime in the same scene.

1. Introduction

Photorealistic Telepresence in Virtual Reality (VR) as proposed in [10, 26], describes a technology for enabling authentic communication over remote distances that each communicating party *feels the genuine co-location presence* of the others. At the core of this technology is the *Codec Avatar*, which is a high fidelity animatable human face model, implemented as the decoder network of a Variational AutoEncoder (VAE). Imagine a two-way communication setting. At the transmitter end, an encoding process is performed: cameras mounted on transmitter's VR headset capture partial facial images and an encoder model encodes the captured images into latent code of the decoder in realtime. At the receiver end a decoding process is performed: upon receiving the latent code over the internet, the decoder decodes the avatar's geometry and appearance so that the transmitter's realtime photorealistic face can be rendered onto the VR display.

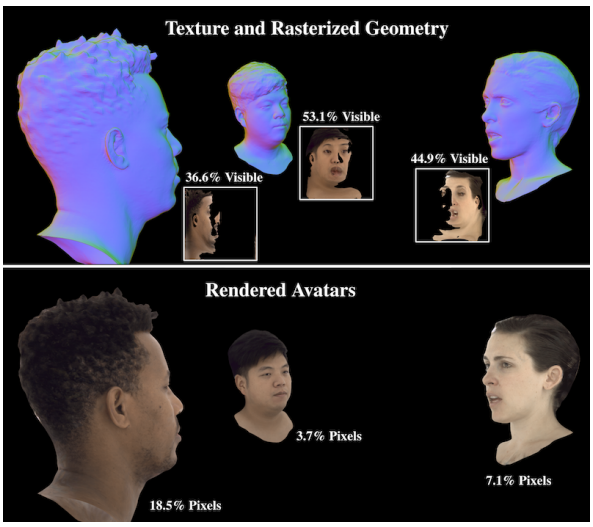


Figure 1. An multi-person configuration for teleconference in VR. At normal interpersonal distances [20], the head occupies only a subset of pixels in the display, where the amount of coverage largely depends on distance to the viewer. Roughly half of the head is not visible from any viewing angle due to self occlusion. Our method avoids wasting computation on areas that do not directly contribute to the final image. In first row we show the generated and rasterized geometry, along with texture maps showing visible pixels from the corresponding views; in the second row we show the rendered avatars and the percentage of pixels they cover over the entire image.

Multi-person communication via Photorealistic VR Telepresence will enable applications that are in great need in the modern society, such as family re-union over far physical distances in which each member genuinely feels the co-location presences of the others, or collaboration in remote working where team members can effectively communicate face-to-face in 3D. However, rendering with the decoder model proposed in [10] does not scale well with the number of communicating parties. Specifically, a full

texture of fixed resolution 1K×1K is decoded at each frame despite the distance of the avatar to the viewer and visibility of different facial regions. This leads to significant waste of computation when the avatar is far away, for which case the rendered avatar only consists a small number of pixels (Fig. 1), resulting in a large number of pixels in the decoded texture map unused. Also, most of the time half of the head is not visible due to self-occlusion, so the pixels in the decoded texture map for the occluded part are also unused. For a 2K display such as the one in Quest2, rendering more than 4 avatars amounts to computing more pixels than that of the display. This is obviously limiting, e.g. family reunion of more than 4 persons or team collaboration of more than 4 members are common place.

To solve this issue and scale the rendering to the number of persons in the VR telepresence, we should compute only the visible pixels, thus upper bounding the computation by the number of pixels of the display. Recent works in neural rendering such as the *defferred neural rendering*[24], the *neural point-based graphics*[2], the *implicit differentiable rendering* [27], use neural network to compute pixel values in the screen space instead of the texture space thus computing only visible pixels. However, in all these works, either a static scene is assumed, or the viewing distance and perspective are not expected to be entirely free in the 3D space. However, for telepresence, the ability to animate the face in realtime and render it from any possible viewing angle and distance is crucial.

In this paper, we present Pixel Codec Avatars (PiCA) that aims to achieve efficient and yet high fidelity dynamic human face rendering that is suitable for multi-person telepresence in VR on devices with limited compute. To avoid wasteful computation in areas of the face that do not contribute to the final rendering, PiCA employs per-pixel decoding only in areas of the image covered by a rasterization of the geometry. Similar to recent advances in implicit neural rendering [11, 17, 21], this decoder relies on a rich face-centric position encoding to produce highly detailed images. We employ two strategies to generate such encodings efficiently. First, we make use of the spatially-shared computation of convolutional networks in texture space to produce spatially varying expression- and view-specific codes at a reduced resolution (256×256). This is complemented by a pre-computed high resolution (1K×1K) *learned non-parametric positional encoding*, that is jointly rasterized into screen space similarly to [24]. To achieve an even higher resolution result, we further compliment the signal with 1D positional encodings at 10K resolution, independently for the horizontal and vertical dimensions of the texture domain. Together, these maps enable the modeling of sharp spatial details present in high resolution facial images. Because the best encoding values for the UV coordinates are directly learned from data, a low 8-dimensional encoding is sufficient to recover high frequencies. This is in contrast to existing positional encoding schemes (*e.g.* [11]) that achieve high details using sinusoidal functions, but require increasing the dimensionality by 20×, with corresponding computational costs. Secondly, in contrast to other works such as [24, 2, 27], we do not employ convolutions in screen space, but instead apply a shallow MLP at each contributing pixel. This has the advantage of avoiding visual artifacts during motion and stereo inconsistencies, as well as challenges in generalizing to changes in scale, rotation and perspective, all of which are common in interactive immersive 3D media.

Our other main insight is that the complexity of view-dependent appearance in prior work stems mostly from inadequate geometric models of the face. Recent work into implicit scene modeling (i.e. NeRF [11]) has demonstrated that complex view dependent effects such as specularity can be adequately modeled using a shallow network given good estimates of the scene's geometry. Inspired by these results, our construction involves a variational geometry decoder that is learned in a self-supervised manner, using image and depth reconstruction as a supervisory signal. The resulting mesh acquired from this decoder contains more accurate geometry information, substantially simplifying the view-dependent texture generation task, allowing for the use of lightweight pixel-wise decoding.

Contributions: Our contributions are as follows:

- We propose *Pixel Codec Avatar*, a novel light weight representation that decodes only the visible pixels on the avatar's face in the screen space towards enabling high fidelity facial animation on compute-constrained platforms such as mobile VR headsets.

- We make the two major technical innovations to achieve high quality decoding with a small model: *learned* positional encoding functions and fully convolutional dense mesh decoder trained in a weakly-supervised fashion.

2. Related Works

2.1. Deep 3D Morphable Face Models

3D Morphable Face Models (3DMFM) are a generative model for 3D human faces. The early works explore ways to represent human facial deformations and appearance with linear subspace representations. Blanz *et al*. [4] models shape and texture of human faces as vector spaces and generates new faces and expressions as linear combinations of the prototype vectors. Since then, blendshape models have been extensively studied and applied in animation - [9] provides a good overview of such methods. To achieve highly expressive models, a large number of blendshapes need to

be manually created and refined, *e.g*. the character of Gollum in the movie *Lord of the Rings* had 946 blendshapes taking over a year's time to create [12].

In recent years, deep learning techniques, especially generative models such as Variational Auto-Encoder (VAE) [8] and Generative Adversarial Networks (GAN) [7] have been actively studied for creating non-linear 3D Morphable Face Model analogues. Tewari *et al*. [23] propose a deep convolutional architecture for monocular face reconstruction, learned from morphable models. Lombardi *et al*. [10] propose to jointly model face shape and appearance with a VAE: the encoder encodes the facial mesh and texture into latent code with fully connected layers and convolutional layers respectively, and the decoder decodes back the facial mesh and view direction conditioned texture with fully connected layers and transposed convolutional layers respectively. This model has been referred to as a *Codec Avatar* by several subsequent works [26, 6, 15, 14] which animate this model using visual and/or audio sensory data. Tran *et al*. [25] also use an autoencoder to model geometry and texture, but train the model from unconstrained face images using a rendering loss. Bagautdinov *et al*. [3] uses a compositional VAE to model details of different granularities of facial geometry via multiple layers of hidden variables. Ranjan *et al*. [13] directly applies mesh convolution to build a mesh autoencoder while Zhou *et al*. [28] extends this idea and jointly models texture and geometry with mesh convolution, leading to a colored mesh decoder.

Generative Adversarial Network (GAN) is also explored. Among the first works that use GAN models to build 3DMFM, Slossberg *et al*. [18] build a GAN model that generates realistic 2D texture image as well as coefficients of a PCA based facial mesh model. Abrevaya *et al*. [1] maps mesh to geometry image (*i.e*. equivalent to position map in this paper) and builds a GAN model of the mesh that has decoupled expression and identity codes, and the decoupling is achieved with auxilary expression and identity classification tasks during training. Shamai *et al*. [16] also maps mesh into geometry image and builds GAN models using convolutional layers for both geometry and texture. Cheng *et al*. [5] proposes GAN model of facial geometry with mesh convolution.

The most distinctive feature of PiCA against the previous 3DMFM is that the pixel decoder decodes color at each pixel given underlying geometry that is generated and rasterized to screen space, hence adaptive resolution and computational cost is achieved. In contrast, in all previous methods, texture is either modeled as a 2D texture map [10, 25, 18] thus fixing the output resolution, or is modeled at mesh vertices [28, 16], thus mesh density determines the rendering resolution. Another advantage is that our method explicitly models the correlation between geometry and texture in the per-object decoding step, which is lacking in most previous 3D DFMM models.

2.2. Neural Rendering

Our method is also related to recent works on Neural Rendering and [22] provides a good survey of recent progress in this direction. In particular, Thies *et al*. [24] propose deferred neural rendering with a neural texture, which in spirit is close to our work: neural textures, *i.e*. a feature output from a deep neural net, is rasterized to screen space and another neural net, *i.e*. the neural renderer, computes colors from it. However, their work does not target realtime animation or dynamics, and the usage of a heavy U-Net for rendering the final result is not possible in our setting. Aliev *et al*. [2] proposes neural point-based graphics, in which the geometry is represented as a point cloud. Each point is associated with a deep feature, and a neural net computes pixel values based on splatted feature points. While being very flexible in modeling various geometric structures, such point-cloud based methods are not yet as efficient as mesh-based representations for modeling dynamic faces, for which the topology is known and fixed. Yariv *et al*. [27] models the rendering equation with a neural network that takes the viewing direction, 3D location and surface normals as input. Mildenhall *et al*. [11] proposes a method for synthesizing novel views of complex scenes and models the underlying volumetric scene with a MLP: the MLP takes a positional encoded 3D coordinate and view direction vector and produces pixel values. A closely related idea is presented in [17], where a MLP with sinusoidal activation functions is used to map locations to colors. The spectral properties of mapping smooth, low-dimensional input spaces to high-frequency functions using sinusoidal encodings was further studied in [21]. Our method is inspired by these methods in using the Pixel Decoder to render image pixels, but we make innovations to adapt these ideas for the problem of creating high-quality 3DMFM with lightweight computations, including a learned positional encodings and a dense geometry decoder.

3. Pixel Codec Avatar

The Pixel Codec Avatar is a conditional variational autoencoder (VAE) where the latent code describes the state of the face (e.g., facial expression) and the decoder produces realistic face images (see Fig.2) conditioned on a viewing direction. At runtime, latent codes can be produced using a face tracker to estimate the facial expression (e.g., from cameras mounted on a VR headset [10, 26, 6]), and the estimated code can be used to decode and render realistic face images. At training time, a variational encoder is used to produce the latent codes using multiview training data, similarly to Lombardi *et al*. [10] (see Fig. 3(a)). The decoder distributes computation across two phases: the *Per-Object Decoding* produces the dense mesh and a small map of view

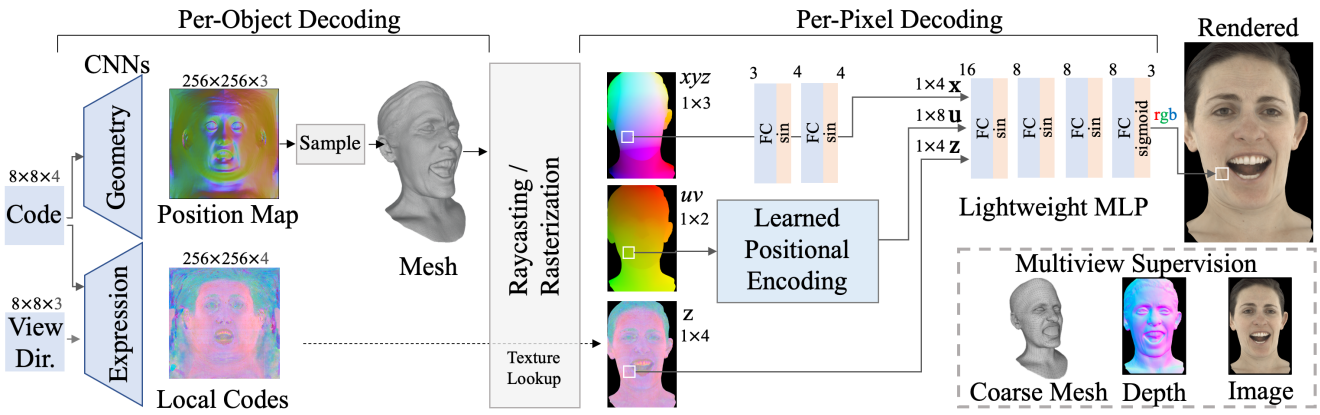


Figure 2. A *Pixel Codec Avatar* renders realistic faces by decoding the color of each rasterized or raycast pixel using a shallow SIREN [17] that takes as input a local expression code, $\mathbf{z}$, the 3D coordinates in object space, $\mathbf{x}$, and the positional encoded surface coordinates, $\mathbf{u}$, (Section 4). This particular combination allows the feature dimensions and network size to remain small and computationally efficient while retaining image fidelity (Section 6). The local expression codes and geometry are decoded using fully convolutional architectures from a global latent code and the viewing direction (Section 3), and require only small resolutions of 256×256. Learnable components (in blue) are supervised on multiview images, depth, and tracked coarse mesh.

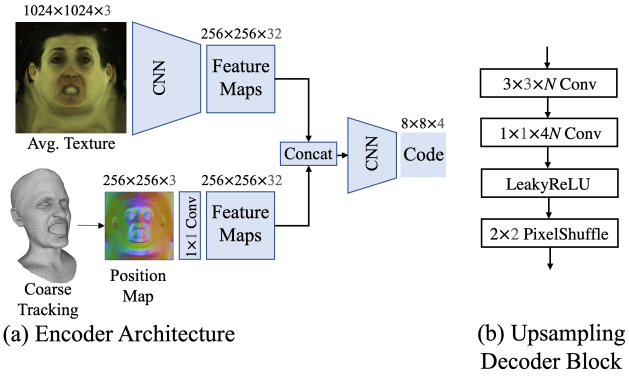
conditioned expression codes (Left of Fig.2), and the *Per-Pixel Decoding* computes the on-screen facial pixel values after determining visibility through rasterization or raycasting. We use a *pixel decoder* f in this second step:

$$\mathbf{c} = f(\mathbf{p}), \quad \mathbf{p} = [\mathbf{z}, \mathbf{x}, \mathbf{u}] \tag{1}$$

where $\mathbf{c}$ is the decoded RGB color for a facial pixel, and $\mathbf{p}$ is the feature vector for that pixel which is concatenation of the local facial expression code $\mathbf{z}$, the encoded face-centric 3D coordinates $\mathbf{x}$, and the encoded surface coordinates (UV) $\mathbf{u}$. We parameterize f as a small SIREN (see Fig. 2) and we describe the encoding inputs in Section 4. The right side of Fig.2 illustrates the Per-Pixel Decoding. We outline the major components:

Encoder (see Fig. 3)(a)) encodes the average texture, computed over unwrapped textures of all camera views, and a tracked mesh into a latent code. Note this tracked mesh is coarse, containing 5K vertices, and doesn't contain vertices for tongue and teeth. We only assume availability of such coarse mesh for training because face tracking using dense mesh over long sequences with explicit teeth and tongue tracking is both challenging and time consuming. Requiring only coarse mesh in training makes our method more practical. In Lombardi *et al*. [10], the 3D coordinates of mesh vertices are encoded using a fully connected layer and fused with texture encoder; in contrast, we first convert the mesh into a position map using a UV unwrapping of the mesh. Joint encoding of the geometry and texture is then applied, and the final code is a grid of spatial codes, in our case an 8x8 grid of 4 dimensional codes.

Geometry Decoder takes the latent code as input and decodes a dense position map describing face-centric 3D coordinates at each location. The architecture is fully convolutional, and the basic building block is shown in Fig. 3(b). We convert the position map to a dense mesh by sampling at each vertex's UV coordinates, and rasterize it to determine visible pixels. In our experiments, the position map is 256×256 and the extracted dense mesh has 65K vertices.



(a) Encoder Architecture (b) Upsampling Decoder Block

Figure 3. (a) The encoder. (b) The basic block in the geometry decoder and expression decoder.

Expression Decoder uses the latent code and the viewing direction to decode a low resolution, view-dependent map of local codes. It consists of the decoder block in Fig. 3(b) and the output map is 256×256 in our experiments.

Pixel Decoder decodes the color at each facial pixel given $\mathbf{p}$. Specifically, rasterization determines whether a screen pixel corresponds to a visible mesh point, and, if so, the triangle id and barycentric coordinates of the mesh point. This allows us to compute the encoding inputs $\mathbf{p}$ from the expression map, the vertex coordinates, and the UV coordinates of the triangle. Inspired by the pixel-wise decoding of

images in Sitzmann *et al*. [17], the *pixel decoder* is designed as a SIREN. However, we use a very lightweight network by design, with 4 layers and a total of 307 parameters. We utilize effective encoding in $\mathbf{u}$ to produce facial details with such a light model, described in Section 4.

4. Positional Encodings for Pixel Decoders

While neural networks and MLPs in particular can represent functions of arbitrary complexity when given sufficient capacity, lightweight MLPs tend to produce low-frequency outputs when given smoothly varying inputs [17, 21, 11]. Thus, given only the smooth face-centric coordinates and surface coordinates as input, a lightweight pixel decoder tends to produce smooth output colors for neighboring pixels, leading to a loss of sharpness in the decoded image. Instead, we encode information about such spatial discontinuities at the input of the Pixel Decoder using two strategies: a low resolution local expression code $\mathbf{z}$ for dynamics, and a learned non-parametric positional encoding $\mathbf{u}$ of surface coordinates for detail. These complement the mesh coordinate input $\mathbf{x}$, which encodes face-centric xyz coordinates using a two-layer SIREN.

Facial Expression Positional Encodings The global expression code, *i.e*. output of the *Encoder*, is decoded to a low resolution map of local expression codes (bottom left of Fig.2) and is further rasterized to the screen space (bottom middle in Fig.2). This leads to a low dimensional encoding $\mathbf{z}$ of local facial expression at each pixel position. We find it crucial to use the local expression codes for decoding high fidelity facial dynamics.

Facial Surface Positional Encodings The local expression codes are too low resolution to capture high-frequency details. We therefore additionally provide the pixel decoder with a positional encoding $\mathbf{u}$ of the facial surface coordinates (u, v) at each pixel. While generic positional encodings such as sinusoids [11] may achieve highly detailed reconstructions, they require a large number of frequency levels and therefore high dimensionality, incurring computational cost. Instead, we dramatically reduce the dimensionality of the input features by designing a *learned non-parametric positional encoding* function,

$$\mathbf{u} = [m_{uv}(u, v),\ m_u(u),\ m_v(v)] \tag{2}$$

where m_{uv} jointly encodes both u and v; m_u and m_v encodes u and v respectively. We directly model m_{uv}, m_v and m_u as non-parametric functions that retrieve a low-dimensional encoding from a learned encoding map given (u, v). Specifically, m_{uv} retrives a 4 dimensional vector from a 1024×1024×4 encoding map at position (u, v) using bilinear interpolation; and, similarly, m_u and m_v retrieve 2-dimensional vectors from two separate 10000x1 maps respectively. All three maps are jointly learned with the rest of the model. Intuitively, m_{uv}, m_u, and m_v are piece-wise linear functions with 1K×1K breakpoints in 2D, and 10K breakpoints in 1D respectively, and the breakpoints' values in the maps contain spatial discontinuity information on the face surface, learned directly from the data. We use 1D encoding functions m_u and m_v in addition to the 2D encoding function m_{uv} as a cost-effective way to model higher resolution while avoiding a quadratic increase in model parameters. Empirically, we found that the combination of the two generates better reconstructions than using either one in isolation (Section 6.2).

5. Joint Learning with a Dense Mesh Decoder

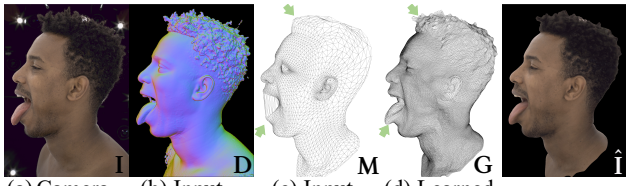
The geometry used for pixel decoders needs to be accurate and temporally corresponded to prevent the pixel decoders from having to compensate for geometric misalignments via complex view-dependent texture effects. To achieve this, we learn the variational decoder of geometry and expression jointly with the pixel decoder.

We use a set of multiview images, $\mathbf{I}_t^c$, (i.e., image from camera c at frame t), with calibrated intrinsics $\mathbf{K}_c$ and extrinsics, $\mathbf{R}_c|\mathbf{t}_c$. For a subset of frames we compute depth maps $\mathbf{D}_t^c$ using multiview stereo (MVS). Additionally, we use a vision-based face tracker to produce a coarse mesh $\mathbf{M}_t$ represented as a position map to provide rough temporal correspondences. Note, however, that the input tracked mesh is low resolution, lacking detail in difficult to track areas like the mouth and eyes (Fig. 4(c)). Intuitively, the more accurate the geometry is, the easier and better the pixel decoder may decode the pixel's color. Therefore, our geometry decoder generates a position map $\mathbf{G}$ of a dense mesh of ∼65K vertices, including the mouth interior, without direct supervision from a tracked *dense* mesh (Fig. 4(d)).

For each training sample, we compute an average texture $\mathbf{T}_t^{\text{avg}}$ by backprojecting the camera images onto the coarse tracking mesh, similarly to [10]. The texture and the position map computed from the coarse mesh are used as input to the convolutional encoder, $E(\cdot)$, Fig. 3(a), to produce the latent code $\mathbf{Z}{=}E(\mathbf{T}_t^{\text{avg}}, \mathbf{M}_t) \in \mathbb{R}^{8\times8\times4}$, where the channel dimension is last. Additionally, we compute the camera viewing direction as $\mathbf{R}_c^T\mathbf{t}_c$ normalized to unit length, in face-centric coordinates. We tile this vector into an 8x8 grid $\mathbf{V}{\in}\mathbb{R}^{8\times8\times3}$. The geometry and expression decoders in Fig. 2 produce the geometry and local codes,

$$\mathbf{G} = D_g(\mathbf{Z}), \qquad \mathbf{E} = D_e(\mathbf{Z}, \mathbf{V}), \tag{3}$$

where $\mathbf{G}{\in}\mathbb{R}^{256\times256\times3}$ is a position map, and $\mathbf{E}{\in}\mathbb{R}^{256\times256\times4}$ is a map of expression codes. The position map is sampled at each vertex's UV coordinates to produce a mesh for rasterization. Rasterization assigns to a pixel at screen position $\mathbf{s}$ its corresponding uv coordinates and face-centric xyz coordinates, from which the encoding



(a) Camera Image (b) Input Depth (c) Input Mesh (d) Learned Dense Mesh (e) Render

Figure 4. We supervise on (a) images, (b) depth, and (c) a coarse tracking mesh of 7K vertices, from which we learn a corresponded, dense face mesh (d) at a higher resolution of 65K vertices, even in places where the coarse tracked mesh provides no information. The final render (e) can represent difficult-to-track expressions, e.g., involving the tongue.

$\mathbf{p}$ is derived as described in Sect. 4. The final pixel color is decoded producing a rendered image, $\hat{\mathbf{I}}_t^c(\mathbf{s}) = f(\mathbf{p})$. At each SGD step, we compute a loss

$$\mathcal{L} = \lambda_i \mathcal{L}_I + \lambda_d \mathcal{L}_D + \lambda_n \mathcal{L}_N + \lambda_m \mathcal{L}_M + \lambda_s \mathcal{L}_S + \lambda_{kl} \mathcal{L}_{KL} \ , \quad (4)$$

where $\mathcal{L}_I = ||\mathbf{I}_t^c - \hat{\mathbf{I}}_t^c||_2$ measures image error, and $\mathcal{L}_D = ||(\mathbf{D}_t^c - \hat{\mathbf{D}}_t^c) \odot \mathbf{W}_D||_1$ measures depth error, where $\mathbf{W}_D$ is a mask selecting regions where the depth error is below a threshold of 10mm. We additionally use a normal loss, $\mathcal{L}_N = ||(N(\mathbf{D}_t^c) - N(\hat{\mathbf{D}}_t^c)) \odot \mathbf{W}_D||_2$ where $N(\cdot)$ computes normals in screen space and encourages sharper geometric details. The remaining terms are regularizations: $\mathcal{L}_M = ||(S(\mathbf{G}) - S(\mathbf{M}_t)) \odot \mathbf{W}_M||_2$, where $S(\cdot)$ is a function that samples the position map at the vertex UVs, penalizes large deviations from the coarse tracking mesh using a mask $\mathbf{W}_M$ to avoid penalizing the mouth area (where the tracked mesh is inaccurate). $\mathcal{L}_S$ is a Laplacian smoothness term [19] on the dense reconstructed mesh. These terms prevent artifacts in the geometry stemming from noise in the depth reconstructions, images with no depth supervision, and noisy SGD steps. Implementation details for the smoothness term and on how differentiable rendering is used to optimize these losses can be found in the supplemental materials. $\mathcal{L}_{KL}$ is the Kullback-Leibler divergence term of the variational encoder.

The above procedure recovers detailed geometry in the decoded dense mesh that is not captured in the input tracked meshes. Especially note-worthy is the automatic assignment of vertices inside the mouth to the teeth and tongue, as well as hair, see Fig. 6 for examples.

6. Experiments

Experiment Setting We evaluate our model on 6 identities on 5 different viewing directions: front, upward, downward, left and right (see example images in the supplemental material). We capture multiview video data for each identity using two face capture systems: Subject 1-4 are captured with 40 cameras with 50mm focal length, while Subject 5 and 6 are captured with 56 cameras at 35mm focal length. We use images of size 2048×1334 for training and testing. The data of each identity consists of expressions, range of facial motion, and reading sentences. We randomly select expressions and sentence readings as testing data, leading to

	Model	Front	Up	Down	Left	Right
S1	Baseline	23.03	20.78	18.13	16.32	18.97
	Full	**21.39**	**19.71**	**17.52**	**15.52**	**18.00**
	No-UV	22.16	20.38	18.28	16.27	18.57
	Coarse	21.64	20.04	17.84	16.02	18.69
S2	Baseline	19.53	20.90	16.62	15.44	13.52
	Full	**18.31**	**19.96**	**16.36**	**14.28**	**12.14**
	No-UV	19.34	20.52	17.61	15.40	13.29
	Coarse	19.88	21.62	17.97	15.97	13.92
S3	Baseline	24.41	22.83	16.54	16.09	16.81
	Full	**23.11**	**22.22**	**16.04**	**15.29**	**15.64**
	No-UV	23.95	22.99	16.42	15.86	16.12
	Coarse	23.94	23.04	16.44	15.81	16.79
S4	Baseline	7.26	6.03	7.34	7.15	7.76
	Full	**6.81**	**5.78**	**7.33**	**7.05**	**7.63**
	No-UV	7.20	6.13	7.40	7.32	8.05
	Coarse	7.19	6.02	7.48	7.21	8.25
S5	Baseline	9.20	10.87	7.24	7.27	6.54
	Full	**8.74**	**10.37**	**7.16**	**7.09**	**6.53**
	No-UV	9.06	10.96	7.39	7.46	6.76
	Coarse	9.09	10.64	7.49	7.49	6.56
S6	Baseline	6.86	6.53	5.85	5.66	5.29
	Full	**6.22**	**6.06**	**5.39**	**4.97**	**4.95**
	No-UV	6.86	6.72	5.85	5.90	5.62
	Coarse	6.54	6.33	5.69	5.29	5.16

Table 1. **MSE** on pixel values of the rendered images against the ground truth images on test set, evaluated on 5 views. *Baseline* is the model in [10]; *Full* is our model PiCA (Fig.2), *No-UV* is PiCA variant that is not using surface coordinates; *Coarse* is PiCA variant that decodes coarse mesh (7K vertices). *Full* PiCA model consistently outperform others on all tested identities over all views.

	Model	Front	Up	Down	Left	Right
S1	**Full**	**21.39**	19.71	**17.52**	**15.52**	**18.00**
	NERF-PE	21.85	20.10	17.86	15.90	18.61
	UV-NoPE	21.45	19.93	17.70	15.98	18.53
	2D-PE	21.56	19.85	17.97	15.98	18.80
	1D-PE	21.40	**19.67**	17.60	15.70	18.29
S2	**Full**	**18.31**	**19.96**	**16.36**	**14.28**	**12.14**
	NERF-PE	18.99	20.35	17.35	15.19	13.18
	UV-NoPE	19.17	20.51	17.53	15.40	13.29
	2D-PE	19.05	20.23	17.47	15.02	13.02
	1D-PE	19.30	20.61	17.64	15.43	13.39
S6	**Full**	**6.22**	6.06	**5.39**	**4.97**	**4.95**
	NERF-PE	6.41	6.16	5.60	5.29	5.14
	UV-NoPE	6.59	6.53	5.68	5.33	5.24
	2D-PE	6.28	**6.00**	5.48	5.26	5.09
	1D-PE	6.58	6.39	5.68	5.26	5.21

Table 2. Ablation on usage of UV coordinates: encoding with learned encoding maps (*Full*), directly using UV (*UV-NoPE*), encoding with sinusoidal functions [11] (*NERF-PE*), joint encoding only (*2D-PE*) and separate encoding only (*1D-PE*)

		18cm (2.7M)	65cm (0.9M)	120cm (0.2M)
DSP Step	Baseline	44.76 ms	44.76 ms	44.76 ms
	PiCA	2.16 ms	2.16 ms	2.16ms
GPU Step	Baseline	2.67 ms	2.47 ms	1.94 ms
	PiCA	8.70 ms	3.27 ms	2.70 ms

Table 3. Runtime performance on the Oculus Quest 2, measured at 3 different avatar distances (the numbers in parenthesis are avatar pixels to render). Note that 60-120cm are typical interpersonal distances [20], while 18cm would be considered intimate.

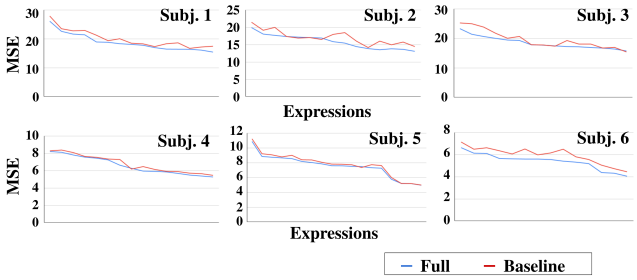


Figure 5. The **MSE** distribution over test expressions, sorted in decreasing order for the *Full* model: x-axis is expressions and y-axis is MSE. We can see that the performance of our model is similar or better than the baseline across expressions for all identities.

$\sim$12K frames for training and $\sim$1K frames for testing per identity. The total number of images is roughly the number of frames multiplied by the number of cameras. All models are trained with batchsize 4, at learning rate 0.001, for 400000 iterations. The weights for different loss terms in Eq. 4 for λ_i, λ_d, λ_n, λ_m, λ_s and λ_{kl} are set to 2, 10, 1, 0.1, 1 and 0.001 respectively. We report Mean Squared Error (MSE) between rendered image and original image on rasterized pixels on testing data as the evaluation metric for reconstruction quality. Note that the results of different identities are not directly comparable due to different camera settings and subject appearance such as facial skin tone and hair style.

6.1. Overall Performance

The baseline model has 19.08*M* parameters and PiCA has 5.47*M*. In particular, the pixel decoder of PiCA only has 307 parameters. **When rendering 5 avatars (evenly spaced in a line, 25cm between neighboring pair) in the same scene on a Oculus Quest 2, PiCA runs at $\sim$50 FPS on average, showing the possibility of multi-way telepresence call. In Table 1 and Fig. 5 we report quantitative comparisons which show PiCA consistently achieves better reconstruction across all tested identities, expressions and views, despite a 3.5$\times$ reduction in model size and much faster computation (Table 3).** Specifically, Table 1 compares the reconstruction quality over 5 views, averaged over all testing expressions. Fig. 5 plots MSE values of Full and Baseline over all testing expressions (sorted in decreasing order of Full's results). Qualitative examples are shown in Fig. 6 and we invite the readers to see more high resolution results in supplemental materials. Example result frames for both our Full model (left) and the baseline model (right) are shown, and we also show local regions at higher resolution for closer inspection. Overall, both models produce very realistic looking faces. Our model produces sharper results in many facial regions, especially the selected regions showing teeth, tongue, and hair.

6.2. Ablation Studies

UV Positional Encoding Many details of the facial surface is represented as discontinuities in color values in neighboring pixels, *e.g.* a skin pixel adjacent to a hair pixel. We model such discontinuities with learned encoding maps such that the encoding function is piece-wise linear with the map entries as the learned breakpoint values (Section 4). In this section, we study the benefit of this proposed method. We train a PiCA variant *No-UV* that does not use UV coordinates for decoding pixel values. In Table 1 one can see that Full PiCA model consistently outperforms the No-UV variant, showing clear advantage of using encoded UV coordinates. We Further compare our approach with directly using UV without encoding, and encoding UV with sinusoidal functions [11]. We train two additional PiCA variants *UV-NoPE* that uses UV without any encoding, and *NERF-PE* that encodes UV using the encoding function of [11] (a 40-dimensions code compared to 8-dimensions for Eq. (2)). The comparison results are shown in Table 2. The *Full* model consistently outperforms both variants over all tested views and subjects, proving the effectiveness of encoding UV with learned encoding maps. We also ablate on our encoding scheme: we train a PiCA variant *2D-PE* that only performs 2D joint encoding (m_{uv} in Eq. (2)) and *1D-PE* that only performs 1D separate encodings (m_u, m_v). The comparison results are shown in Table 2. The Full PiCA model combining both joint encoding and 1D encodings outperforms these two variants, showing that the two encoding methods are complementary and by combining both we can achieve consistent performance improvement.

Dense Mesh Decoder In Fig. 6, we show depth images alongside the rendered images. The dense mesh generated by our model contains more geometry information and the corresponding rendered images are sharper: in particular, one may inspect the teeth, tongue and hair regions. In Fig. 7 we compare novel viewpoint rendering results of Full and Baseline at a viewing position that is very close to the mouth: there are no such views in our training set. While the baseline results look like a pasted plane inside the mouth, ours look more realistic thanks to the more accurate geometry in the generated dense mesh *e.g.* at teeth, tongue and lips. For quantitative study, we train a PiCA model variant *Coarse* which decodes coarse meshes of the same topology used in [10]. In Table 1, we evaluate it on

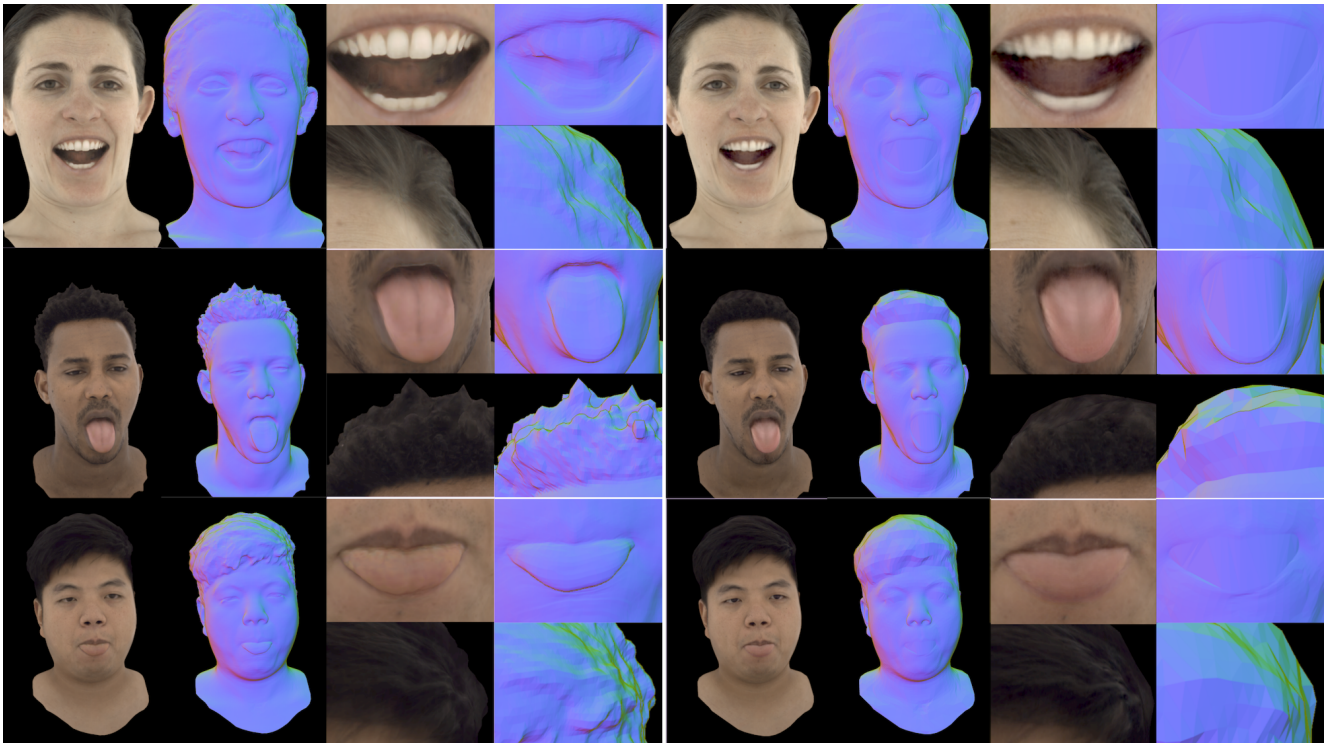


Figure 6. Example rendered faces comparing our *Full* model (left) with the baseline [10] (right). For each example, we show the rendered full face and the depth image, and close looks for two facial regions. The visual qualities of rendered images are good for both models, while our model produce sharper details at teeth, tongue and hair. The depth images show more geometry details generated by our model.

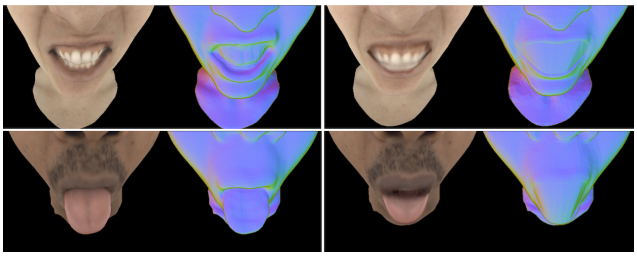


Figure 7. Rendering at a novel viewing position, much closer to the mouth than any training views. Two example frames are shown with the rendered depth as well: left column is PiCA Full, and right is the Baseline model [10], best viewed when magnified.

the test set, and the results show it being consistently inferior to the Full PiCA model, illustrating the benefit of the dense geometry decoder in the Pixel Codec Avatar.

6.3. Runtime Performance on Mobile SoC

We present runtime performance on a Oculus Quest 2 VR headset [1] in Table 3. We measure the time spent on both the DSP (Digital-Signal-Processing unit) and the GPU steps - note the two steps are pipelined at runtime. There is 20× reduction for DSP time from Baseline to PiCA. Overall, Baseline runs at ∼22 FPS, while PiCA hits the Quest 2's maximum framerate at 90 FPS. While the baseline model always decodes entire texture map of the avatar head at fixed resolution, PiCA decodes only visible regions with resolution adaptive to the distance of the avatar. Further more, PiCA allows a pipelined decoding process balanced in computation load distribution on a mobile SoC: while the per-object decoding needs to be done on the DSP for the convolution operations, the lightweight pixel decoder can be implemented in the highly optimized fragment shader so that the per-pixel decoding can be done on the GPU. In contrast, for the baseline model the decoding computation of the mesh and the texture needs to be done entirely on the DSP and the GPU only performs the final rendering given decoded texture and mesh.

[1]The baseline model and the geometry and expression decoders of PiCA are 8-bit quantized to execute on the DSP, with small quality drops.

7. Conclusion and Future Work

We present the Pixel Codec Avatar as a high quality lightweight deep deformable face model, as a potential technology for enabling multi-person telecommunication in virtual reality on a mobile VR headset. This work only focuses on the *decoder* and we can follow the method in Wei *et al.* [26] to build the *encoder* for the telepresence communication system. Achieving high fidelity low latency telepresence communication by improving the encoder and decoder models is the main direction for future work.

A. Appendices

A.1. Encoder and Decoder Architectures

Encoder The encoder consists of three major components: the *tex-head*, *geom-head* and the *tex-geom-encoder*. The *tex-head* has two blocks of *conv+leakyrelu*, where the *conv* for both layers have kernel size 4, stride 2. The first one has 512 output channels, and the second has 256 channels. The *geom-head* has one block of *conv+leakyrelu* where the *conv* has kernel size 1, stride 1 and output channel number 256. The output of *tex-head* and *geom-head* are both 256x256x256, and are concatenated and passed to *tex-geom-encoder*, which has 5 blocks of *conv+leakyrelu*. The kernel size and stride are all 4 and 2 for all *conv*, while the output channel numbers are 128, 64, 32, 16 and 8 respectively. The output of *tex-geom-encoder* is further passed to two separate 1x1 *conv* layers to produce mean and variance. *leakyrelu* always having leaky threshold set to 0.2.

Per-Object Decoder This decoder decodes the local expression code and the dense mesh from the latent code which is of dimension 8x8x4. It consists the *geometry decoder*, containing 5 blocks of the building block showing in Fig. 3b in the main text, with output channel numbers 32, 16, 16, 8, 3 respectively. The output size is 256x256x3, from which the dense mesh can be retrieved using the uv coordinates of the mesh vertices. The *expression decoder* takes the concatenated latent code and view direction as input, which is of size 8x8x7, and it contains 5 building blocks as showing in Fig. 3b as well, with output channel numbers 32, 16, 16, 8, 4 respectively. Note in both cases the first *conv* in the block in Fig. 3b has a per-channel per-spatial location bias parameter, following [10].

Pixel Decoder The entries in the 2D and 1D encoding maps in the pixel decoder are initialized to have uniform distribution in the range [-1, 1]. The 3D coordinate input (x,y,z) are first converted to a 4-dimensional vector via a two layer SIREN with output channel numbers 4 and 4 respectively, and then it is concatenated with the encoded uv (8-dimension) and the local expression code to form a 16 dimensional input to the final SIREN. The final SIREN has 4 layers with output channel numbers 8, 8, 8, 3 respectively to compute the RGB color at a pixel.

A.2. Geometric Smoothness

To recap, $\mathbf{G} \in \mathbb{R}^{w \times w \times 3}$, with w=256, is a decoded position map describing the geometry, and $S(\cdot) : \mathbb{R}^{w \times w \times 3} \rightarrow \mathbb{R}^{N_V \times 3}$ is a function that bilinearly interpolates the position map at the vertex uv locations to produce face-centric xyz locations for the set of N_V mesh vertices, where N_V is the number of vertices in a fixed mesh topology. Our geometric smoothness regularization term $\mathcal{L}_S$ combines two common gradient-based smoothness energies,

$$\mathcal{L}_S = \lambda_g \left[||\mathcal{D}_x(\mathbf{G})||_2 + ||\mathcal{D}_y(\mathbf{G})||_2 \right] \quad (5)$$
$$+ \lambda_l ||\mathbf{W}_L \mathbf{L}(S(\mathbf{G}) - \mathbf{V}_\mu)||_2, \quad (6)$$

where we identify:

Gradient Smoothness. Linear operators $\mathcal{D}_*$ compute the x and y derivatives of the position map using finite differences. These terms prevent large changes across neighboring texels in the position map itself.

Mesh Laplacian. The linear operator $\mathbf{L} \in \mathbb{R}^{N_V \times N_V}$ represents the mesh Laplacian discretized using cotangent weights [19] computed on the coarse neutral input mesh. Here, $\mathbf{V}_\mu \in \mathbb{R}^{N_V \times 3}$ is a mean face mesh used as a regularization target. The diagonal matrix $\mathbf{W}_L \in \mathbb{R}^{N_V \times N_V}$ weights the regularization on hair and mouth vertices at 1.25 and the remaining vertices at 0.25. This regularization prevents the differential mesh coordinates (as computed by the mesh Laplacian) from deviating excessively from the regularization target.

The regularization target $\mathbf{V}_\mu$ is initialized with the coarse neutral mesh geometry. However, because the coarse geometry lacks detail in the mouth, hair, and eye regions, using it as a regularization target tends to oversmooth these areas. Therefore, we update the target on the fly during training using exponential smoothing, obtaining a slowly-changing, moving average estimate of the mean face geometry at dense resolutions. At every SGD iteration, we update $\mathbf{V}_\mu$ as follows:

$$\mathbf{V}_\mu \leftarrow (1 - \lambda_\mu)\mathbf{V}_\mu + \lambda_\mu \frac{1}{B} \sum_{b=1}^{B} S(\mathbf{G}_b), \quad (7)$$

where $\lambda_\mu = 1e^{-4}$ and $b \in \{1 \ldots B\}$ iterates over samples in the SGD batch. No SGD gradients are propagated by the update in Eq. (7).

In our experiments, we set $\lambda_l = 0.1$ and $\lambda_g = 1$.

A.3. Differentiable Rasterizer

We use a differentiable rasterizer for computing the screen space inputs given dense mesh and local expression code map, as illustrated in Fig.2 in the main text. Note that the geometry information affects the final decoded image via two gradient paths: one is in the rasterization, and the other is as input to the pixel decoder. We empirically found that allowing gradient from the image loss to pass to the geometry decoder from both paths leads to unstable training and geometry artifacts, so we disable the second gradient path mentioned above to achieve stable training. Intuitively, this is to enforce that the geometry decoder should focus on producing correct facial shape, instead of *coordinating* with the pixel decoder to produce correct color values.

References

[1] Victoria Fernandez Abrevaya, Adnane Boukhayma, Stefanie Wuhrer, and Edmond Boyer. A decoupled 3d facial shape model by adversarial training. October 2019.

[2] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. *arXiv preprint arXiv:1906.08240*, 2019.

[3] Timur Bagautdinov, Chenglei Wu, Jason Saragih, Pascal Fua, and Yaser Sheikh. Modeling facial geometry using compositional vaes. June 2018.

[4] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. page 187–194, 1999.

[5] Shiyang Cheng, Michael Bronstein, Yuxiang Zhou, Irene Kotsia, Maja Pantic, and Stefanos Zafeiriou. Meshgan: Non-linear 3d morphable models of faces, 2019.

[6] Hang Chu, Shugao Ma, Fernando De la Torre, Sanjia Fidler, and Yaser Sheikh. Expressive telepresence via modular codec avatars. 2020.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. 27:2672–2680, 2014.

[8] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. 2014.

[9] J. P. Lewis, K. Anjyo, T. Rhee, M. Zhang, F. Pighin, and Zhigang Deng. Practice and theory of blendshape facial models. 2014.

[10] Stephen Lombardi, Jason Saragih, Tomas Simon, and Yaser Sheikh. Deep appearance models for face rendering. *TOG*, 37(4), 2018.

[11] 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. 2020.

[12] Bay Raitt. The making of gollum. Presentation at U. Southern California Institute for Creative Technologies's Frontiers of Facial Animation Workshop, August 2004.

[13] Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J. Black. Generating 3d faces using convolutional mesh autoencoders. September 2018.

[14] Alexander Richard, Colin Lea, Shugao Ma, Juergen Gall, Fernando de la Torre, and Yaser Sheikh. Audio- and gaze-driven facial animation of codec avatars. 2021.

[15] Gabriel Schwartz, Shih-En Wei, Te-Li Wang, Stephen Lombardi, Tomas Simon, Jason Saragih, and Yaser Sheikh. The eyes have it: An integrated eye and face model for photorealistic facial animation. *ACM Trans. Graph.*, 39(4), 2020.

[16] Gil Shamai, Ron Slossberg, and Ron Kimmel. Synthesizing facial photometries and corresponding geometries using generative adversarial networks. *ACM Trans. Multimedia Comput. Commun. Appl.*, 15(3s), 2019.

[17] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. 2020.

[18] Ron Slossberg, Gil Shamai, and Ron Kimmel. High quality facial surface and texture synthesis via generative adversarial networks. September 2018.

[19] Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. Laplacian surface editing. pages 179–188, 2004.

[20] Agnieszka Sorokowska, Piotr Sorokowski, Peter Hilpert, Katarzyna Cantarero, Tomasz Frackowiak, Khodabakhsh Ahmadi, Ahmad M. Alghraibeh, Richmond Aryeetey, Anna Bertoni, Karim Bettache, Sheyla Blumen, Marta Błażejewska, Tiago Bortolini, Marina Butovskaya, Felipe Nalon Castro, Hakan Cetinkaya, Diana Cunha, Daniel David, Oana A. David, Fahd A. Dileym, Alejandra del Carmen Domínguez Espinosa, Silvia Donato, Daria Dronova, Seda Dural, Jitka Fialová, Maryanne Fisher, Evrim Gulbetekin, Aslıhan Hamamcıoğlu Akkaya, Ivana Hromatko, Raffaella Iafrate, Mariana Iesyp, Bawo James, Jelena Jaranovic, Feng Jiang, Charles Obadiah Kimamo, Grete Kjelvik, Fırat Koç, Amos Laar, Fívia de Araújo Lopes, Guillermo Macbeth, Nicole M. Marcano, Rocio Martinez, Norbert Mesko, Natalya Molodovskaya, Khadijeh Moradi, Zahrasadat Motahari, Alexandra Mühlhauser, Jean Carlos Natividade, Joseph Ntayi, Elisabeth Oberzaucher, Oluyinka Ojedokun, Mohd Sofian Bin Omar-Fauzee, Ike E. Onyishi, Anna Paluszak, Alda Portugal, Eugenia Razumiejczyk, Anu Realo, Ana Paula Relvas, Maria Rivas, Muhammad Rizwan, Svjetlana Salkičević, Ivan Sarmány-Schuller, Susanne Schmehl, Oksana Senyk, Charlotte Sinding, Eftychia Stamkou, Stanislava Stoyanova, Denisa Šukolová, Nina Sutresna, Meri Tadinac, Andero Teras, Edna Lúcia Tinoco Ponciano, Ritu Tripathi, Nachiketa Tripathi, Mamta Tripathi, Olja Uhryn, Maria Emília Yamamoto, Gyesook Yoo, and Jr. John D. Pierce. Preferred interpersonal distances: A global comparison. *Journal of Cross-Cultural Psychology*, 48(4):577–592, 2017.

[21] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020.

[22] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, Rohit Pandey, Sean Fanello, Gordon Wetzstein, Jun-Yan Zhu, Christian Theobalt, Maneesh Agrawala, Eli Shechtman, Dan B Goldman, and Michael Zollhöfer. State of the art on neural rendering, 2020.

[23] Ayush Tewari, Michael Zollöfer, Hyeongwoo Kim, Pablo Garrido, Florian Bernard, Patrick Perez, and Theobalt Christian. MoFA: Model-based Deep Convolutional Face Autoencoder for Unsupervised Monocular Reconstruction. 2017.

[24] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Trans. Graph.*, 38(4), 2019.

[25] Luan Tran and Xiaoming Liu. Nonlinear 3d face morphable model. June 2018.

[26] Shih-En Wei, Jason Saragih, Tomas Simon, Adam W. Harley, Stephen Lombardi, Michal Perdoch, Alexander Hy-

pes, Dawei Wang, Hernan Badino, and Yaser Sheikh. Vr facial animation via multiview image translation. *TOG*, 38(4), 2019.

[27] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Ronen Basri, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. 2020.

[28] Yuxiang Zhou, Jiankang Deng, Irene Kotsia, and Stefanos Zafeiriou. Dense 3d face decoding over 2500fps: Joint texture & shape convolutional mesh decoders. June 2019.