Supplementary material for "Locally Adaptive Neural 3D Morphable Models"

Michail Tarasiou Rolandos Alexandros Potamias Eimear O'Sullivan
Stylianos Ploumpis Stefanos Zafeiriou
Imperial College London
{michail.tarasiou10,r.potamias19,e.o-sullivan,s.ploumpis,s.zafeiriou}@imperial.ac.uk

1. Quads-based data

To effectively learn a model capable of disentangled human head identity manipulations, a large scale dataset of 3D heads with a neutral expression is required. For this reason, 6k high quality head meshes from the original UHM dataset [4] were registered with a new mesh template consisting of 12k vertices, illustrated in Fig.1. While many previous models have employed triangle-based template meshes, here we opt for a quad-based mesh template. Quad meshes are often preferred for rigging and animation purposes as they are generally easier to rig and deform more predictably and smoothly than their triangle-based counterparts. Quads can also be easily subdivided into smaller quads, producing smoother results than triangle based meshes, where artifacts can occur upon subdivision. Given the applicability of this work for the creation and manipulation of 3D assets, a quad-based template is an intuitive choice.

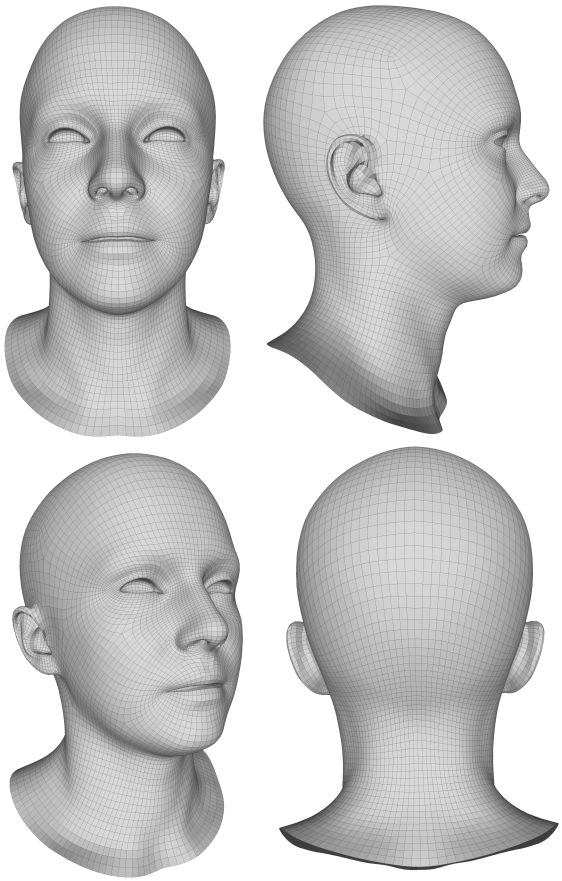

Figure 1. **Quad-based template** for UHM12k data.

2. Aditional experiments

2.1. Ablation on LAMM hyperparameters

In Table 1 we present an ablation study on the design of the most important parameters of our architecture. Here, we only consider the Transformer backbone. We further discuss how these choices affect the performance of architectures based on the MLPMixer when differences between the two are significant. As discussed in section 4 of the main manuscript, our architecture consists of five encoder and three decoder layers with 512 feature dimension. For the UHM12k data we provide 11 distinct regions to split our input data.

During our search over a suitable **feature dimension** we have observed almost flat performance for values > 512, while there some drop is observed for smaller feature dimensions. Our architecture's complexity is linear to the feature size, so we select the smallest value that does not lead to significant performance degradation, which is 512. The MLPMixer had a similar response to this parameter and the same value was used there as well.

The **features multiplier** refers to the ratio of the inner feature dimension for the Transformer feedforward layer (within a Transformer layer, after self-attention) to the architecture's feature dimension. Multipliers < 1 indicate a bottleneck design for the feedforward layer while values > 1 suggest an inverted bottleneck design. Our Transformer backbone appears to benefit significantly from a large inverted bottleneck design with a $\times 4$ multiplier. MLPMixers behave fundamentally different with regards to this hyperparameter as performance is very similar among all exam-

Ablation	Settings		Mean Error
Feature size	256		14.53
	512		**9.08**
	1024		9.04
Features Multiplier	0.5		14.02
	1		11.97
	2		10.45
	4		**9.08**
Depth	Encoder	Decoder	
	3	3	10.50
	4	4	9.10
	5	3	**9.08**
	3	5	9.34
	5	5	8.48
Number of Regions	29		8.96
	17		9.05
	13		9.07
	11		**9.08**
Loss	$\|\hat{\mathcal{V}} - \mathcal{V}\|$		12.14
	L_{enc}		10.52
	L_{dec}		10.13
	$L_{enc} + L_{dec}$		**9.08**
	$\times 10$ learn rate		49.39

Table 1. **Ablation on hyperparameters for LAMM Transformer architecture**. We present mean per-vertex Euclidean distance error (mm $\times 10^{-2}$) in UHM12k evaluation set. Bold letters indicate the value for each hyperparameter that was used in the final design. Each ablation shows performance differences when modifying the values of respective hyperparameter.

ined values. We choose a value of $\times 0.5$ for the MLPMixer. In addition to the key and query parameters which are not used in the MLPMixer, using a small multiplier value here accounts for the significant reduced number of parameters in the MLPMixer compared to Transformer backbones as shown in Table 3.

An ablation over **network depth** suggests an increase in performance with depth. Using fewer than eight layers combined for the encoder and decoder, was found suboptimal. More layers lead to increased performances, but these were not used because of the additional computation cost. There was not significant difference in how the number of layers was split among the encoder and decoder modules. For this reason we opted for an assymetric design, reducing the size of our decoder as it is this module that will be subsequently used after training. Thus, our model includes five encoder and three decoder layers.

Our initial design for the head regions involved 29 distinct areas selected by a 3D artist. We further proceeded with merging neighbouring regions to obtain splits in 17, 13 and 11 regions, all of which are illustrated in Fig.2. Ablating over these, we observe that the **number of regions** does not affect our model's performance in dimensionality reduction. Similar results were obtained for the MLPMixer. Since the Transformer complexity in quadratic to that number (linear for MLPMixer) we used 11 regions in our design.

Finally, we do an ablation on the **loss used during training**. We observe that using a single loss at the level of outputs ($||\hat{\mathcal{V}} - \mathcal{V}||_1$) leads to some performance degradation. Replacing this with the multilayer component for the encoder L_{enc} leads to some improvements, however, the effect of L_{dec} is more significant. Using both components $L_{enc} + L_{dec}$ leads to best performance overall and is chosen thereafter. For an eight layer architecture (five encoder and three decoder layers) our multilayer loss consists of ten different components (encoder tokens + $\times 5$ encoder layers + learned decoder tokens + $\times 3$ decoder layers) all of which are used with $\lambda = 1$. To assess whether performance gains can simply be attributed to an increased combined learning rate, we train with only one loss component at the level of outputs but now use $\times 10$ our learning rate to 10^{-3}. We observe that training with this value leads to significant performance degradation suggesting that the benefits of multilayer loss can not be simply be attributed to increased learning rate from adding the supervision signals from the $\times 10$ loss components.

2.2. Additional results for 3D reconstruction

In addition to the 3D reconstruction results reported in Table 1 (main manuscript), here, in Table 2, we present additional results for all the datasets with a varying number of latent feature dimension. We can make the following observations. First, LAMM-MLPMixer is always found to outperform baselines significantly among all datasets and feature dimensions tested. LAMM-Transformer is best in UHM12k data and clearly best at UHM12k with expressions. However, it losses in UHM and is very close in Handy, when compared to SpiralNet++. PCA outperforms all GCN baselines, but not LAMM, at latent dimension 256. This is not the case for smaller values in UHM12k and Handy data, where Graph Neural Networks (GCNs) clearly outperform PCA. This observation is similar to results shown in previous studies [1, 5].

Finally, we show additional results using *region-PCA*. To obtain these results we split the latent space of PCA into evenly sized segments, using the same number of regions as our models, and used each segment of the latent code to control respective regions. It is observed that region-PCA is clearly outperformed by all other methods, with differences being more pronounced for small latent sizes. We believe there is a straightforward explanation for this effect. This issue arises from the fundamental principle that both human head and hands, as anatomical entities, exhibit global geometric patterns that are not confined to isolated regions but

Table 2. **Quantitative evaluation of 3D shape reconstruction** for models trained exclusively in dimensionality reduction (autoencoding). Presented values are per vertex mean Euclidean distances ($\times 10^{-2}$mm). For region-PCA we split the latent space is evenly split into segments, controlling the same regions used in our models.

	UHM12k				UHM12k+expr.		UHM		Handy			
	32	64	128	256	128	256	128	256	32	64	128	256
PCA	54.32	31.78	16.39	10.42	24.12	11.49	20.84	11.73	87.27	60.89	40.97	25.90
region-PCA	191.48	99.12	49.10	23.36	50.91	25.31	50.34	25.92	189.11	108.11	68.16	42.61
COMA [5]	56.5	35.30	16.23	13.11	26.80	14.20	20.70	15.40	89.50	60.07	41.60	26.20
SpiralNet [1]	52.57	29.55	15.69	11.82	18.92	14.77	22.34	13.88	85.66	58.20	40.32	26.75
SpilarNet++ [3]	52.00	29.62	15.60	11.53	16.74	14.28	21.60	13.55	84.45	57.75	39.83	26.72
LAMM-Transformer	**42.88**	**19.55**	**10.71**	9.08	**10.65**	**9.09**	21.72	13.70	84.34	57.22	40.50	26.31
LAMM-MLPMixer	46.59	23.46	10.88	**7.97**	12.32	9.51	**19.93**	**11.48**	**82.63**	**56.89**	**39.20**	**24.60**

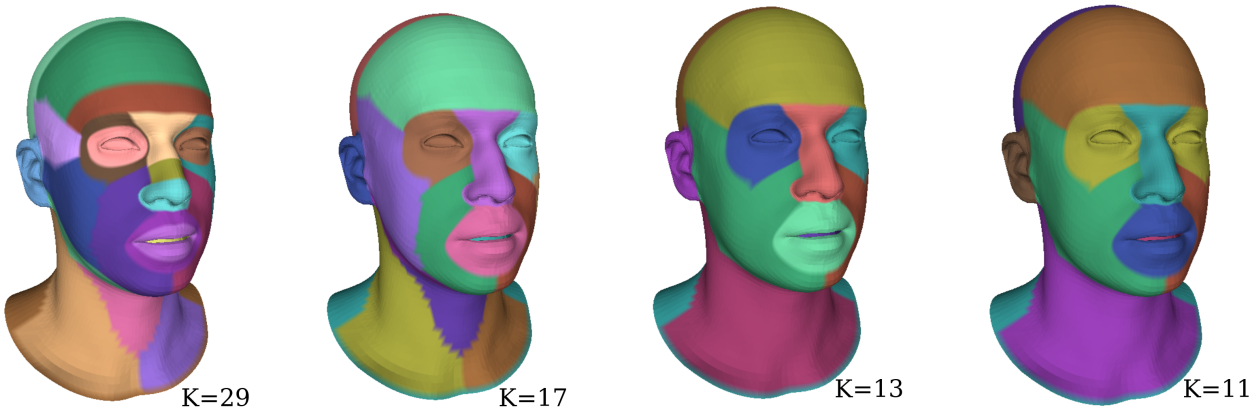

Figure 2. **Head regions** used in ablation experiments with UHM12k.

rather span across them, creating a cohesive and interdependent structure. For instance, the curvature of the cheek not only describes the cheek itself but also implies certain characteristics about the jawline, the nose structure, the forehead and is indicative of the head's overall size. When PCA is applied separately to each region, these global patterns are effectively segmented, and the holistic nature of the object's geometry is lost. As a result, certain key patterns that extend beyond the boundaries of individual regions have to be redundantly encoded in each of the split latent codes. This redundancy leads to an inefficient use of the latent space, as a portion of each region's code is occupied with replicating aspects of the global pattern that are shared across multiple regions. This explains why the performance degradation of region-PCA compared to global PCA is more pronounced for smaller latent sizes, as fewer latent dimensions remain available to describe unique regional patterns after a portion of each is allocated to replicating global patterns. We believe the same effect is observed in the case of small AE performance of the SD-VAE [2] explored in section 4 of the paper.

2.3. Mesh manipulation with non-bottleneck architecture

In LAMM, the decoder incorporates information from the *source* identity via bottleneck latent features $\mathbf{z}$. This bottleneck approach is pivotal in generating novel global instances of 3D objects. By fitting a probability distribution to these latent features, we can effectively draw new samples. However, for mesh manipulation, the use of a bottleneck design, which inherently restricts the information flow to the decoder, is not evident.

To investigate the impact of the bottleneck on manipulation efficacy, we conducted an experiment using a LAMM-Transformer model. In this setup, we bypassed the initialisation of decoder region tokens $\mathbf{y}_i^0$ and instead utilized the final encoder features $\mathbf{x}_i^L$, thereby enhancing the information relay to the decoder. While we noted some performance improvements, they were not as substantial as anticipated. Specifically, in the UHM12k training for manipulation, our LAMM$-$Transformer recorded error rates of 18.47 and 26.03 ($\times 10^{-2}$mm) for control and non-control vertices respectively, as detailed in Table 2 of the main manuscript. The non-bottleneck architecture yielded slightly better results, 16.55 and 23.38, respectively. Although this suggests a marginally superior model, the dif-

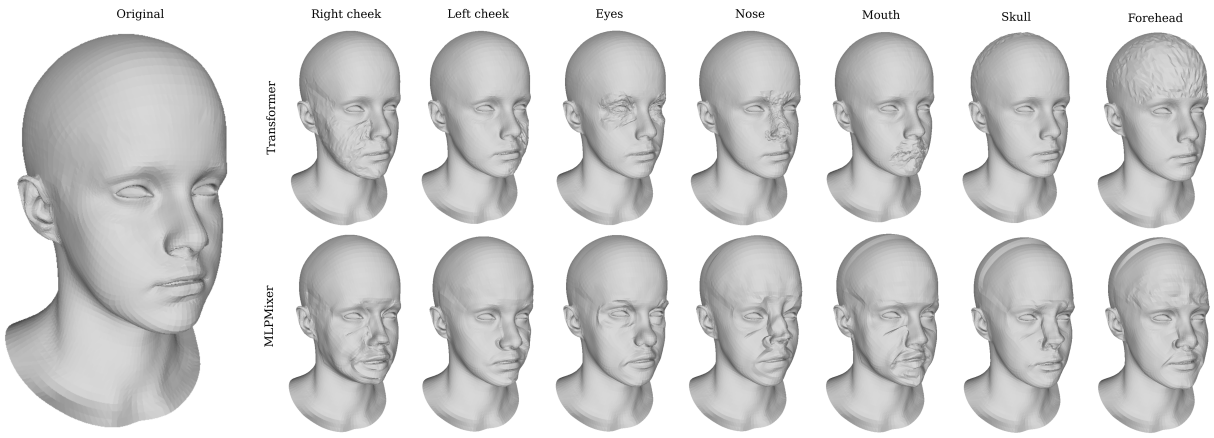

Figure 3. **Region disentanglement is an intrinsic property of LAMM**. Models used here are trained only as AEs (no manipulation training) with $L1$ supervision. (left) ground truth, (right) predictions with corrupted region tokens using Transformer and MLPMixer backbones. We corrupt the values of individual, learned region tokens $\mathbf{y}_i^{0'} = \mathbf{y}_i^0 + \mathbf{n}$ through some random noise $\mathbf{n}$. Learned region tokens are parameters of our model, thus, we expect output geometry to degrade. We observe that output geometry does indeed degrade, but only within the confines of region $\mathcal{R}_i$. This effect is more pronounced for the Transformer-based LAMM, where regions other than i remain practically unaffected.

ference is not pronounced. Furthermore, our bottleneck design offers a versatile architecture that integrates mesh compression and manipulation. This unified framework allows for both global sampling of 3D objects and local 3D parts, alongside manual manipulations. In contrast, the non-bottleneck architecture invariably necessitates an input mesh. However, the exploration of a non-bottleneck architecture presents an intriguing avenue for future research. The slight improvements in error rates observed here indicate potential for enhanced performance in mesh manipulation.

3. Disentanglement

3.1. Intrinsically disentangled representations

In sections 3 and 4 we discussed how LAMM can achieve disentangled editing without the need for loss components that enforce this behaviour and alluded to this being an intrinsic property of our architecture. Here we provide some additional evidence supporting this claim. As a reminder, we build our decoder inputs by appending K additional learned region tokens $\mathbf{y}_i^0 \in \mathbb{R}^D$ to the projected latent code $\mathbf{y}_0^0$. For AE only training with $\mathcal{V}^s = \mathcal{V}^t \Rightarrow \delta\mathcal{V}_{Ci} = 0$ we can omit the use of $f_{\delta i}$ as their output will be zero and will have no effect on generated geometry. Under this perspective, the learned tokens $\mathbf{y}_i^0$ can be viewed as defining a template which is progressively transformed to $\mathcal{V}^s$.

Here we assess region disentanglement prior to any manipulation training by examining the spatial effect of region tokens over generated geometry. In order to do so we corrupt their values and observe the effect on generated output. More specifically, we corrupt the values of individual, learned region tokens $\mathbf{y}_i^{0'} = \mathbf{y}_i^0 + \mathbf{n}$, $i > 0$ through some random Gaussian noise $\mathbf{n}$ (uniform noise leads to similar results). Learned tokens are parameters of our model, thus, by corrupting their values it is not expected that the model will continue generating reasonable outputs. Generated outputs with corrupted tokens are illustrated in Fig.3. We observe that corrupting individual tokens typically degrades the shape of the respective regions but, importantly, has little to no spillover effect in remaining regions. For the MLPMixer this is not entirely the case as there can be discontinuities among regions and obvious effects for neighbouring regions, however, most regions remain unaffected and the person's identity can still be discerned despite corruptions. For the Transformer-based model, corrupted geometry stays strictly within the confines of the region whose corresponding token was corrupted and no effect is observed in remaining regions. Note that in Fig.3 we use LAMM checkpoints pretrained only in autoencoding with a single $L1$ loss $||\hat{\mathcal{V}}^t - \mathcal{V}^t||_1$ at the level of outputs. In this manner we ensure that observed effects are not attributed to either the use of our multilayer loss or self-supervised manipulation training.

3.2. Additional results on disentanglement

In Fig. 4, we demonstrate the application of random sampling to the fingers of a hand mesh. Our model successfully generates disentangled samples that adhere to the statistical characteristics typical of human hands. We note that alterations in properties such as finger thickness or length predominantly influence the targeted region alone. Conversely, changes to attributes like the tip shape often impact adjacent fingers. This phenomenon is attributed to the inherent statistical correlations existing among an individual's fingers,

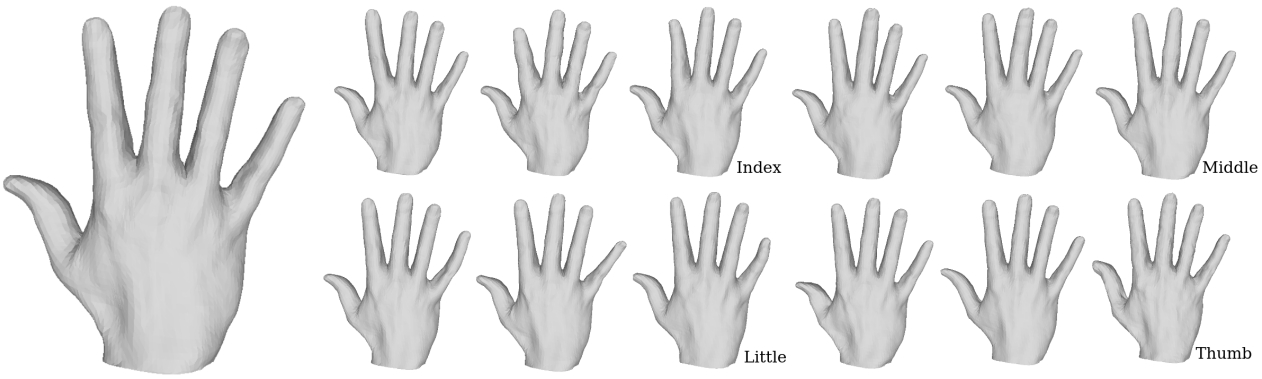

Figure 4. **Random generation of finger regions** for a sample in the Handy evaluation set. The model leads to disentangled sampling as manipulations tend to mostly affetc selected regions.

Table 3. **Model complexity and runtimes** for low (12k) and high (72k) resolution meshes. Runtimes are measured in commodity hardware, a i7-7820X CPU @ 3.60GHz intel CPU and a Nvidia GTX 1080ti GPU using single batch (average among 100 runs).

Model	12k Vertices				72k Vertices			
	latency (s)		#params. (M)		latency (s)		#params. (M)	
	cpu	gpu	decoder	token^{-1}	cpu	gpu	decoder	token^{-1}
COMA	0.5	0.015	3.88	-	3.80	0.089	19.38	-
SpiralNet++	0.075	0.003	4.27	-	0.58	0.016	19.67	-
Transformer (ours)	0.015	0.006	9.45	19.5	0.047	0.011	9.45	110
MLPMixer (ours)	0.015	0.006	0.8	19.5	0.045	0.011	0.8	110

highlighting the interplay between local and global modifications within the model's framework.

4. Model size and complexity

In Table 3 we provide an in-depth analysis of the model size and runtimes for LAMM and various GCN decoders used in our experiments. A notable distinction emerges between LAMM and the GCN baselines. Despite having a higher parameter count, LAMMs demonstrate superior performance in terms of inference speed, particularly on a CPU. This can be attributed to differences in the dimensions of the feature space of LAMM in comparison to GCNs.

Delving into the specifics, Table 3 highlights that the majority of LAMM model parameters reside in the inverse tokenization weights $\mathbf{W}^{out}$, amounting to 19.5 million for the 12k vertices dataset and 110 million for the 72k vertices dataset. As detailed in section 3, these parameters are utilized only once in computation, thus minimizing their computational load. The remaining parameters are distributed between the Transformer (9.45M) and the MLPMixer (0.8M) backbones. This disparity in parameter counts can be explained by two key factors: firstly, the MLPMixer replaces the transformer's keys and queries weights of size $D \times D$ (where D=512 represents the feature dimension) with smaller $K \times K$ weights (K=11 denotes the number of regions); secondly, the MLPMixer operates effectively with a reduced size multiplier $m = 0.5$ for the inner dimension of feedforward layers (size 256), in contrast to the Transformer's need for a larger multiplier $m = 4$ (size 2048). These parameters, while shared among features, are applied to a limited set of features. As discussed in sections 3 and 4 of the main manuscript, LAMM can model 3D head data using as few as 11 regions. This is the key to understanding LAMMs low computational requirements. Even in the case of the 72k vetrex meshes our internal representation length is just 12 (11 region features plus 1 latent code feature) in all layers. In comparison, the decoders of GCNs typically consist of four layers, each upscaling features by a factor of 4. Applied on 12k and 72k mesh inputs, this upscaling leads to feature lengths [47, 187, 745, 2979, 11916] and [281, 1124, 4496, 17982, 71926], respectively for each

layer. This increased number of features in all layers of GCNs is critical to their low inference speeds in CPUs. For 12k resolution meshes SpiralNet++ can run faster on our GPU than LAMM, however this changes with increasing resolution. For 72k vertex meshes LAMM is faster on a GPU and significantly faster on a CPU.

References

[1] Giorgos Bouritsas, Sergiy Bokhnyak, Stylianos Ploumpis, Michael Bronstein, and Stefanos Zafeiriou. Neural 3d morphable models: Spiral convolutional networks for 3d shape representation learning and generation. In *The IEEE International Conference on Computer Vision (ICCV)*, 2019. 2, 3

[2] Simone Foti, Bongjin Koo, Danail Stoyanov, and Matthew J. Clarkson. 3d shape variational autoencoder latent disentanglement via mini-batch feature swapping for bodies and faces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18730–18739, 2022. 3

[3] Shunwang Gong, Lei Chen, Michael Bronstein, and Stefanos Zafeiriou. Spiralnet++: A fast and highly efficient mesh convolution operator. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019. 3

[4] Stylianos Ploumpis, Evangelos Ververas, Eimear O'Sullivan, Stylianos Moschoglou, Haoyang Wang, Nick Pears, William AP Smith, Baris Gecer, and Stefanos Zafeiriou. Towards a complete 3d morphable model of the human head. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):4142–4160, 2020. 1

[5] Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J. Black. Generating 3D faces using convolutional mesh autoencoders. In *European Conference on Computer Vision (ECCV)*, pages 725–741, 2018. 2, 3