DOI: 10.1111/cgf.14760
EUROGRAPHICS 2023 / K. Myszkowski and M. Nießner
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 42 (2023), Number 2

Face Editing Using Part-Based Optimization of the Latent Space

Mohammad Amin Aliari[1], Andre Beauchamp[3], Tiberiu Popa[2] and Eric Paquette[1]

[1]Ecole de technologie supérieure, Montreal, Canada
[2]Concordia University, Montreal, Canada
[3]Ubisoft La Forge, Montreal, Canada

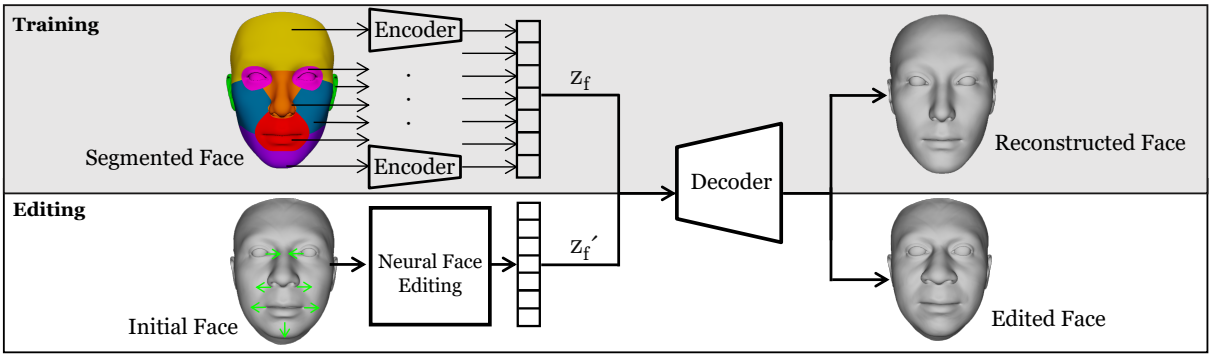

Figure 1: *Our 3D face model. Training: We feed each segment of the face to its part encoder. Then, we merge and pass the encoded representation to a decoder that reconstructs the face. Editing: Our approach modifies the latent vector of the face based on user constraints.*

Abstract

We propose an approach for interactive 3D face editing based on deep generative models. Most of the current face modeling methods rely on linear methods and cannot express complex and non-linear deformations. In contrast to 3D morphable face models based on Principal Component Analysis (PCA), we introduce a novel architecture based on variational autoencoders. Our architecture has multiple encoders (one for each part of the face, such as the nose and mouth) which feed a single decoder. As a result, each sub-vector of the latent vector represents one part. We train our model with a novel loss function that further disentangles the space based on different parts of the face. The output of the network is a whole 3D face. Hence, unlike part-based PCA methods, our model learns to merge the parts intrinsically and does not require an additional merging process. To achieve interactive face modeling, we optimize for the latent variables given vertex positional constraints provided by a user. To avoid unwanted global changes elsewhere on the face, we only optimize the subset of the latent vector that corresponds to the part of the face being modified. Our editing optimization converges in less than a second. Our results show that the proposed approach supports a broader range of editing constraints and generates more realistic 3D faces.

CCS Concepts

• ***Computing methodologies*** → ***Mesh models; Neural networks;***

1. Introduction

Face modeling and editing have been very active topics in computer vision and graphics. It has a wide range of applications in multiple contexts, such as video games, film, visual effects, and the metaverse. In the context of the metaverse and video game production, especially in open-world games, artists need to generate and edit new 3D meshed faces at a large scale, and this with a simple and intuitive interface.

Current commercial tools (e.g., ZBrush® [Max]) for manual 3D face modeling and editing require a lot of expertise to achieve the desired output. Our work focuses on developing an approach for generating new faces at a large scale while providing an intuitive editing capacity for the artists. This will have a considerable impact on the time spent by artists on the task.

Over the past 20 years, linear generative models [BV99] have been the dominant technology for face generation because of their

simplicity and efficiency. With such generative models, a new face is generated by a linear combination of an orthogonal basis. This basis is obtained by applying a statistical analysis (Principal Component Analysis – PCA) on a set of face scans. PCA-based linear models suffer from multiple drawbacks. Their control mechanism is not intuitive. Also, the generated faces are generally bound by the statistical prior space; If the parameters are varied a little, the generated faces are believable but rather similar to the ones in the dataset, and if the parameters are varied a lot, we get novel faces, but they are not necessarily realistic. In other words, these methods tend not to generalize very well. Furthermore, local face editing is not possible with most PCA-based methods because the orthogonal basis does not provide a semantically meaningful separation of different parts of the face. More recently, non-linear generative models [RBSB18] based on Deep Neural Networks (DNN) have emerged and achieved a better generalization capacity than PCA-based methods. However, it is challenging to balance the realism of the face with the requirement of allowing the generation of a large variety of faces. Moreover, providing an intuitive user interface for local face editing remains a big challenge.

In this work, we propose a novel approach for face modeling that enables users to generate a wide variety of realistic faces using a simple and intuitive user interface. We frame the problem as an optimization problem over the latent space of a graph-based variational autoencoder (VAE) [KW14] that incorporates both a set of part-based encoders as well as a global face decoder. This combination is key to allowing a large variety of faces as well as maintaining local user control. The part-based encoders allow for more flexibility in the generated result as well as local user control, while the global decoder ensures the realism of the result. This formulation also allows the user to perform direct vertex manipulation as an editing paradigm. Furthermore, our method runs at interactive rates, taking under one second to generate a new face after user interaction. Our novel contributions are summarized as:

- A graph convolutional VAE architecture with a contrastive loss function designed to disentangle the latent space into local parts;
- An approach to editing the face through an optimization of the latent variables enforcing the locality of the edit.

Compared to state-of-the-art methods, our approach has an improved generalization in contrast to PCA-based methods and a better locality of the editing compared to DNN methods.

2. Related Work

Classical 3D morphable face model (3DMM) methods as well as many recent ones [CWZ*14, BRZ*16, LBB*17, BRP*18, PWP*19, PVO*20, GRF*20, EST*20] use PCA to sample the distribution of the face models. The popularity of PCA-based methods is due to their simplicity and efficiency. User control is achieved by optimizing for the eigenvector weights given vertex-based constraints, typically resulting in a linear system of equations. In many cases this is done globally on the entire face face [CWZ*14, BRZ*16, LBB*17, BRP*18] resulting in a lack of local control. Local control is less important when the application is global face reconstruction, but it is a critical limitation when the application is face editing. To address this limitation, Ghafourzadeh et al. [GRF*20] propose a method that further decomposes the face into semantic parts allowing independent generation for each part followed by an ARAP-inspired reassembling of the parts into one coherent face mesh. Vertex-based editing was added to this part-based method [GFR*21]. Similarly to our approach, the user edits the face by moving mesh vertices. Alas, linear models such as PCA tend to perform poorly in the presence of large complex changes, and blending together local parts may result in uncanny effects.

More recent neural methods leverage the generative power of deep learning by replacing the PCA with either autoencoders [TGLX18, BWS*18, BBP*19] or Generative Adversarial Networks (GAN) [BBP*19, CBZ*19]. Abrevaya et al. [FA20] uses a fully connected network along with UV representation of the vertex positions. Image representation of the geometry is particularly useful as many state-of-the-art convolutional neural network (CNN) models can be used with this type of data. However, reconstructing the final mesh from a UV mesh has its own problems. There are areas around the lips or the eyes where there is no data. As a result, those vertices would collapse to (0, 0, 0) coordinates. That is why Abrevaya et al. [FA20] use the flattened representation as input of the GAN model while training the discriminator with the geometry map. Li et al. [LBZ*20] also use a UV representation and a hybrid method to address reconstruction issues. They use a combined linear and non-linear method. The face area, where there are reliable pixel values, is sampled directly, while the rest of the geometry is morphed using linear 3DMM deformation modes.

Vesdapunt et al. [VRWW20] uses modeling bones to represent the face. This representation benefits from the lower count of parameters due to the compactness of bone data. The JNR model learns the skinning weights to reconstruct the 3D faces. The predicted modeling bones can be used for face modeling. However, this approach needs a base model with modeling bones and initial hand-painted skinning. Also, the bones are directly transformed to deform the face, and neither higher-level control nor non-linear transformations exist.

Ranjan et al. [RBSB18] use graph convolution to build their CoMA model. A novel mesh sampling operation is also used to capture details of the face at different levels. It also helps reduce the convolution input dimension so the model can converge more easily. As a result, CoMA and other works inspired by CoMA [TGLX18, LLLY19, YLL*19] can outperform linear models in face reconstruction tasks. Bagautdinov et al. [BWS*18] use a variational autoencoder (VAE) architecture to model faces, but there are several fundamental differences, both conceptual and technical, between their work and ours. Their goal is high-fidelity 3D face reconstruction from images and video. As such, their VAE design matches this goal by globally reconstructing the faces using several latent spaces corresponding to a range from coarse to dense levels of detail. In contrast, our goal is novel face synthesis through local editing, and, as such, we employ an architecture where we segment the face and use part-based encoders and a global decoder.

While the methods presented so far use triangular meshes as a surface representation, alternative representations like voxel based exist but are primarily suitable for lower-level detail or a specific category of 3D meshes. For instance, Guan et al. [GJvK20] use voxelization to reconstruct and generate 3D desk and chair shapes.

Most of the work in DNN for faces has been about encoding the identity and expression of the face rather than enabling editing operators. Tan et al. [TGLX18] use a set of expression labels with a conditional-VAE to achieve control. However, describing face modeling with labels is not feasible as we would need a much lower level of control over the face. Yang et al. [YZW*20] propose a method that can reconstruct a rigged model from a single photograph, while Wang et al. [WCY*22] and Bao et al. [BLC*21] propose methods that can reconstruct a face model from RGB-D data. Very recently, Jung et al. [JJK*22] proposed a DNN model to generate 3D caricatures. While this model works remarkably well for caricatures, it exhibits a global deformation behavior during editing that makes it difficult to control.

We propose an approach that outperforms PCA-based 3DMM methods in terms of generalization from the faces in the dataset. Furthermore, our approach outperforms DNN methods in terms of application to face editing, in particular providing a local editing paradigm.

3. Method

We present a DNN architecture to learn a compact latent space representation for 3D meshes of faces. The two main design objectives of our new architecture are to be able to disentangle different parts of the face and to allow local editing of the face mesh. For this, our key idea is to decompose the faces into different semantic parts and employ an autoencoder architecture where we feed each part to a separate encoder that produces a semantic latent vector for that part of the face. A single decoder is then used to aggregate all the latent vectors and produce the final mesh. The main intuition of having a separate embedding for each part of the face is to enforce local face editing by design. The use of a single decoder, that has access to all the embedding vectors, produces a realistic and consistent mesh. Our novel loss function enforces the disentanglement of the latent variables so that each has an influence on a localized region of the face. After training on a dataset of faces, we get a tailored latent space that enables multiple applications. It can be used to easily generate random faces with meaningful variations of facial characteristics, enabling character designers to generate a large set of facial assets. We also demonstrate that the latent space is very powerful in enabling the editing of the face through an optimization of the latent variables from user constraints in terms of dragging and dropping vertices. This provides a very intuitive interaction paradigm, and our latent space optimization outputs meaningful deformations of the face. This section introduces the face generator network, its training procedure, and our neural face editing approach (Figure 1).

3.1. 3D Face Generator

The first role of the generator is to learn a latent representation of the face that leverages the generalization capacity of the generative model to create new faces. For this goal, we introduce a network that encodes the face into a low-dimensional data representation. The second role of the generator is to learn a disentangled latent space where each group of latent variables is related to one specific part of the face. This encourages local and independent changes when the latent vector is modified.

Input and Output Data. The input of our network is a 3D face model represented as a 3D mesh. As we primarily use graph convolution operators, we represent the input as the canonical graph induced by the 3D mesh. For this reason, all of the faces in the dataset follow graph representation: $\mathcal{F} = (\mathcal{V}, \mathbf{A})$ with matrix $\mathcal{V} \in \mathbb{R}^{n\times 3}$, n vertices , and an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ showing the edge connections. Similarly to other methods [BV99,RBSB18,LLLY19, FA20,GRF*20], we require that all faces be wrapped with one base head; they share an identical mesh topology (same triangles and vertex connectivity). In addition, each face is segmented into seven different parts: forehead, eyes, ears, nose, cheeks, mouth, and chin. Each part $\mathcal{P}_i$ has n_i vertices $\mathcal{V}_i \in \mathbb{R}^{n_i\times 3}$. The segmentation is user-provided and shared by all faces. The output is the generated face with the same mesh topology and dimensions as the input face.

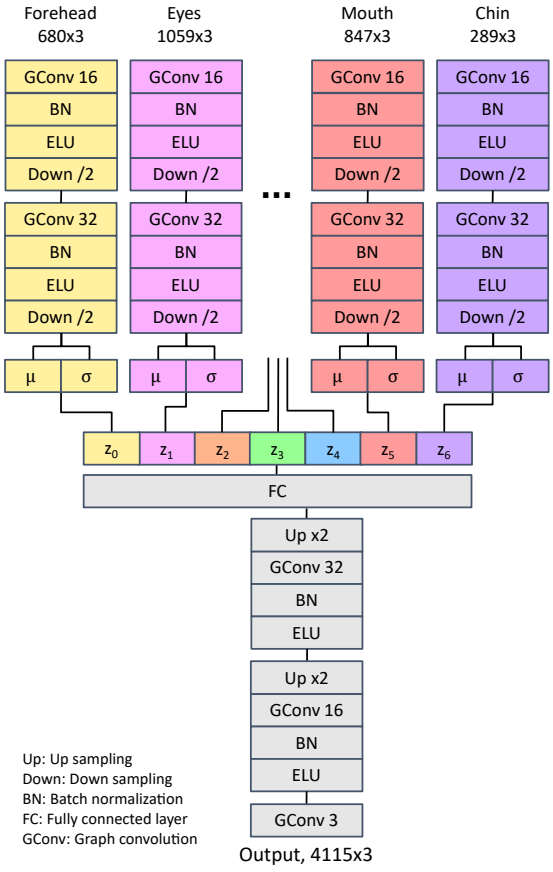

Figure 2: *3D Face Generator: Network architecture*

Network Architecture. We choose variational autoencoders [KW14] as our generative model. As in the work of Ranjan et al. [RBSB18], we use fast spectral convolutions [DBV16] along with their mesh sampling operation. The mentioned graph convolution uses kernel g_θ, which is parameterized with Chebyshev polynomials

$$g_\theta(L) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L}), \quad (1)$$

where K is the order of the polynomial, $\tilde{L}$ is the scaled Laplacian, $\theta \in \mathbb{R}^K$ is Chebyshev coefficients, and $T_k \in \mathbb{R}^{n \times n}$ is the Chebyshev polynomial order k that can be obtained recursively. The definition of spectral convolution then becomes:

$$y_j = \sum_{i=0}^{F_{in}} g_{\theta_{i,j}}(L) x_i \in \mathbb{R}^n, \tag{2}$$

where $x \in \mathbb{R}^{n \times F_{in}}$ is the input, $F_{in} = 3$ is the number of input feature assuming it represents 3D vertex positions, and y_j is each of $y \in \mathbb{R}^{n \times F_{out}}$ features. For more detail, we can refer to the spectral convolution papers [DBV16,RBSB18]. To achieve latent space disentanglement, we design an asymmetric model where each face part is fed to a separate encoder called *part encoder*. Specifically, each encoder uses two Chebyshev convolutional filters with $K = 6$ polynomials and dimensions of 16 and 32, respectively. Then, we apply the ELU activation function [CUH16] to each filter output. Furthermore, we place a down-sampling layer of factor two between each filter. Next, similar to traditional VAEs, we use two fully connected layers in parallel to transform the output to two 8-dimensional vectors $\boldsymbol{\mu}_i$, and $\boldsymbol{\sigma}_i$, which are the mean and standard deviation of the part $\mathcal{P}_i$. The details of each part encoder are shown in Figure 2. The initial vertex count of each part affects the dimensions of the following layers, and this is the difference between each part encoder. Finally, we sample the latent vector $\mathbf{z}_i \in \mathbb{R}^8$ from $\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2)$. As for the last step to encode the face, we concatenate all the latent vectors into the final latent vector $\mathbf{z}_f \in \mathbb{R}^{56}$. We feed $\mathbf{z}_f$ to our decoder block to reconstruct the face. This block is built with a similar set of components. It starts with a fully connected layer that maps $\mathbf{z}_f$ to the appropriate dimension for the convolutional filters. We use three filters of dimensions 32, 16, and 3. Also, we use ELU and up-sampling layers of factor two between the filters. The decoder output $D(\mathbf{z}_f) \in \mathbb{R}^{n \times 3}$ is the final face with the same dimension as the number of vertices in the initial face. Consequently, it will learn to merge the parts into a whole face. The decoder architecture is also shown in Figure 2.

Loss Function. Our loss function is composed of three terms: a reconstruction loss, a KullbackLeibler (KL) divergence loss [KW14], and a contrastive loss:

$$l = l_{Rec} + w_{kld} l_{KL} + w_{con} l_C. \tag{3}$$

We use an L_1 distance between the vertex positions of the ground-truth and decoded meshes.

$$l_{Rec} = \left\| \mathcal{F} - D(\mathbf{z}_f) \right\|_1 \tag{4}$$

The KL loss enforces a normal-like distribution for the latent vector $Q(z_i|\mathcal{F})$. The KL loss weight w_{kld} is 1e-3, and the number of face parts is $N_p = 7$.

$$l_{KL} = \sum_{i=0}^{N_p} KL(\mathcal{N}(0,1) \| Q(z_i|\mathcal{F})) \tag{5}$$

Inspired by the work of Deng et al. [DYC*20] in disentangling the latent space for human-face image generation, our third term to reinforces latent space disentanglement and ensures that each part of the latent vector only affects the assigned part of the face:

$$l_C = \sum_{i=0}^{N_p} \left\| (\mathcal{F}_i' - \mathcal{F}) \odot \delta_{\notin \mathcal{P}_i} \right\|_1. \tag{6}$$

For one part $\mathcal{P}_i$, we start with latent vector $\mathbf{z}_f$ and replace the part $\mathbf{z}_i$ with a randomly sampled $\mathbf{z}_i'$, while leaving the rest of $\mathbf{z}_f$ as before. We randomly sample with a uniform distribution $\mathcal{U}(-10, 10)$ to ensure we "push and pull" $\mathbf{z}_f$ far enough to ensure that even large deformations of the face remain local. The result is a new final latent vector $\mathbf{z}_f{}'$ that only differs in $\mathbf{z}_i$. We calculate the L_1 distance between the vertex positions of the new face $\mathcal{F}_i'$ generated by decoding $\mathbf{z}_f{}'$ and the vertex positions of the face $\mathcal{F}$ decoded from the original $\mathbf{z}_f$. However, we ignore the current part vertices $\mathcal{V}_i$ to only penalize changes outside of part $\mathcal{P}_i$ by multiplying (Hadamard product) with $\delta_{\notin \mathcal{P}_i}$ defined that entries matching the part are zero and others are one. We sum up this L_1 distance for each of the N_p parts.

3.2. Training Procedure

We train the model for 70 epochs using the Adam optimizer [KB15] and a batch size of 16. Like CoMA [RBSB18], we set the learning rate to 8e-4 and decay that rate by 0.99 every epoch. As mentioned before, we set w_{kld} to 1e-3. In addition, we set the w_{con} to 1e-4 at the beginning and gradually increase it by 1.1 every epoch. This helps the training process in the early stages by enabling the optimizer to converge to a good point in terms of reconstruction quality (Equation 4), plus having a more normal-like distribution (Equation 5). After that, we apply more weight to the l_C loss to fortify the latent space disentanglement (Equation 6). We tested with other numbers of epochs (up to 100), and 70 provided a good compromise between generalization and overfitting. We describe the details of our datasets in Section 4.1.

3.3. Neural Face Editing and Generation

We can now use the trained face generator to create new faces. The user may directly manipulate each latent variable and observe the changes in the output of the network. However, operating directly on the latent space is often unintuitive as there is no clear semantic meaning behind each latent variable. For a more intuitive editing, we propose a flexible workflow where the user controls vertex positions, and latent variables are automatically adjusted by our approach to reach the desired vertex movement (Figure 3).

Our main goal is to ensure the locality of the edits. As such, at any given time, the user either manipulates one vertex (e.g., the tip of the nose) or a pair of symmetric vertices (e.g., corners of the mouth). At every stage, we aim to fulfill the 3D deformation prescribed by the user, maintain the locality of the edit, and preserve past edits. We formalize this as an optimization problem and use the Adam optimizer [KB15] to find the best latent values. Given our editing workflow, the user controls one part at a time, which keeps the deformations local and helps the optimizer work on a smaller subspace and converge in a limited number of runs. After each modification, we start from the current face latent representation and gradually optimize that latent vector to achieve the next edit. To do this, we define the editing loss:

$$l_E = \sum_{i=0}^{N_v} \left\| v_i' - c_i \right\|_2 + w_{reg} \left\| \left(\mathcal{F} - D(\mathbf{z}_f{}') \right) \odot \delta_{\notin V_s} \right\|_1. \tag{7}$$

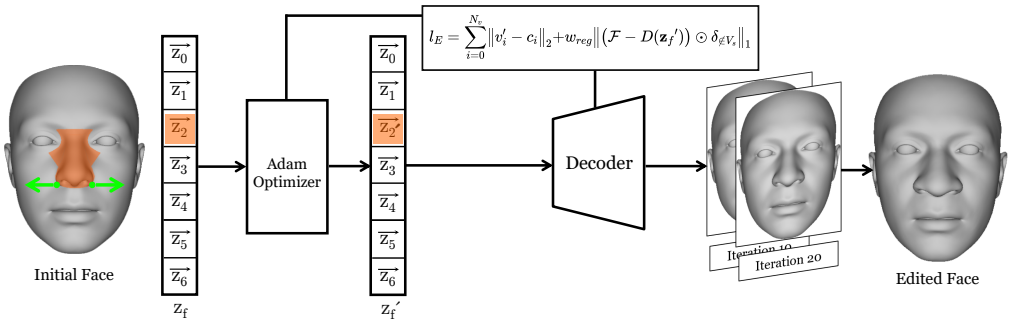

Figure 3: *Neural face editing workflow example. The user input can be the movement of the selected vertices. For example, moving the green spheres in the direction of green arrows. Then to apply the change, we start from the initial face representation $\mathbf{z}_f$, and optimize for l_E. The updated latent vector $\mathbf{z}_f{}'$ is only different in the part latent sub-vector $\mathbf{z}_2$. As shown above, the edited face is the decoded result of the optimizer after a number of iterations. It has a wider nose since the user has moved the desired nose vertices apart.*

The first term is the average Euclidean distance between the edited vertices v_i' and the constraints c_i set by the user. N_v corresponds to the number of vertices V_s selected for editing (either 1 or 2 in our workflow). The second term is regularization. We use this term to avoid deviating too much from the current face. We measure the average per-vertex L_1 distance between the two faces, excluding the selected vertices by defining $\delta_{\notin V_s}$ such that entries corresponding V_s are zero and others are one. In addition, w_{reg} is used to adjust the regularization term effect (we use $w_{reg} = 3$). To further fulfill local face editing, we only update the required part of the latent vector by identifying which segment of the face is being modified and will not include the rest of the vector as parameters of the optimizer. For example, if the selected vertices belong to part $\mathcal{P}_i$, only the related part of the latent $\mathbf{z}_i$ is modified. Therefore, the rest of the latent vector would remain the same.

Considering that we aim to integrate our solution into an interactive application, we want to run the optimizer only for a limited number of iterations. We use the Adam optimizer [KB15] to find a solution quickly. Figure 4 shows the decrease of the loss through 500 iterations. We see that with learning rates of 1e-2 and 1e-3 (taking small steps), the learning is predictable yet very slow. The learning rate of 1e-1 (taking larger steps) still makes reasonably stable progression and has the advantage of converging much faster, which is important in our interactive editing context. We can see that the optimizer makes significant progress toward the solution in the first tens of iterations and reaches a plateau around the 100th step, where the per iteration progress becomes negligible. Given our current target hardware (RTX3070), we choose to run the optimizer for 50 iterations with a learning rate of 1e-1. As a result, the process takes about 500 ms and is within the interactive range.

Random Face Generation. The trained network can be used as a 3D face generator by sampling from the learned latent space of the VAE. Our model consistently outputs plausible faces where the latent space is sampled from a normal distribution $\mathcal{N}(0, 1)$. This is in contrast to methods like the one of Ghafourzadeh et al. [GFR*21] that might generate unacceptable faces and need an additional verification step to decline them.

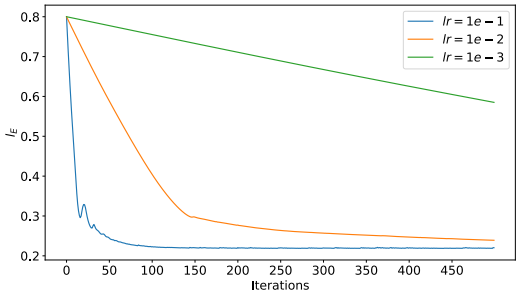

Figure 4: *Graph of the l_E loss history (Equation 7), with different learning rates of the optimizer [KB15] for vertex-based editing.*

4. Results and Experiments

In this section, we first cover the details of our datasets. Then, we present our editing and random face generation results. Afterward, we compare our approach with three state-of-the-art methods showcasing how our approach is better suited for local face modeling.

4.1. Datasets

We evaluate our approach on two different datasets. The first dataset is composed of 892 scanned faces that share the same mesh topology. This is our primary dataset used to show the results in this paper (except Figure 8, which uses our second dataset, and one clip in the accompanying video). We segment all the faces into the parts shown in Figure 1. Similar to the work of Ghafourzadeh et al. [GRF*20], the segmentation is an offline process done manually and based on artists' feedback. We use FaceWarehouse [CWZ*14] as our second dataset composed of 150 heads that also have the same mesh topology. The purpose of using this dataset is to evaluate the generalization capacity of our approach when trained on a relatively small dataset. Moreover, training independently on each of these two datasets demonstrated that our approach is not dependent on a specific one. In addition, we train the CoMA [RBSB18] model with the FaceWarehouse dataset to compare it with ours. We use 80% of a dataset for the training set and the rest for validation.

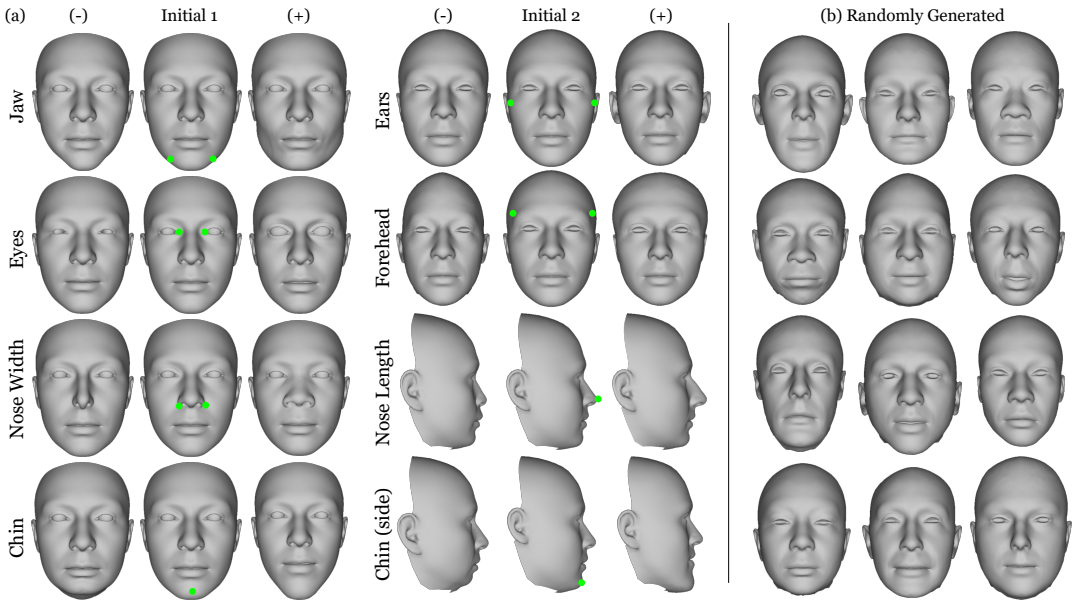

Figure 5: *(a) Vertex editing: Results of various face operations. For example, for the top left, we moved the two green vertices on the sides of the jaw outward to increase its width and inward for the opposite effect. (b) Examples of using our method to generate faces randomly.*

4.2. Results

In Figure 5 (a), we show some examples of vertex editing (Section 3.3). We modify two different initial faces (initial 1 and 2) with different operations. It can be the movement of one vertex or two vertices. In the case of two control vertices, the user selects one vertex , and our system automatically selects the symmetric one. Similarly, the user moves one vertex, and our system moves the second one in a symmetric manner. In each example, we modify one part of the face and, thus, one part of the latent vector. We see that the resulting deformations on the face are local. Moreover, we observe that even when moving only one or two vertices, we can effectively edit the face and benefit from our editing-friendly latent space and workflow. In addition, we have included a number of clips in the supplementary materials. In the videos, we edit an initial face with a latent vector of zero. We see that the deformation from each vertex movement remains effective, realistic, and predictable throughout the sequence of consecutive edits on the same face. In the editing clips, we outline a few vertices with red spheres. This is done to make it easier for the user to select and track the position of vertices. The user can still select and move any of the other vertices. Additionally, in Figure 5 (b), we demonstrate some of faces that are the randomly generated from $\mathcal{N}(0,1)$ using our 3D face generator (Section 3.1). We can observe the broad range of facial characteristics that our generator can achieve and mix together.

4.3. Ablation Studies

We perform two ablation studies. First, we cumulatively remove key parts one after the other, and second, we remove them one by one. These are two complementary analysis, and each helps us gain better insight into different elements of our approach. We evaluate the locality of the edits in each step with the following two quantitative measures: (i) The average vertex displacement inside the edited part $\overline{\delta}_{in}$. (ii) The average vertex displacement outside of the mentioned part $\overline{\delta}_{out}$ (vertices located in other parts of the face). $\overline{\delta}_{in}$ shows how much the part has changed locally, and $\overline{\delta}_{out}$ is essential for determining if the changes did not cause unwanted changes in other areas. In Table 1, we measure the average *local* vertex displacement (inside the part, $\overline{\delta}_{in}$) compared to the average *global* vertex displacement (outside the edited part, $\overline{\delta}_{out}$) which we want to minimize to maintain the locality of the edit.

Cumulative. Figure 6 shows the effect of different elements of our approach: the regularization term (Equation 7), the local latent optimization, and the contrastive loss (Equation 6). In (a), we see our current approach, where all of the elements are present. In (b), we remove the regularization term from the optimizer. This change increases the unwanted global deformations (see the mouth and $\overline{\delta}_{out}$ in Table 1 which increases from from 0.18 mm to 0.34 mm). In (c), on top of removing the regularization term, we also update the whole latent vector (global latent optimization) instead of only updating the part's sub-vector (local latent optimization). By comparing (b) and (c), we see how effectively our local latent optimization works. It performs much better than the global latent optimization because it mitigates most of the global deformations visible in (c). Finally, it is shown in (d) that when we exclude all the previous factors and also train our model without the contrastive loss, the deformations become even less local. For both (c) and (d), we see in Table 1 that the global deformation $\overline{\delta}_{out}$ drastically increases (0.18 mm to 4.61 mm and 6.92 mm).

One by One. Figure 7 reflects the individual absence of each pivotal part of the method when editing the ears of the subject. In (a), we see our current approach. In (b), we again remove the regular-

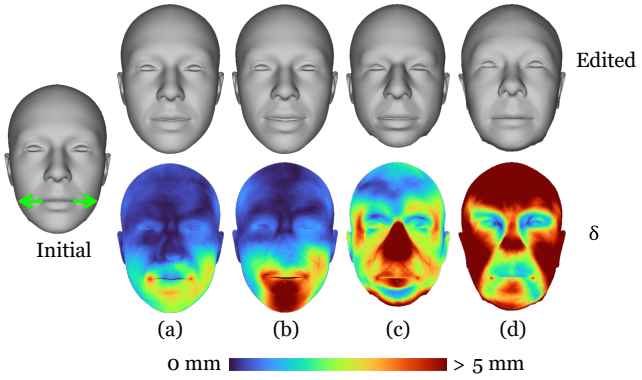

Figure 6: *Ablation study: showing the effect of each element of our approach. (a) Our current approach. (b) No regularization term (Equation 7). (c) Like (b) but global latent optimization instead of local. (d) Like (c) but the model is trained without the contrastive loss (Equation 6). δ is the vertex displacement.*

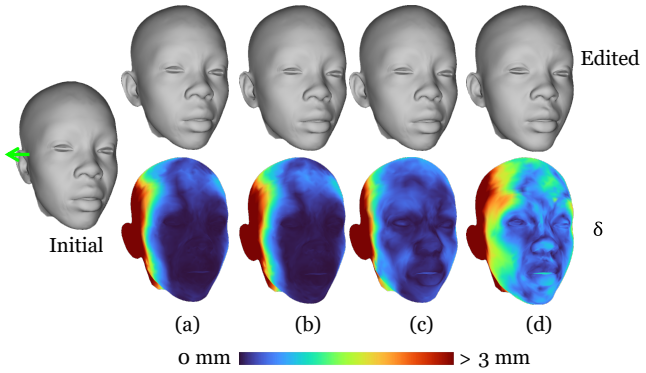

Figure 7: *Ablation study: showing the effect of each element of our approach. (a) Our current approach. (b) No regularization term (Equation 7). (c) Global optimization instead of local. (d) As (a) but the model is trained without the contrastive loss (Equation 6).sc δ is vertex displacement.*

ization term. Depending on the edit, the regularization term sometimes makes a big difference (as we saw in Figure 6 (b)), while here we see that our model is already benefiting from a local latent space, and removing the regularizer makes little difference (only a slight increase of global deformation $\bar{\delta}_{out}$ from 0.19 mm to 0.22 mm, see Table 1). In (c), we only replace the local optimization with a global optimization (optimizing the whole latent vector instead of the part sub-vector). Other elements remain the same as in (a). We observe that there are more unwanted global deformations. Furthermore, in terms of speed, the same optimization task takes 37% more time to compute. This is because we are optimizing the whole latent vector. Hence we have more variables to tune. Finally, in (d), we replace the model in (a) with a model that was trained without the contrastive loss. The unwanted deformations become even more problematic. Similarly to what we observed for the cumulative ablation study, for both (c) and (d), the global deformation increases further more, even if in this case we make the changes one by one ($\bar{\delta}_{out}$ increases from 0.19 mm to 0.26 mm and 0.48 mm).

Figure, Part		$\bar{\delta}_{in}$	$\bar{\delta}_{out}$
Fig. 6, Mouth	(a)	4.31 mm	0.18 mm
	(b)	5.00 mm	0.34 mm
	(c)	3.54 mm	4.61 mm
	(d)	7.60 mm	6.92 mm
Fig. 7, Ears	(a)	6.41 mm	0.19 mm
	(b)	6.75 mm	0.22 mm
	(c)	6.56 mm	0.26 mm
	(d)	7.17 mm	0.48 mm

Table 1: *Quantitative results of ablation studies. Average vertex displacement inside ($\bar{\delta}_{in}$) and outside ($\bar{\delta}_{out}$) the edited part.*

4.4. Comparisons

Comparison with CoMA [RBSB18]. We compare our method with CoMA [RBSB18] as it also uses graph convolution and mesh sampling. We compare it to the VAE version of CoMA since, as our approach, it has a better interpolation space. This is because the VAE version is not only focused on reconstruction tasks in contrast to the autoencoder version. In order to conduct different comparisons, we first need to train both models with the same dataset. FaceWarehouse dataset [CWZ*14] because the CoMA dataset [RBSB18] has an insufficient number of subjects in the neutral pose, and a reasonable number of neutral poses is necessary to train a neural network for editing. We think this is a fair comparison since neither model has been designed around the FaceWarehouse dataset. We first compare the reconstruction capabilities of the models. Our model has an average error of 1.40 mm on the training dataset (80% of the dataset) and 1.77 mm on the unseen samples (20% of the dataset). In the case of CoMA, it is 2.10 mm and 2.28 mm respectively. We can observe that even though our model's main task is not reconstruction, it has a better performance than CoMA. Next, we want to check each model's output in various face editing scenarios. We use our vertex editing approach (Section 3.3) with both our model and CoMA. As shown in Figure 8, we edited different areas of the face with both models. Our model has a much more editing-friendly latent space and results in localized changes on the face. For example, we can see that CoMA [RBSB18] introduces a lot of non-local movement: on the forehead when editing the mouth, toward the chin/jaw when editing the nose, and on the ears when editing the forehead. Furthermore, this test shows that using a regularization term with vertex-editing (Section 3.3) is not enough to prevent global changes, as apparent in the results of CoMA [RBSB18]. Table 2 verifies this with quantitative measures: the vertex displacement outside of the edited part ($\bar{\delta}_{out}$) is roughly an order of magnitude larger for CoMA [RBSB18]. Therefore, the complexity introduced

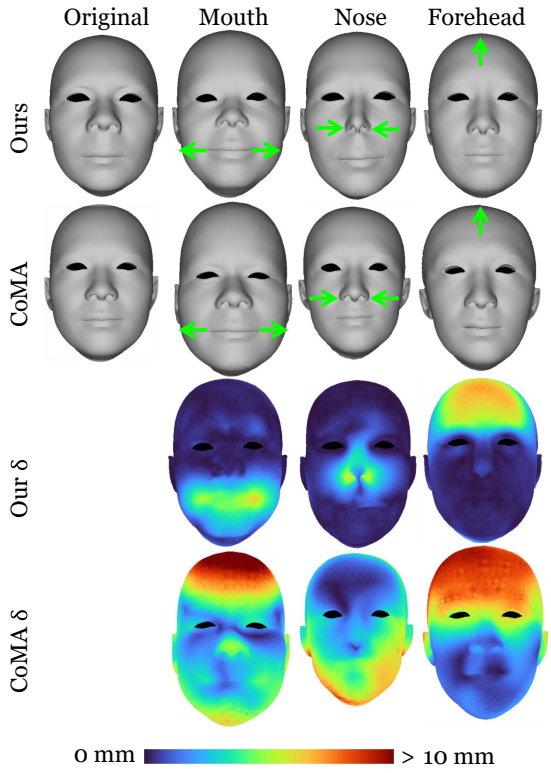

Figure 8: *Comparison with [RBSB18]. δ is vertex displacement.*

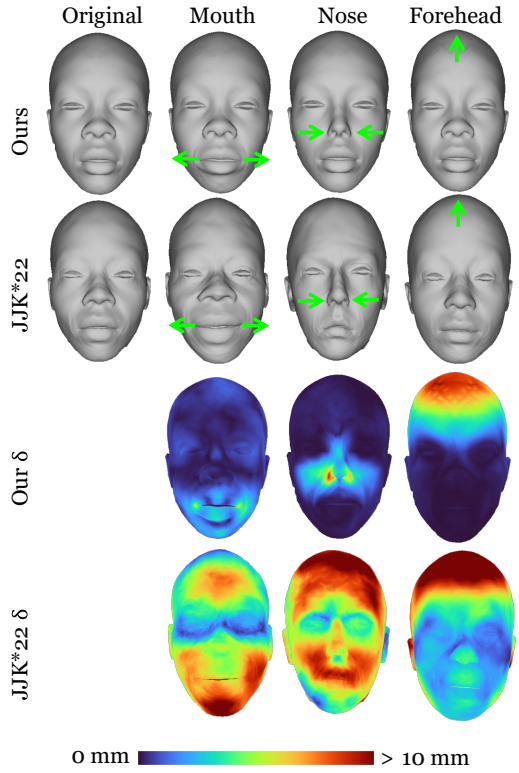

Figure 9: *Comparison with [JJK*22]. δ is vertex displacement.*

in our model is justified. Finally, we can conclude that newer extensions of CoMA [LLLY19, TGLX18, YLL*19] that only focus on the face and expression reconstruction also suffer from a non-editing-friendly latent space.

Comparison with Jung et al. [JJK*22]. In order to compare our method with that of Jung et al. [JJK*22], we calculate the average face of our primary dataset and then train the model with our primary dataset but keep the hyperparameters the same as the original work. The original dataset of Jung et al. [JJK*22] contains 1268 meshes for the training set compared to ours with 713 faces (80% of the dataset). In addition, our faces have 64% fewer vertices, but the model converges to a similar loss. Consequently, when we train the model of Jung et al. [JJK*22] on the whole dataset of 892 faces (hence, no validation set), the reconstruction error is 5.56 mm. This is a very high error compared to our model, where the reconstruction error is 1.22 mm on the training set and 1.51 mm on the validation set. We also should mention that their method has a latent dimension of 128, which is about double larger than ours. To compare the face editing capabilities of the models, we use both the "point-handle-based editing" of their work and our method (Section 3.3) to make similar modifications to an initial face. The initial face is not identical since the reconstruction power of the models differs. We run each method for 50 iterations. Both take 500 ms to converge on average on an RTX3070. The results are shown in Figure 9. We observe that while the method of Jung et al. [JJK*22] can converge to a solution, it cannot prevent the unwanted changes that appear globally on the face. In Table 2, we observe that this method has a noticeably higher vertex displacement outside of the edited part ($\bar{\delta}_{out}$ is roughly a order of magnitude larger) in all scenarios.

Model	Mouth $\bar{\delta}_{in}$, $\bar{\delta}_{out}$	Nose $\bar{\delta}_{in}$, $\bar{\delta}_{out}$	Forehead $\bar{\delta}_{in}$, $\bar{\delta}_{out}$
Fig. 8, Ours	4.71, 0.39 mm	3.44, 0.28 mm	3.73, 0.36 mm
Fig. 8, CoMA	2.47, 3.14 mm	1.96, 3.12 mm	7.54, 2.54 mm
Fig. 9, Ours	3.40, 0.92 mm	2.55, 0.18 mm	4.75, 0.23 mm
Fig. 9, [JJK*22]	6.90, 3.04 mm	4.32, 3.55 mm	7.29, 2.68 mm

Table 2: *Comparing the locality of the editing for our approach against competing methods [RBSB18, JJK*22] quantitatively (average vertex displacement inside, $\bar{\delta}_{in}$, and outside, $\bar{\delta}_{out}$, the edited part).*

Comparison with Ghafourzadeh et al. [GFR*21]. We compare our approach with the localized 3DMM editing method of Ghafourzadeh et al. [GFR*21] since both offer localized vertex-based editing. We fit that 3DMM model with our primary dataset to evaluate its face modeling application directly. The [GFR*21] method is a clustered PCA-based approach that aims to pick the best eigenvectors that bring the reconstruction loss to under 1 mm.

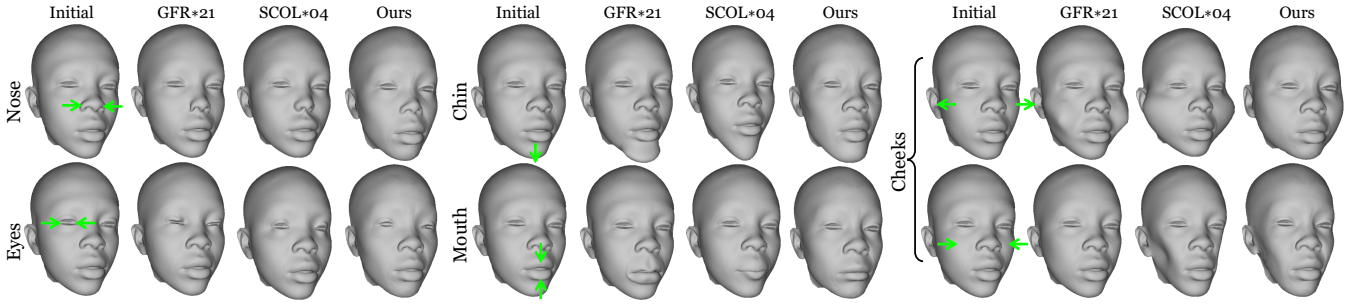

Figure 10: *Comparison with [GFR*21] and [SCOL*04]: Editing different features of the face. The green arrows indicate the edited vertices and direction of editing.*

The budget is 50 eigenvectors on average as it differs for female and male heads. Also, it segments the face, similar to our method. The method of [GFR*21] deforms each part in isolation and uses an additional smooth blending step to merge the generated part into the face. As a result, it ensures that the deformation only appears locally. Therefore, the method of Ghafourzadeh et al. [GFR*21] performs better regarding reconstruction quality and keeping the changes local. Nonetheless, in Figure 10, we observe that our model deforms the face more naturally and meaningfully while keeping the changes comparably local. For this comparison, we selected an initial face that is edited by both models. The face has an African ethnicity because we wanted to examine how well each model generalizes and works with an underrepresented face in a dataset (our dataset only contains 19 faces of this ethnicity out of 892 total). For the nose editing, we observe that while both models can decrease the width of the nose, our model preserves the shape of the nose, but theirs [GFR*21] cannot achieve the same. For mouth editing, we try to close the gap between the lips of the subject. We can see that the PCA model does not generalize well and fails to close the gap. On the other hand, our model manages to achieve its goal naturally while not getting too far from the original shape of the mouth. When editing the cheeks, we move one vertex on each cheek to create a chubbier or skinnier face. Our model achieves noticeably more plausible outputs. In contrast, the PCA model fails to output any visible changes on the face when it tries to make it skinnier. We find similar results when changing other features of the face. In conclusion, the method of [GFR*21] either cannot make visible changes or, when pushed too far, results in uncanny and linear deformations.

Comparison with direct deformation. Commercial software allow to deform surfaces using various techniques. Such techniques typically do not rely on prior knowledge of the deformed object, as opposed to our approach which uses a data set of faces. We selected the well-known Laplacian surface editing (LSE) technique [SCOL*04] to evaluate how our results can compare to such non-data-driven deformation techniques. We pick the same face that was used in the previous comparison. Editing with LSE required more manual work in defining a proper "deformable" region (the vertices which are solved) and a "fixed" region (boundary conditions) for each edit to restrict the deformation to where we expect it. We pulled on the same vertices as in our approach and moved them trying to achieve a deformation similar to the one from our results (see Figure 10). The method works well in terms of keeping the deformations local, but this is at the expense of requiring a manual design of the fixed region for each individual edit. Furthermore, the changes may look overly linear and unrealistic. The mouth, chin, and cheeks are such examples. Another reason the results look unnatural is that the method is general and does not take into account that we want to remain within the manifold of realistic faces defined by a dataset. To a certain extent, one can circumvent these issues by carefully moving the handles and defining the fixed regions iteratively and by trial and error. Nonetheless, this process can become time-consuming and more similar to manual 3D face modeling applications.

4.5. Limitations

Even if our approach compares favorably against other methods in terms of the locality of the edits, some minor deformation (< 1 mm) of other parts of the face can still be observed. We observe this in Figure 8 and Figure 9, where we show deformation heat maps in our model's outputs. When the mouth area is changed, we can see about 1 mm of unwanted changes in the forehead and eye areas.

Regarding reconstruction accuracy, DNN models such as ours need more data to improve the model accuracy. This can be seen in our tests where the reconstruction error for the dataset of 150 faces (1.44 mm training dataset and 1.77 mm validation dataset) is larger than the error from the dataset containing 892 faces (1.22 mm training dataset and 1.51 mm validation dataset). Still, along the lines of the reconstruction error, should a user want to edit a face that was manually modified in a modeling application, we will need to first encode that face to latent space, and thus inducing a reconstruction error. As such, for faces created outside of our system, the benefit of the advanced editing power comes at the expense of a slight global deformation of the face (the reconstruction error for the validation set of our primary dataset is 1.51 mm). Finally, our optimization based methods will inherently be slower compared to direct inference methods. Nevertheless, optimization based methods provide more control to the user and a performance of 0.5 seconds for meshes of around $8,000$ vertices is still very practical and even for meshes with $3\times$ higher vertex count, our method scales linearly achieving 1.5 seconds to compute the target face.

5. Conclusion

We presented a novel variational autoencoder architecture to disentangle facial features in the latent space. We feed the separate parts of the face mesh to individual encoders while we use a single decoder to reconstruct the facial mesh. We take advantage of the graph neural networks to improve the learning from meshes. As such, we can successfully train even with datasets containing only 150 meshes. Our architecture leverages the disentanglement properties of VAEs. Furthermore, given our new loss function, the network learns a latent space where each variable of the latent space influences a local region of the face mesh. Given our disentangled latent variables, our decoder is effective in sampling random faces as well as conducting face editing. For the face editing application, we developed a new loss function and a process that ensure that the face editing will remain local. The user can thus push and pull on a vertex and see the deformed face in interactive time. Thanks to our tailored latent space, the random face sampling and the facial editing both reconstruct faces that are realistic variations of the faces from the training dataset. We validated that our whole DNN architecture and learning strategy are not dependent on a specific dataset by successfully training it independently on our dataset as well as the FaceWarehouse dataset [CWZ*14]. Finally, we compared our approach with state-of-the-art methods in the application of facial editing. These comparisons demonstrated that our network has a better generalization property compared to 3DMM methods. Furthermore, our approach provides local editing while other DNN methods deform the face globally, for example, deforming the ears when editing the nose or mouth.

For now, we allocate the same number of variables for each face part in the latent vector. For future work, we would want to derive a strategy to automatically decide how many latent variables are necessary for each part. Ghafourzadeh et al. [GRF*20] derived such a technique, but in the context of 3DMM made out of PCA eigenvectors. Deriving such a strategy is quite different for DNNs. Another avenue for future work lies in the automatic adjustment of the graph convolution aspects of the learning, similar to the work of Li et al. [LLLY19]. We feel that adjusting the K factor (Equation 1) adaptively based on the mesh density of each part would improve the learning, locality, and generalization aspects of our approach.

6. Acknowledgments

This work was supported by Ubisoft Inc., the Mitacs Accelerate Program, and École de technologie supérieure. We would like to thank the anonymous reviewers for their vital feedback.

References

[BBP*19] Bouritsas G., Bokhnyak S., Ploumpis S., Bronstein M., Zafeiriou S.: Neural 3d morphable models: Spiral convolutional networks for 3d shape representation learning and generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 7213–7222. 2

[BLC*21] Bao L., Lin X., Chen Y., Zhang H., Wang S., Zhe X., Kang D., Huang H., Jiang X., Wang J., et al.: High-fidelity 3d digital human head creation from rgb-d selfies. *ACM Transactions on Graphics (TOG) 41*, 1 (2021), 1–21. 3

[BRP*18] Booth J., Roussos A., Ponniah A., Dunaway D., Zafeiriou S.: Large scale 3d morphable models. *International Journal of Computer Vision 126*, 2 (2018), 233–254. 2

[BRZ*16] Booth J., Roussos A., Zafeiriou S., Ponniah A., Dunaway D.: A 3d morphable model learnt from 10,000 faces. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 5543–5552. 2

[BV99] Blanz V., Vetter T.: A morphable model for the synthesis of 3D faces. In *Proceedings of SIGGRAPH 99* (1999), Annual Conference Series, pp. 187–194. 1, 3

[BWS*18] Bagautdinov T., Wu C., Saragih J., Fua P., Sheikh Y.: Modeling facial geometry using compositional vaes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2018). 2

[CBZ*19] Cheng S., Bronstein M., Zhou Y., Kotsia I., Pantic M., Zafeiriou S.: Meshgan: Non-linear 3d morphable models of faces. *arXiv preprint arXiv:1903.10384* (2019). 2

[CUH16] Clevert D., Unterthiner T., Hochreiter S.: Fast and accurate deep network learning by exponential linear units (ELUs). In *Proceedings of ICLR* (2016). 4

[CWZ*14] Cao C., Weng Y., Zhou S., Tong Y., Zhou K.: FaceWarehouse: A 3D facial expression database for visual computing. *IEEE Transactions on Visualization and Computer Graphics 20*, 3 (Mar. 2014), 413–425. 2, 5, 7, 10

[DBV16] Defferrard M., Bresson X., Vandergheynst P.: Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, Dec. 2016), NIPS'16, Curran Associates Inc., pp. 3844–3852. 3, 4

[DYC*20] Deng Y., Yang J., Chen D., Wen F., Tong X.: Disentangled and controllable face image generation via 3D imitative-contrastive learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2020). 4

[EST*20] Egger B., Smith W. A., Tewari A., Wuhrer S., Zollhoefer M., Beeler T., Bernard F., Bolkart T., Kortylewski A., Romdhani S., et al.: 3D morphable face models—past, present, and future. *ACM Transactions on Graphics (TOG) 39*, 5 (2020), 1–38. 2

[FA20] Fernandez-Abrevaya V.: *Large-scale learning of shape and motion models for the 3D face*. Theses, Université Grenoble Alpes [2020-....], Nov. 2020. 2, 3

[GFR*21] Ghafourzadeh D., Fallahdoust S., Rahgoshay C., Beauchamp A., Aubame A., Popa T., Paquette E.: Local control editing paradigms for part-based 3D face morphable models. *Computer Animation and Virtual Worlds* (2021), e2028. 2, 5, 8, 9

[GJvK20] Guan Y., Jahan T., van Kaick O.: Generalized Autoencoder for Volumetric Shape Generation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2020), pp. 1082–1088. 2

[GRF*20] Ghafourzadeh D., Rahgoshay C., Fallahdoust S., Beauchamp A., Aubame A., Popa T., Paquette E.: Part-based 3D face morphable model with anthropometric local control. In *Proceedings of Graphics Interface 2020* (2020), CHCCS, pp. 7 – 16. 2, 3, 5, 10

[JJK*22] Jung Y., Jang W., Kim S., Yang J., Tong X., Lee S.: Deep deformable 3d caricatures with learned shape control. In *ACM SIGGRAPH 2022 Conference Proceedings* (2022). 3, 8

[KB15] Kingma D. P., Ba J.: Adam: A method for stochastic optimization, 2015. 4, 5

[KW14] Kingma D. P., Welling M.: Auto-encoding variational bayes. In *Proceedings of ICLR* (2014). 2, 3, 4

[LBB*17] Li T., Bolkart T., Black M. J., Li H., Romero J.: Learning a model of facial shape and expression from 4d scans. *ACM Trans. Graph. 36*, 6 (2017), 194–1. 2

[LBZ*20] LI R., BLADIN K., ZHAO Y., CHINARA C., INGRAHAM O., XIANG P., REN X., PRASAD P., KISHORE B., XING J., LI H.: Learning formation of physically-based face attributes. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 3407–3416. 2

[LLLY19] LI K., LIU J., LAI Y.-K., YANG J.: Generating 3D faces using multi-column graph convolutional networks. *Computer Graphics Forum 38*, 7 (2019), 215–224. 2, 3, 8, 10

[Max] MAXON: ZBrush. Accessed: 2022-09-30. URL: https://pixologic.com/. 1

[PVO*20] PLOUMPIS S., VERVERAS E., O'SULLIVAN E., MOSCHOGLOU S., WANG H., PEARS N., SMITH W., GECER B., ZAFEIRIOU S. P.: Towards a complete 3D morphable model of the human head. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020). 2

[PWP*19] PLOUMPIS S., WANG H., PEARS N., SMITH W. A., ZAFEIRIOU S.: Combining 3d morphable models: A large scale face-and-head model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), pp. 10934–10943. 2

[RBSB18] RANJAN A., BOLKART T., SANYAL S., BLACK M. J.: Generating 3D faces using convolutional mesh autoencoders. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018). 2, 3, 4, 5, 7, 8

[SCOL*04] SORKINE O., COHEN-OR D., LIPMAN Y., ALEXA M., RÖSSL C., SEIDEL H.-P.: Laplacian surface editing. In *Proceedings of the EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing* (2004), ACM Press, pp. 179–188. 9

[TGLX18] TAN Q., GAO L., LAI Y.-K., XIA S.: Variational autoencoders for deforming 3D mesh models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2018), pp. 5841–5850. 2, 3, 8

[VRWW20] VESDAPUNT N., RUNDLE M., WU H., WANG B.: JNR: Joint-based neural rig representation for compact 3D face modeling. In *Computer Vision – ECCV 2020* (Cham, 2020), Lecture Notes in Computer Science, Springer International Publishing, pp. 389–405. 2

[WCY*22] WANG L., CHEN Z., YU T., MA C., LI L., LIU Y.: Faceverse: a fine-grained and detail-controllable 3d face morphable model from a hybrid dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022), pp. 20333–20342. 3

[YLL*19] YUAN C., LI K., LAI Y.-K., LIU Y., YANG J.: 3D face representation and reconstruction with multi-scale graph convolutional autoencoders. In *2019 IEEE International Conference on Multimedia and Expo (ICME)* (2019), pp. 1558–1563. 2, 8

[YZW*20] YANG H., ZHU H., WANG Y., HUANG M., SHEN Q., YANG R., CAO X.: Facescape: a large-scale high quality 3d face dataset and detailed riggable 3d face prediction. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (2020), pp. 601–610. 3