

DEMEA: Deep Mesh Autoencoders for Non-Rigidly Deforming Objects

— Supplementary Material —

Edgar Tretschk¹ Ayush Tewari¹
Michael Zollhöfer² Vladislav Golyanik¹ Christian Theobalt¹

¹ Max Planck Institute for Informatics, Saarland Informatics Campus
² Stanford University

In this supplementary material, we expand on several points from the main paper. In Sec. 1, we describe how we apply temporal smoothing in latent space. Sec. 2 contains details about skinning template meshes to embedded graphs. Sec. 3 shows more examples of the artifacts seen in purely convolutional architectures. Sec. 4 provides low-level details of our architecture. Sec. 5 gives the mathematical description of the losses we train with. In Sec. 6, we describe how we normalize depth maps and meshes (for reconstruction from real depth data). An expanded version of Table 2 from the main manuscript is in Sec. 7. In Sec. 8, we describe how to generate embedded graphs. In Sec. 9, we give further details and experiments on the graph convolutions. Sec. 10 contains more results of FCA and CA. We show artifacts due to coarse embedded graphs in Sec. 11.

1 Depth-to-Mesh Tracking

We can apply temporal smoothing to the reconstruction of a sequence of real depth images $\{\mathbf{D}_i\}_i$, by decoding a running (causal) exponential average of the latent vectors of this sequence. First, we encode the sequence into latent vectors $\{\mathcal{D}_i\}_i$. We then define a smoothed sequence of latent vectors $\{\mathcal{D}'_i\}_i$ as follows: let $\mathcal{D}'_0 = \mathcal{D}_0$ and set $\mathcal{D}'_i = \alpha \cdot \mathcal{D}_i + (1 - \alpha) \cdot \mathcal{D}'_{i-1}$ for $i > 0$ for some $\alpha \in [0, 1]$. The smoothed sequence of meshes $\{\mathbf{M}_i\}_i$ is obtained by decoding $\{\mathcal{D}'_i\}_i$. See the supplemental video for examples.

2 Skinning

We compute the linear blending weight $w_l(\mathbf{p})$ as [5]:

$$w_l(\mathbf{p}) = \exp\left(\frac{-\|\mathbf{g}_l - \mathbf{p}\|^2}{2 \cdot \sigma^2}\right), \quad (1)$$

where we determine $\sigma \in \mathbb{R}$ heuristically as follows:

$$\sigma = \sigma_0 \cdot d_{max} \cdot \frac{1}{\sqrt{L}}, \quad (2)$$

where $\sigma_0 = \frac{2}{3}$ and d_{max} is the maximum Euclidean distance between any two mesh vertices, a proxy for the absolute scale of the mesh.

Note that \mathcal{N}_p , w_l and σ are pre-computed on the template mesh and graph and are kept fixed within each dataset. We use $|\mathcal{N}_p| = 6$ for embedded graphs on the first level and $|\mathcal{N}_p| = 12$ for embedded graphs on the second level.

3 Artifacts

Tables 1 and 2 show additional examples of artifacts that occur when not integrating the embedded graph into the network. Even the most competitive network against which we compare (*i.e.* our ablation) suffers from visually unpleasing artifacts due to large non-rigid deformations, which most visibly occur on the hands and feet of DFaust. However, due to the localized nature of the artifacts, they do not have a large impact on the quantitative errors.

4 Architecture

Fig. 1 contains our low-level architecture. $GC(f)$ is a graph-convolutional layer

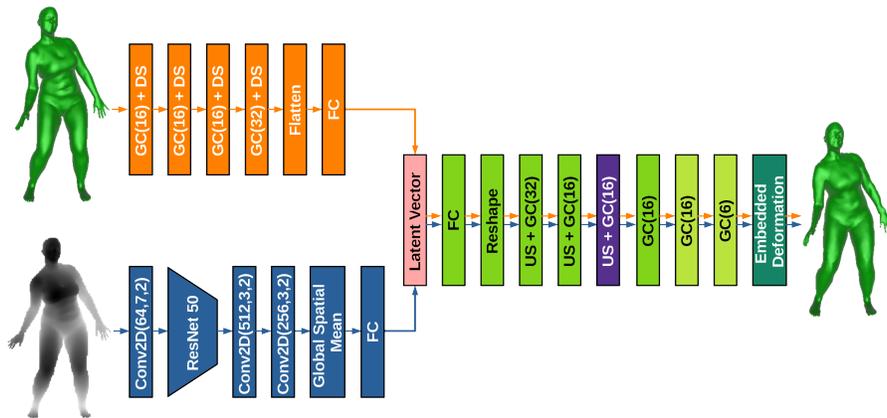


Fig. 1. The low-level architecture of DEMEA (orange path) and the depth-to-mesh network (blue path). Note that the two paths are not trained simultaneously.

with f output features. DS is a downsampling layer and US is an upsampling layer. $Conv2D(f, k, s)$ is a 2D convolution with f output features, kernel size $k \times k$ and stride s . We modified ResNetV2 50 by removing its first convolutional layer and its final non-convolutional layers, and keep its architecture unmodified otherwise. We use ELU non-linearities after every graph-convolutional, 2D convolutional and fully-connected layer except for the first 2D convolutional layer in

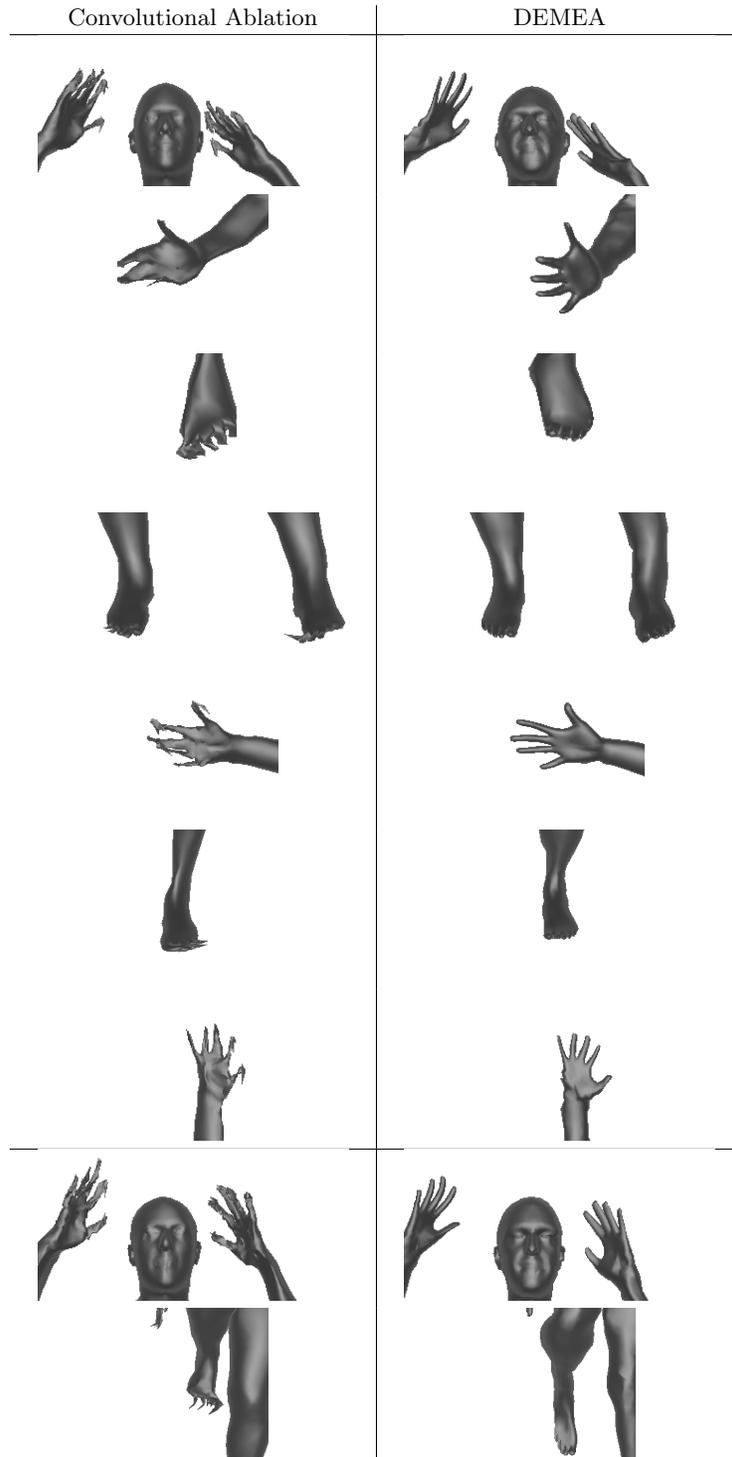


Table 1. Artifacts. The top rows use 32 latent dimensions, the last two rows are for 8.

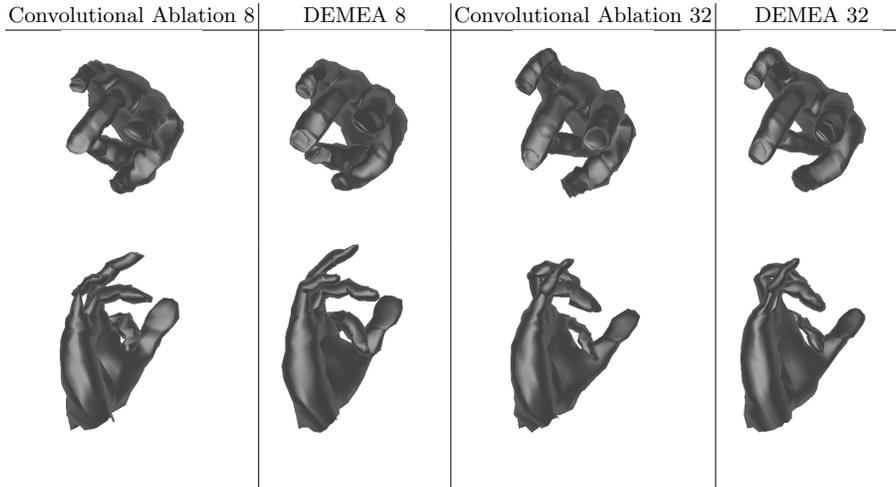


Table 2. Artifacts on SynHand5M. In contrast to CA, DEMEA yields a smooth index finger in the examples shown above.

the depth encoder and the last graph-convolutional layer. The third upsampling module (*i.e.* upsampling layer followed by a graph convolution) is only used for higher-resolution embedded graphs. In the case of spiral graph convolutions, we use the default settings of [1] to determine the length of the spirals. In the case of spectral graph convolutions, we always use $K = 6$, except for the last two layers, which use $K = 2$ for local refinement.

5 Losses

The vertex loss mentioned in Sec. 3.4 is:

$$\mathcal{L}_{vertex} = \frac{1}{N_v} \sum_{i=1}^{N_v} \|\hat{\mathbf{v}}_i - \mathbf{v}_i^*\|_1, \quad (3)$$

where $\hat{\mathbf{v}}_i$ is the i -th deformed vertex and \mathbf{v}_i^* is the i -th ground-truth vertex. The loss is averaged across the batch.

The deformed vertex can be either directly regressed (CA, MCA, FCA) or it can be computed via EDL (DEMEA, FCED). In the latter case, we follow Eq. 2 from the main manuscript. While DEMEA regresses all EDL parameters, we also try out a modification that instead uses local Procrustes inside the network, called LP, see Sec. 3.4. In that case, we use the same loss (Eq. 3) as for DEMEA but only regress the translation parameters, \mathbf{t}_l , while computing the rotation parameters of EDL, \mathbf{R}_l , using local Procrustes as described in Sec. 3.4. Note that we do not backpropagate through the rotation computation in this case.

The graph loss described in Sec. 3.4 of the main manuscript operates on the graph nodes. It encourages the regressed graph nodes positions to be close to

the ground-truth vertex positions. The graph nodes \mathbf{N} are a subset of the mesh vertices \mathbf{V} and we denote the vertex index corresponding to a graph node l as i_l . Then, the graph loss is:

$$\mathcal{L}_{graph} = \frac{1}{L} \sum_{l=1}^L \|\mathbf{t}_l - \mathbf{v}_{i_l}^*\|_1, \quad (4)$$

6 Normalization

Depth All depth-to-mesh networks rescale the depth values of the input depth map from between $0.3m$ and $7m$ to $[-1, 1]$.

Bodies: Depth For our depth-to-mesh network on bodies, we employ a number of additional normalization steps to focus on non-rigid reconstruction. First, we assume to be given a segmentation mask that filters out the background. The depth value of background pixels is set to 2. We crop the foreground tightly and use bilinear sampling to isotropically rescale the crop to 256×256 . Given such a depth crop, we compute the average (foreground) depth value and subtract it from the input. Such normalization necessitates normalizing the network output, as we will describe next.

Bodies: Meshes We first normalize out the global translation from the meshes by subtracting from each mesh vertex the average vertex position. Since scale information is also lost, we fix the scale of the meshes by normalizing their approximate spine length. To that end, we compute the approximate spine length of the template mesh and of each mesh in the dataset. We then isotropically rescale all the meshes to the same spine length as the template mesh. The depth-to-mesh body reconstruction errors in the main paper are reported for these normalized meshes.

7 Standard Deviations in Table 2

Due to space reasons, we could not fit standard deviations across the test set in Table 2 of the main manuscript. Table 3 contains the expanded version.

	DFaust		SynHand5M		Cloth		CoMA	
	8	32	8	32	8	32	8	32
CA	6.35 ± 2.40	2.07 ± 0.73	8.12 ± 1.77	2.60 ± 0.60	11.21 ± 4.58	6.50 ± 1.85	1.17 ± 0.47	0.72 ± 0.22
MCA	6.21 ± 2.48	2.13 ± 0.79	8.11 ± 1.77	2.67 ± 0.60	11.64 ± 4.58	6.59 ± 1.96	1.20 ± 0.46	0.71 ± 0.21
Ours	6.69 ± 2.76	2.23 ± 0.99	8.12 ± 1.73	2.51 ± 0.59	11.28 ± 4.65	6.40 ± 1.96	1.23 ± 0.41	0.81 ± 0.22
FCA	6.51 ± 2.45	2.17 ± 0.82	15.10 ± 4.06	2.95 ± 0.69	15.63 ± 7.18	5.99 ± 1.86	1.77 ± 0.57	0.67 ± 0.22
FCED	6.26 ± 2.35	2.14 ± 0.86	14.61 ± 3.95	2.75 ± 0.63	15.87 ± 7.73	5.94 ± 1.81	1.81 ± 0.71	0.73 ± 0.20

Table 3. Average per-vertex errors on the test sets of DFaust (in *cm*), SynHand5M (in *mm*), textureless cloth (in *mm*) and CoMA (in *mm*) for 8 and 32 latent dimensions, including standard deviations across the test sets.

8 Mesh Hierarchy

We use the code of [4] to generate the mesh hierarchy. Then, in practice, the first or the second level of this automatically generated mesh hierarchy can be used as the embedded graph.

However, we propose to use more uniform embedded graphs. Using MeshLab’s [2] implementation of quadric edge collapse decimation with default settings works well. (Note that we decimate the mesh until we reach the same number of graph nodes as used by [4].) We experimented with different ways of obtaining better embedded graphs, including hand-crafting them, but found no major differences, except that graphs generated by [4] were too non-uniform. Among the tested methods, MeshLab constitutes the least involved method of obtaining a uniform graph.

Since the embedded graph needs to be a subset of the mesh, graph nodes obtained this way need to be projected to their closest vertices. This may lead to multiple nodes projecting to the same vertex. We resolve this with a greedy assignment from nodes to vertices: looping over all nodes, the current node is assigned to its closest vertex that is not yet taken by another node. With the modification of the code of [4] described in the main paper, we can then generate the mesh hierarchy for this embedded graph.

Figure 2 shows the hierarchy we use for Cloth.

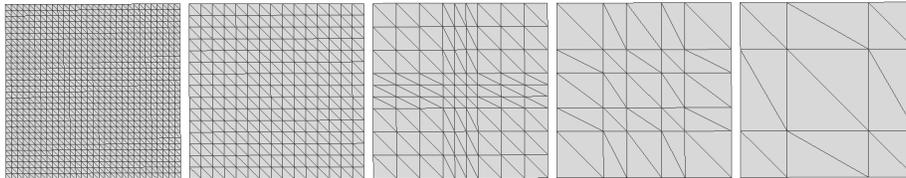


Fig. 2. The Cloth hierarchy.

9 Graph Convolutions

Our graph encoder-decoder architecture can work with multiple types of graph convolutions, which we show in this section. The results of the main paper use spiral graph convolutions, as we found these to give slightly better results than spectral graph convolutions. After defining the two types of graph convolutions, we report quantitative results.

Given an F_{in} -channel feature tensor $\mathbf{x} \in \mathbb{R}^{N \times F_{in}}$, where the features are defined at the N graph nodes, let $\mathbf{x}_{*,i} \in \mathbb{R}^N$ denote the i -th input graph feature map. The complete output feature tensor, that stacks all F_{out} feature maps, is denoted as $\mathbf{y} \in \mathbb{R}^{N \times F_{out}}$. We apply the graph convolutions without stride, *i.e.*, the input graph resolution equals the output resolution.

9.1 Spiral Graph Convolutions

We mainly work with spiral graph convolutions [1]. Let $\mathbf{x}_{n,*}^T \in \mathbb{R}^{F_{in}}$ denote the feature vector of graph node n . Assuming a fixed topology of the graph, we may choose an ordering of the neighboring nodes of graph node n : $n_0, \dots, n_s, \dots, n_{S-1}$. Bouritsas *et al.* pick a spiral ordering that starts with $n_0 = n$ and proceeds along the 1-ring (n_0^1, n_1^1, \dots) , then the 2-ring (n_0^2, n_1^2, \dots) , and so on. The spiral ordering is thus given by: $n, n_0^1, n_1^1, \dots, n_0^2, n_1^2, \dots$. All the spirals of the graph are ultimately cut to a fixed length S and zero-padded if necessary. The output of the spiral convolution is then defined as:

$$\mathbf{y}_{j,*}^T = \sum_{s=0}^{S-1} G_s \cdot \mathbf{x}_{n_s,*}^T, \quad (5)$$

where $G_s \in F_{out} \times F_{in}$ is a trainable matrix.

9.2 Spectral Graph Convolutions

The second type of graph convolutions is based on fast localized spectral filtering [3], which Ranjan *et al.* use in CoMA [4]. We compute the j -th output graph feature map $\mathbf{y}_{*,j} \in \mathbb{R}^N$ as follows:

$$\mathbf{y}_{*,j} = \sum_{i=1}^{F_{in}} g_{\theta_{i,j}}(\mathbf{L}) \cdot \mathbf{x}_{*,i}. \quad (6)$$

Here, \mathbf{L} is the normalized Laplacian matrix of the graph and the filters $g_{\theta_{i,j}}(\mathbf{L})$ are parameterized using Chebyshev polynomials of order K . More specifically,

$$g_{\theta_{i,j}}(\mathbf{L}) = \sum_{k=0}^{K-1} \theta_{i,j,k} \cdot T_k(\tilde{\mathbf{L}}), \quad (7)$$

where $\theta_{i,j,k} \in \mathbb{R}$ and $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{max} - \mathbf{I}$, with $\lambda_{max} = 2$ being an upper bound on the spectrum of \mathbf{L} . The Chebyshev polynomial T_k is defined as $T_k(x) = 2x \cdot T_{k-1}(x) - T_{k-2}(x)$, $T_1(x) = x$, and $T_0(x) = 1$.

This leads to K -localized filters that operate on the K -neighbourhoods of the nodes. Each filter $g_{\theta_{i,j}}(\mathbf{L})$ is parameterized by K coefficients, which in total leads to $F_{in} \times F_{out} \times K$ trainable parameters for each graph convolution layer.

9.3 Results

We compare our proposed DEMEA with spiral convolutions against a version of DEMEA that uses spectral convolutions. Table 4 contains the results. Except for DFaust on latent dimension 8, spiral graph convolutions always perform at least slightly better than spectral graph convolutions. These results show that EDL obtains similar accuracy with both graph convolutions, which further validates its robustness.

	DFaust		SynHand5M		Cloth		CoMA	
	8	32	8	32	8	32	8	32
Spiral	6.69	2.23	8.12	2.51	11.28	6.40	1.23	0.81
Spectral	6.56	2.40	8.74	3.83	11.76	6.52	1.40	0.98

Table 4. Average per-vertex errors on the test sets of DFaust (in *cm*), SynHand5M (in *mm*), textureless cloth (in *cm*) and CoMA (in *mm*). We compare two versions of DEMEA: one with spiral convolutions and one with spectral convolutions.

10 FCA and CA Results

Fig. 5 contains qualitative results for FCA and Fig. 6 shows artifacts when using FCA. Fig. 7 contains qualitative results for CA. We show depth-to-mesh results in Fig. 8.

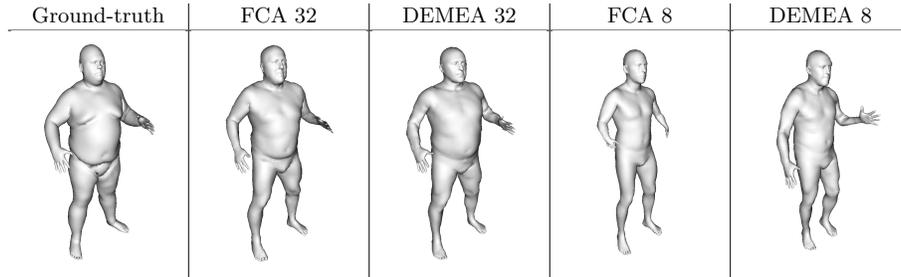


Table 5. Qualitative Results on FCA.

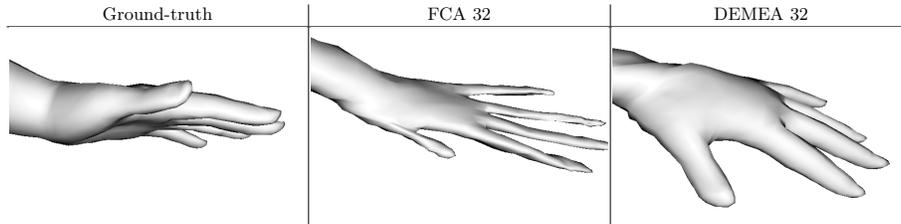


Table 6. Artifacts on FCA. While the reconstruction by DEMEA only matches the ground-truth as well as FCA, it is a significantly more plausible shape.

11 Coarse Embedded Graphs

In Fig. 9, we show how an embedded graph can lead to over-smoothing and a loss of detail.

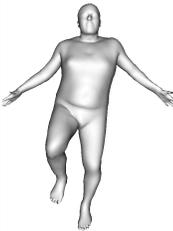
Ground-truth	CA 32	DEMEA 32	CA 8	DEMEA 8
				
				

Table 7. Qualitative Results on CA.

Depth	DEMEA	CA	FCA
			
			
			

Table 8. Depth-to-Mesh Results for CA and FCA. The bottom row shows artifacts that DEMEA avoids.

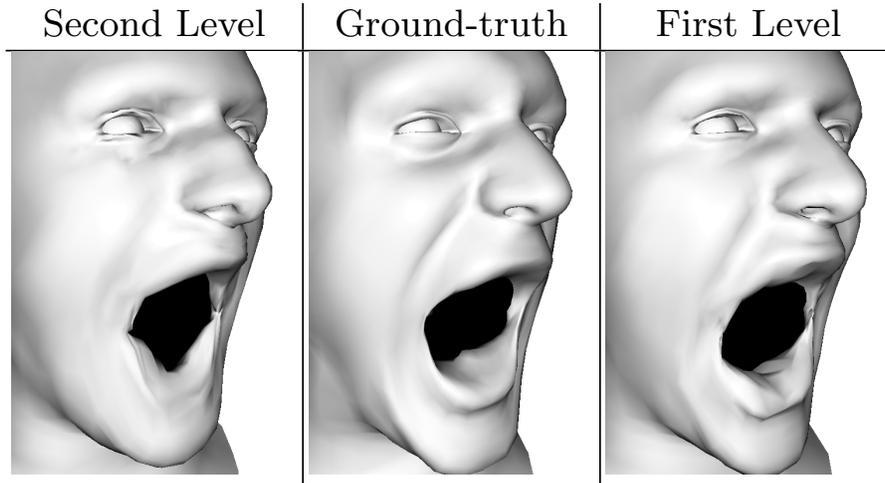


Table 9. Coarse Embedded Graphs. Note the lips. An embedded graph on the second level of the mesh hierarchy instead of the first level can lead to over-smoothing.

References

1. 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: The IEEE International Conference on Computer Vision (ICCV) (2019)
2. Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G.: MeshLab: an Open-Source Mesh Processing Tool. In: Scarano, V., Chiara, R.D., Erra, U. (eds.) Eurographics Italian Chapter Conference. The Eurographics Association (2008). <https://doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136>
3. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: International Conference on Neural Information Processing Systems (NIPS). pp. 3844–3852. NIPS’16 (2016)
4. Ranjan, A., Bolkart, T., Sanyal, S., Black, M.J.: Generating 3D faces using convolutional mesh autoencoders. In: European Conference on Computer Vision (ECCV). pp. 725–741 (2018)
5. Sumner, R.W., Schmid, J., Pauly, M.: Embedded deformation for shape manipulation. In: ACM SIGGRAPH (2007)