Learning meshless parameterization with graph convolutional neural networks

Carlotta Giannelli, Sofia Imperatore, Angelos Mantzaflaris, Felix Scholz

Learning meshless parameterization with graph convolutional neural networks

Carlotta Giannelli[1], Sofia Imperatore[1], Angelos Mantzaflaris[2], and Felix Scholz[3]

[1] Dipartimento di Matematica e Informatica, Università degli Studi di Firenze, Italy
{carlotta.giannelli,sofia.imperatore}@unifi.it
[2] Centre Inria of the French Riviera University, Sophia Antipolis, France
angelos.mantzaflaris@inria.fr
[3] Institute of Applied Geometry, Johannes Kepler University Linz, Austria
felix.scholz@jku.at

Abstract. This paper proposes a deep learning approach for parameterizing an unorganized or scattered point cloud in $\mathbb{R}^3$ with graph convolutional neural networks. It builds upon a graph convolutional neural network that predicts the weights (called *parameterization weights*) of certain convex combinations that lead to a mapping of the 3D points into a planar parameter domain. First, we compute a radius neighbours graph that yields proximity information to each 3D point in the cloud. This radius graph is then converted to its line graph, which encodes edge adjacencies, and is equipped with appropriate weights. The line graph is used as input to a graph convolutional neural network trained to predict optimal parameterizations. The proposed model outperforms closed-form choices of the parameterization weights and produces high quality parameterizations for surface reconstruction schemes.

Keywords: meshless parameterization, geometric deep learning, graph convolutional neural networks, surface reconstruction, least squares fitting

1 Introduction

Graph neural networks have been designed to process data characterized by a graph structure, e.g. 3D shapes, chemical molecules, social/relational networks, citation networks, unorganized point clouds [20]. In particular, Graph Convolutional Networks (GCNs) have become the counterpart of Convolutional Neural Networks by defining convolution operators on graph domains [3]. The translation of standard filter operators to graph operators relies on suitable aggregations of vertex and neighbour features. There are two approaches to design convolutional filters on graphs, namely *spatial* and *spectral* methods, which result in the definition of *spatial*-GCNs [10,17] and *spectral*-GCNs, see e.g., [4,5,6,16]. A comprehensive survey on graph convolutional operators can be found in [23].

While the majority of the existing research on graph neural networks focuses on classification and segmentation problems, in this paper we consider a regression problem. Due to their definition and design, GCNs aim to perform classification, segmentation or regression on the vertices $\mathcal{V}$ of a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Being interested in performing regression on the edges $\mathcal{E}$ instead of the vertices $\mathcal{V}$ of $\mathcal{G}$,

given the directed graph $\mathcal{G}$, we extract its line (dual) graph [11,12], which represents the adjacencies between edges of the original graph.

We propose a deep learning approach called PARGCN for parameterizing an unorganized or scattered point cloud in $\mathbb{R}^3$ with graph convolutional neural networks. It builds upon a GCN that predicts the weights (called *parameterization weights*) of certain convex combinations that lead to a mapping of the 3D points into a planar parameter domain.

This is a novel learning approach that goes beyond closed-form heuristic choices of the parameterization weights, see, e.g., [9]. The potential of exploiting learning techniques for point sequence parameterization, within the framework of B-spline curve approximation, was originally proposed in [14] by exploiting multilayer perceptron architectures, whereas in [21], the parameterization of point sequences for polynomial curve approximation was computed via residual deep neural networks. Note that in [24] a dynamic network was used to construct parameterizations of 3D triangular meshes. To this end, a separate network is trained for each point cloud. In the present work instead, we propose a new general network based mapping *without* connectivity information.

The structure of the paper is as follows. Section 2 briefly introduces known techniques for meshless barycentric parameterization. Our approach to construct data-driven meshless parameterizations is presented in Section 3, together with details on the training procedure. Finally, numerical experiments are presented in Section 4.

2 Meshless barycentric parameterization

Let $\mathcal{P} = \left\{\mathbf{p}_i \in \mathbb{R}^3, i = 1, \dots, m\right\}$ be an unorganized point cloud partitioned into two disjoint subsets, $\mathcal{P}_I = \{\mathbf{p}_1, \dots, \mathbf{p}_n\}$ and $\mathcal{P}_B = \{\mathbf{p}_{n+1}, \dots, \mathbf{p}_m\}$ that identify the sets of interior and boundary points, respectively. In order to consider parametric surface reconstruction schemes, we want to identify a parameterization $\left\{\boldsymbol{u}_i = [u_i^1, u_i^2]^T \in \Omega \subset \mathbb{R}^2, k = 1, \dots, m\right\}$ to associate suitable parameter values $\boldsymbol{u}_i$ to any point in $\mathcal{P}$.

Standard methods to parameterize unstructured point clouds over convex polygonal domains usually consist of two steps, see e.g., [9]. Firstly, the boundary points in $\mathcal{P}_B$ are parameterized over the boundary of the parameter domain Ω, resulting in parameters $\{\boldsymbol{u}_{n+1}, \dots, \boldsymbol{u}_m\}$. Secondly, the interior parameters are then found as convex combinations of their neighbours in a certain directed graph.

As far as the first step of the method is concerned, several approaches are available for the parameterization of point sequences. Standard choices are uniform, chord length and centripetal parameterization [18], as well as subsequent improvements [1,7]. Point sequence parameterization can also be computed using iterative schemes [13,19] or artificial neural networks [14,21]. In this work, we therefore assume that the parametric values $\{\boldsymbol{u}_{n+1}, \dots, \boldsymbol{u}_m\}$ corresponding to the boundary points in $\mathcal{P}_B$ are known.

In the second step of the parameterization method, for each interior point $\boldsymbol{p}_i \in \mathcal{P}_I$, $i = 1, \dots, n$, a neighbourhood $N_i \subset \{1, \dots, m\} \setminus \{i\}$ has to be determined. In particular, for each $i = 1, \dots, n$, we consider the ball neighbourhood

$$N_i = \{j \in \{1, \dots, m\} : 0 < \|\boldsymbol{p}_i - \boldsymbol{p}_j\| < r\}, \tag{1}$$

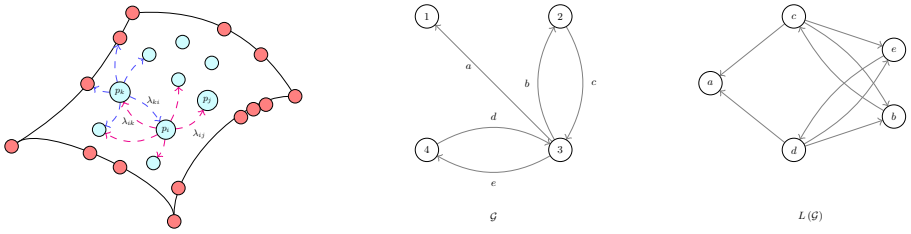

Fig. 1: The sets N_i and N_k associated to the interior points $\boldsymbol{p}_i$ and $\boldsymbol{p}_k$ (left). A directed graph $\mathcal{G}$ (center) and the corresponding directed line graph $L(\mathcal{G})$ (right) are also shown.

for some radius $r > 0$. The choice of a neighbourhood for each point in $\mathcal{P}_I$ leads to a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, whose vertices $\mathcal{V} = \{1, \ldots, m\}$ are the indices of $\mathcal{P}$ and whose directed edges $\mathcal{E}$ are all ordered pairs (i, j) such that $i \in \{1, \ldots n\}$ and $j \in N_i$. As a consequence, each directed edge $(i, j) \in \mathcal{E}$ connects an interior index i with either an interior or a boundary index j. The graph $\mathcal{G}$ is the so-called fixed-radius near neighbour graph [2] from $\mathcal{P}_I$ into $\mathcal{P}$ or, for short, *radius graph* of the points $\mathcal{P}$. An example of two interior points $\boldsymbol{p}_i, \boldsymbol{p}_k \in \mathcal{P}_I$ and their corresponding neighbourhoods N_i and N_k are shown on the left of Figure 1, where the vertices in $\mathcal{G}$ are identified with the corresponding points in $\mathcal{P}$.

The unknown parameter $\boldsymbol{u}_i$ for each point $p_i \in \mathcal{P}_I$ is then computed as a convex combination of its neighbour points in the graph $\mathcal{G}$, by considering a set of *parameterization weights* $\lambda_{ij} > 0$ with $\sum_{j \in N_i} \lambda_{ij} = 1$. We can then arrange the linear equations as the linear system

$$\mathbf{u}_i = \sum_{j \in N_i} \lambda_{ij}\, \mathbf{u}_j, \;\; i = 1, \ldots, n, \qquad \Rightarrow \qquad A\, U_{\mathrm{I}} = C, \tag{2}$$

where A is the sparse $n \times n$-matrix $A = [-\lambda_{ij}] + I_n$ with $\lambda_{ij} = 0$ if $j \notin N_i$, C is the $n \times 2$ matrix $C = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n]^\top$ with $\boldsymbol{c}_i = \sum_{j=n+1}^{m} \lambda_{ij} \boldsymbol{u}_j$ and $U_{\mathrm{I}} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n]^\top$ is the $n \times 2$ matrix that contains the unknown parameters of the interior points.

In order to guarantee that the matrix A in (2) has full rank, the choice of the radius r is crucial. As proved in [9, Proposition 3.3], each interior parameter needs to be related to at least one boundary point by a sequence of linear equations, which corresponds to the existence of a path in $\mathcal{G}$ from each interior index $i = 1 \ldots n$ to any boundary index.

The specific choice of the parameterization weights λ_{ij}, for $i = 1, \ldots, n$ and $j \in N_i$, determines the parameterization of the interior points $\mathcal{P}_I$ and, consequently, strongly influences the quality of the results. For example, the following three heuristic choices were suggested in [9]. The *uniform* weights are simply defined as $\lambda_{ij} := 1/d_i$ with $d_i = |N_i|$, in order to generalize the concept of uniform parameterization of curves to surfaces.The *reciprocal distance weights* instead, defined as $\lambda_{ij} := \|\boldsymbol{p}_j - \boldsymbol{p}_i\|^{-1} / \sum_{k \in N_i} \|\boldsymbol{p}_k - \boldsymbol{p}_i\|^{-1}$, are meant to generalize the concept of chord-length parameterization to surfaces. The third choice is the *shape preserving* parameterization, which is not defined on the radius graph. Instead, for all interior points $\mathbf{p}_i$ the neighbourhood (1) is projected onto its best-approximating plane.

The resulting planar point cloud is then triangulated by a Delaunay triangulation. The neighbours of $\mathbf{p}_i$ in the Delaunay triangulation are its neighbours in the *local projection* graph $\mathcal{G}$. The corresponding weights are found based on the shape preserving weights for triangulated surfaces presented in [8]. For each interior point $\mathbf{p}_i$, the neighbours are ordered counter-clockwise and intermediate local parameters $\tilde{\mathbf{u}}_j$ are determined for all $j \in N_i$ such that the edge lengths and the angle ratios around $\mathbf{p}_i$ are preserved. Finally, corresponding weights λ_{ij} are computed from the parameters $\tilde{\mathbf{u}}_j$.

In general, there is no constraint on the choice of the weights, besides all weights being positive and weights that belong to a common interior point to sum up to one. Therefore, the space of possible weights, and resulting parameterizations, is very large. This motivates us to train a deep neural network to predict the optimal choice of parameterization weights.

3 Meshless data-driven parameterization

In this section, we present the details of our neural network based method for predicting the optimal parameterization weights λ_{ij} in (2) for an unstructured point cloud with parameterized boundary curves. The output of the network is a set of parameterization weights λ_{ij} corresponding to the directed edges of the radius graph of the point cloud. We first obtain the corresponding parameterization and then perform an additional post-processing step.

3.1 Pre-processing and feature extraction

As a preprocessing step, the input point cloud is normalized by translation and scaling into $[0,1]^3$. As outlined in Section 2, the interior points in $\mathcal{P}_I$ are parameterized using neighbourhood relationships in a directed graph $\mathcal{G} = (\mathcal{E}, \mathcal{V})$, which is the radius graph from $\mathcal{P}_I$ into $\mathcal{P}$, based on the ball neighbourhoods (1). Hence, predicting optimal parameterization weights corresponds to predicting edge weights on the directed graph. We then encode additional information by defining for each edge $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, i.e. $(i,j) \in \mathcal{E}$, the edge feature $\boldsymbol{e}_{ij} \coloneqq [\boldsymbol{p}_i, \boldsymbol{p}_i - \boldsymbol{p}_j]^T \in \mathbb{R}^6$. This choice of features was suggested in [22] and is motivated by the aim to achieve partial translation-invariance.

Numerous existing graph neural network architectures can incorporate given *edge weights* in their convolution; their predictions, however, live on the vertices of the graph. Therefore, in order to predict an output on the directed edges of $\mathcal{G} = (\mathcal{E}, \mathcal{V})$, we transform $\mathcal{G}$ into its *line graph* $L(\mathcal{G}) = (\mathcal{V}', \mathcal{E}')$ whose vertices $\mathcal{V}'$ correspond to $\mathcal{E}$, i.e. $\mathcal{V}' = \{(i,j) : i,j \in \mathcal{V}, (i,j) \in \mathcal{E}\}$ and two vertices $(i,j), (k,\ell) \in \mathcal{V}'$ join with an edge in $\mathcal{E}'$ if $i,j,k,\ell \in \mathcal{V}$ and $j = k$. An example of a directed graph $\mathcal{G}$ and its corresponding directed line graph $L(\mathcal{G})$ is shown in Figure 1. Finally, the edge features defined on $\mathcal{E}$ of $\mathcal{G}$ are consequently transferred to vertex features on $\mathcal{V}'$ of $L(\mathcal{G})$. Additional details on line graphs can be found in [12].

3.2 Architecture

The architecture developed for this study is a graph convolutional neural network, characterized by a suitable choice of fast and localized spectral convolutional ope-

rators introduced in [6]. More precisely, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we are interested in performing vertex regression on $\mathcal{G}$ to properly process the features defined on $\mathcal{V}$. Assuming $\mathcal{V}$ to be finite, with $|\mathcal{V}| = v$, let $W \in \mathbb{R}^{v \times v}$ be the (weighted) adjacency matrix describing the graph connections between pair of vertices, i.e., $W_{ij} = 0$ if $(i,j) \notin \mathcal{E}$ and $W_{ij} > 0$ if $(i,j) \in \mathcal{E}$. Moreover, let $D \in \mathbb{R}^{v \times v}$ be the degree matrix, i.e. a diagonal matrix so that $D_{ii} \coloneqq \sum_{j=1, j \neq i}^{v} w_{ij}$, for $i = 1, \ldots, v$. The spectral domain of a graph can be determined by the normalized self-adjoint Laplacian operator $\Delta \coloneqq I_v - D^{\frac{1}{2}} W D^{\frac{1}{2}}$, where I_v is the $v \times v$ identity matrix. Let $\{\mu_\ell\}_{\ell=1}^{v}$ $\mu_\ell \in \mathbb{R}_{\geq 0}$ for each ℓ, be the set of eigenvalues for Δ and let $\{\boldsymbol{x}_\ell\}_{\ell=1}^{v}$, $\boldsymbol{x}_\ell \in \mathbb{R}^v$ their associated orthonormal eigenvectors. It follows that for $\boldsymbol{s} \in \mathbb{R}^v$, $\Delta \boldsymbol{s} = \sum_{i=1}^{v} \mu_i \langle \boldsymbol{s}, \boldsymbol{x}_i \rangle \boldsymbol{x}_i$, where $\langle \cdot, \cdot \rangle$ is an inner product in $\mathbb{R}^v$. In addition, let $g_\theta : \mathbb{R} \to \mathbb{R}$ be a filter function depending on the parameter $\theta \in \mathbb{R}$, so that $g_\theta(\Delta)\, \boldsymbol{s} = \sum_{i=1}^{v} g_\theta(\mu_i) \langle \boldsymbol{s}, \boldsymbol{x}_i \rangle \boldsymbol{x}_i$. In particular, for some filters, $g_\theta(\Delta)$ has an explicit formulation. This is the case of [6], where g_θ is defined as a polynomial filter, so that $g_\theta(\mu_i) = \sum_{j=0}^{d} \theta_j \mu_i^j$ for each $i = 1, \ldots, v$, where θ_j are the Chebychev polynomial coefficients. During the training phase, described later in Section 3.5, optimal values for θ_j will be computed. For further details about spectral convolutional operators and their properties, see [6] and [15].

To predict the weights needed for constructing a meshless parameterization of an unorganized point cloud, we design a sequential graph convolutional neural network to which we add shortcut connections. The layout of the learning architecture is illustrated in Figure 2. For each scattered data point cloud, we perform the operations descibed in Section 3.1. The resulting directed line graph $L(\mathcal{G}) = (\mathcal{V}', \mathcal{E}')$ together with its vertex features form the input (Input) of the neural network. The data are then forwarded through 4 spectral convolutional layers (ChC), which are characterized by the Chebychev polynomial filters proposed in [6], based on the normalized graph Laplacian Δ. For each layer $\ell = 1 \ldots, 4$, the corresponding convolutional layer $\mathrm{ChC}\big(f_\ell^i, f_\ell^o, g_\ell\big)$ is defined by declaring the size of input and output features f_ℓ^i, f_ℓ^o and the dimension of the convolving filter g_ℓ. More precisely, we choose $g_\ell = 2d + 1$, where d is the polynomial degree used in the loss function. Regarding the dimension of the layer features, for the first layer we set $f_1^i = 6$, according to the input features definition, whereas we choose $f_1^o, f_\ell^{in}, f_\ell^o = 64$ for $\ell = 2, \ldots, 4$. A rectified linear unit activation function (relu) is then applied element-wise to the output of any ChC layer. In order to prevent the degradation problem, due to numerical instabilities from a potentially too high number or learnable parameters, we introduce short-cut connections. We concatenate the output of each ReLU function (cat), thereby enabling the network to skip the sequence of layers in between. The content of the concatenation layer is subsequently processed by a multi layer perceptron (mlp) with one input layer, one hidden layer and one output layer, of dimension $f^i = 262$, $f^h = 64$, $f^o = 1$, respectively. Note that, f^i is a constrained dimension due to the dimension of the previous concatenation layer, f^h is an authors choice, and f^o corresponds to the dimension of the output features related to the problem. In this case we want to predict one weight for each vertex of the line graph $L(\mathcal{G})$, corresponding to each edge of the graph $\mathcal{G}$. As a final activation function, we apply the softmax function (σ) on each local neighbourhood N_i, with $i = 1, \ldots, n$, in order to guarantee that the parameterization weights are positive and form a partition of unity for each N_i.

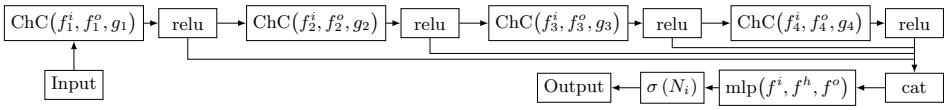

Fig. 2: Architecture design.

Finally, the output (Output) of the model is a vector of length $|\mathcal{V}'|$, whose components correspond to the predicted parameterization weights λ_{ij} for $i = 1, \dots, n$ and $j \in N_i$. These weights are used to assemble and solve the linear system in (2).

3.3 Loss function

For training the network, we pursue an unsupervised learning approach: from the predicted parameterization weights, we first solve the linear system (2) to obtain a suitable parameterization $\{\boldsymbol{u}_1, \dots, \boldsymbol{u}_m\}$ of the point cloud. When training the network, we use this parameterization to fit the data with a tensor-product Bézier surface $\boldsymbol{S}(u^1, u^2) = \sum_{i=0}^{d} \sum_{j=0}^{d} \boldsymbol{b}_{ij} B_i^d(u^1) B_j^d(u^2)$, where $B_i^d(\cdot)$ are the Bernstein polynomials of bidegree $(d, d) \in \mathbb{N}^2$ and $\boldsymbol{b}_{ij} \in \mathbb{R}^3$, $i, j = 0, \dots, d$, are the unknown control points. This is done by solving the linear least squares problem $\arg\min_{\boldsymbol{b}} \sum_{k=1}^{m} \left\| \boldsymbol{S}(u_k^1, u_k^2) - \boldsymbol{p}_k \right\|^2$. The residual of the solution to this least squares problem is used to define our loss function $L(\lambda_{ij})$, which shall guide our learning procedure during the training.

One advantage of this unsupervised learning approach is that the network can be trained on arbitrary point clouds, even if the points where not sampled from a tensor-product Bézier surface or if no parameterization is known. Moreover, even for training data sampled from tensor-product Bézier surfaces, minimizing the residual of the fitting problem leads to better experimental results than minimizing the mean squared error with respect to the parameterization weights or the parameters.

3.4 Post-processing

After the system in (2) is solved with the *parameterization weights* in output from the network, we use the resulting parameter values to obtain a *global* Delaunay triangulation of the planar domain. Subsequently, for each interior point $\mathbf{p}_i$, $i = 1, \dots, n$, we determine the shape preserving weights λ_{ij} with respect to the corresponding neighbours $\mathbf{p}_j$ in the planar triangulation, as described in Section 2. By solving the linear system (2) once more based on the neighbourhoods defined by the Delaunay triangulation and the new weights, we obtain the final parameterization.

3.5 Data generation and training

Since there are no public data sets available that are suitable for the parameterization of unorganized point clouds, we generate synthetic data. To this end, we generate tensor-product Bézier surfaces with control points $\boldsymbol{b}_{ij} = [b_{ij}^1, b_{ij}^2, b_{ij}^3]^T$, where $[b_{ij}^1, b_{ij}^2]^T$ are chosen as the tensor-product Greville abscissae for bidegree

(d, d) and b_{ij}^3 are sampled randomly accordingly to the uniform distribution in $[0, 1]$. To attain rotation invariance, we rotate each surface around a random rotation axis by a random angle. From each surface, we sample $m = n + 4\lceil\sqrt{n} + 1\rceil + 4$ items, more precisely n interior points as well as $4\lceil\sqrt{n} + 1\rceil + 4$ boundary points. Moreover, for the latter ones we store the exact parameters in $[0, 1]^2$, in addition to their positions in $\mathbb{R}^3$. We used this algorithm to generate a training data set of 10.000 point clouds sampled from polynomial parametric surfaces of bidegree (d, d) for $d = 2$, each consisting of $m = 264$ points of which $n = 200$ were sampled from the surface interior. Before training, we performed the preprocessing and feature extraction steps described in Section 3.1. For computing the radius graph, we set the radius to $r = 0.2$. This choice of hyperparameters leads to a preprocessed training set of size 29GB. We additionally generated a validation data set of 2.500 preprocessed point clouds (7.2GB) with the same properties of the training set in order to choose the final model.

The architecture we design has 80.641 learnable parameters, whose (optimal) values are set by solving the stochastic optimization problem using the Adam optimizer with learning rate $1e-3$ and momentum 0.9. In order to compute the loss function described in Section 3.3, we solve the polynomial least squares problem with bidegree (d, d) with $d = 2$, namely the same polynomial degree that characterizes the training and validation data. According to the analysis of the learning curves, we consider the trained model corresponding to the 50th epoch.

We remark that our network architecture can also be trained on the graph obtained by local projections that is used for the shape preserving parameterization, see Section 2. In our experiments, this resulted in similar training and validation errors as for training on the radius graph.

4 Numerical results

In this section we analyze the performance and the generalization capability of the meshless learning parameterization method (PARGCN) on a variety of different scattered point cloud data sets. For each test we estimate the quality of the reconstructed geometric model in terms of the mean square error (MSE) and compare the results of PARGCN with those obtained using the reciprocal distance (RECD) and the shape preserving (LPSP) parameterization weights. We remind that after the preprocessing step, each point cloud is contained in the unit cube $[0, 1]^3$, hence a few digits of difference in the error measures correspond to a significant gain in accuracy.

4.1 Polynomial approximation of synthetic data

In this section we validate the generalization capability of the proposed learning model with respect to polynomial fitting for different bidegrees, as well as the robustness of the model in presence of noise.

We evaluate the PARGCN method on unorganized point clouds sampled from polynomial surfaces of various bidegrees. In particular, we generate 100 unorganized point clouds by sampling $m = 264$ points, of which $n = 200$ are interior points and 64 are boundary points, from tensor-product polynomial surfaces of

	$d=2$			$d=3$			$d=4$			$d=5$		
	RECD	LPSP	PARGCN	RECD	LPSP	PARGCN	RECD	LPSP	PARGCN	RECD	LPSP	PARGCN
(a)	5.01e−4	5.75e−5	**1.69e−5**	3.44e−4	2.93e−5	**1.40e−5**	2.38e−4	3.51e−5	**1.34e−5**	1.34e−4	2.60e−5	**1.06e−5**
(b)	5.00e−4	5.31e−5	**3.43e−5**	3.67e−4	1.15e−4	**6.41e−5**	2.64e−4	9.14e−5	**5.15e−5**	1.58e−4	7.60e−5	**4.53e−5**
(c)	6.80e−4	1.32e−4	**7.55e−5**	2.55e−4	**2.51e−5**	4.10e−5	1.74e−4	2.27e−5	**1.10e−5**	1.20e−4	1.68e−5	**6.39e−6**
(d)	3.54e−3	1.55e−3	**8.62e−4**	2.25e−3	7.10e−4	**4.31e−4**	1.71e−3	4.30e−4	**3.57e−4**	1.21e−3	3.37e−4	**2.42e−4**
(e)	2.68e−4	8.43e−5	**2.76e−5**	2.42e−4	8.97e−5	**2.52e**−5	2.43e−4	8.13e−5	**2.48e−5**	2.31e−4	8.48e−5	**2.39e−5**

Table 1: MSE errors for different parameterization values and different degrees. Lines (a) and (b) correspond to the experiments of Section 4.1, line (c) shows the results obtained in Section 4.2, while lines (d) and (e) report the values obtained in Section 4.3.

bidegree (d, d) for $d = 2, 3, 4, 5$. In addition, we generate a test set similar to the previous one but corrupted with random Gaussian noise by a factor $\epsilon = 1\mathrm{e} - 2$. For each point cloud, we perform the feature extraction described in Section 3.1, setting the radius to $r = 0.2$. Finally we run the least squares fitting scheme for the corresponding polynomial bidegree (d, d). The MSE for each bidegree and the different parameterization methods are shown on line (a) and (b) of Table 1 for the case of exact and noisy data, respectively. We observe that the accuracy gained by PARGCN with respect to LPSP in terms of MSE is between 52% and 70% on exact data, see line (a) of Table 1. For noisy data PARGCN gains on average for each degree 41% of accuracy with respect to the MSE values obtained with LPSP, see line (b) of Table 1. The metric values for PARGCN prove that performing the training with a fixed polynomial degree does not induce a bias in the final learning model. In addition, the results of PARGCN are suitable for solving the parameterization problem of scattered data for a variety of degrees while remaining robust with respect to noise.

4.2 Polynomial approximation of a ship hull

In this example we show the capabilities of the PARGCN method to generalize with respect to data sets of different nature and size. We consider the point cloud shown in Figure 3 obtained by randomly sampling $m = 596$ points (500 interior and 96 boundary points) from a B-spline model of a ship hull. We then preprocess the data as described in Section 3.1 building a radius graph with $r = 0.1$. Finally, we parameterize the input data with the RECD, LPSP, and PARGCN methods and compute the least squares tensor-product polynomial approximation of bidegree (d, d) with $d = 2, 3, 4, 5$. The MSE errors are reported on line (c) of Table 1, whereas the polynomial reconstructed models are shown in Figure 3. While the approximations obtained with RECD parameterization show a significant mesh distortion, the LPSP and PARGCN parameterizations lead to effective reconstructions, always with a reduced MSE for the second one. When comparing the MSE for PARGCN and LPSP, we register an average gain of 49% accuracy for the first method with respect to the second one, see again line (c) of Table 1.

4.3 B-spline approximation of a face

In this experiment, we demonstrate that the learning meshless parameterization model is capable of properly generalizing from polynomial to B-spline fitting of

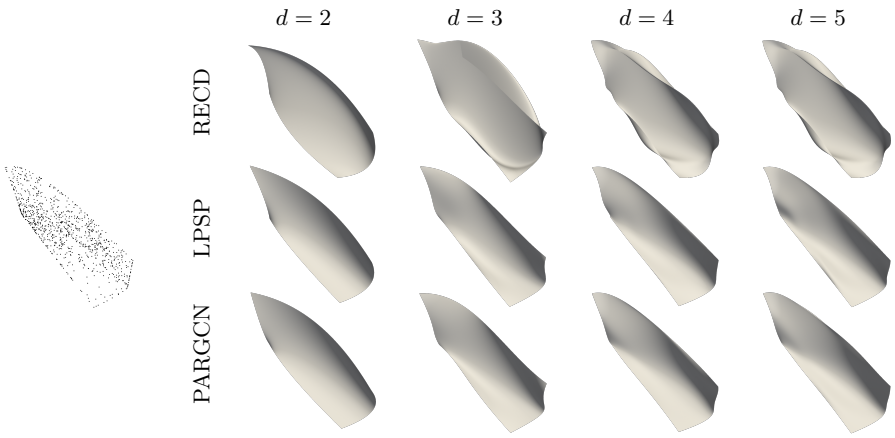

Fig. 3: Polynomial least squares approximation of the ship-hull point cloud (left), obtained with RECD, LPSP, and PARGCN for different bidegree (d, d).

scattered data. In particular, we show the suitability of the output parameterization for tensor-product B-spline penalized least squares fitting. To this end, we process an unorganized point cloud consisting of $m = 543$ points (458 interior points and 85 boundary points) sampled from a face model, see Figure 4 (left) and build the radius graph with $r = 0.2$. The MSE for polynomial approximation for different bidegrees (d, d), for $d = 2, 3, 4, 5$ is shown on line (d) of Table 1, while line (e) of Table 1 reports the MSE obtained when performing a (penalized) least squares fitting with tensor-product B-splines characterized by 4 levels of uniform refinement. When PARGCN is used to perform polynomial least squares approximation of different bidegrees, we are able to gain on average 32% of accuracy with respect to LPSP . Moving to the B-spline setting, the MSE obviously decreases for all methods, but PARGCN reaches a gain in accuracy of 70% on average for each degree. The considered B-spline model is shown in Figure 4 (center), together with the parameter distribution along the first parametric direction of the resulting B-spline surface (right). Moreover, the parametric values in $[0, 1]^2$, obtained with RECD, LPSP and PARGCN parameterization methods, are shown on top of Figure 5. We may note that the parametric values for RECD and LPSP are more clustered and therefore lead to bigger voids (i.e. lack of data) in the parametric domain. Finally, the triangulations resulting from mapping a Delaunay triangulation of the parameters to the point cloud are shown on the bottom of Figure 5 for RECD, LPSP, and PARGCN methods. In particular, mesh distortion phenomena and the presence of artifacts can be observed when RECD and LPSP parameterizations are considered, whereas none of them arises for PARGCN parameterization.

Acnowledgements

CG acknowledges the contribution of the National Recovery and Resilience Plan, Mission 4 Component 2 – Investment 1.4 – CN_00000013 “CENTRO NAZIONALE HPC, BIG DATA E QUANTUM COMPUTING”, spoke 6. CG and SI are members

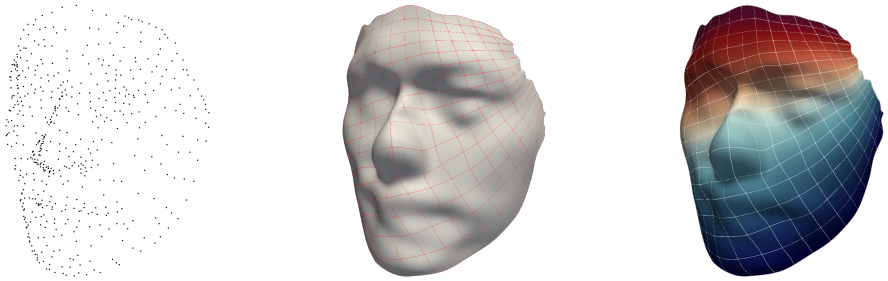

Fig. 4: Tensor-product B-spline appoximation (center) of the face model point cloud (left) with PARGCN parameterization. The resulting parameter distribution for the first parametric direction is also shown (right).

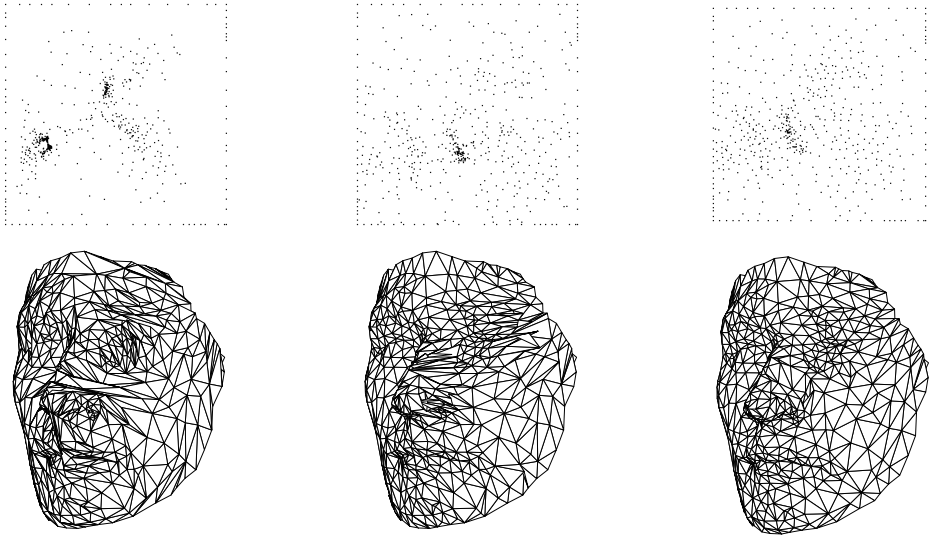

Fig. 5: Parameter distributions (top) and 3D Delaunay triangulations (bottom) obtained with RECD (left), LPSP (center) and PARGCN (right) for the face model.

of the INdAM group GNCS. The INdAM-GNCS support is gratefully acknowledged. CG, SI, AM also acknowledge the PHC GALILEE project 47786N and AM acknowledges H2020 Marie Sklodowska-Curie grant GRAPES No. 860843.

References

1. Balta, C., Öztürk, S., Kuncan, M., Kandilli, I.: Dynamic centripetal parameterization method for b-spline curve interpolation. IEEE Access **8**, 589–598 (2020)
2. Bentley, J.L., Stanat, D.F., Williams, E.: The complexity of finding fixed-radius near neighbors. Information Processing Letters **6**(6), 209–212 (1977)
3. Bronstein, M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: Going beyond Euclidean data. IEEE Sig. Proc. Mag. **34**(4), 18–42 (2017)

4. Bruna, J., Zaremba, W., Szlam, A., Lecun, Y.: Spectral networks and locally connected networks on graphs. In: Int'l Conf.on Learning Repr. (ICLR2014) (2014)
5. Chung, F.R.: Spectral graph theory, vol. 92. American Mathematical Soc. (1997)
6. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. Adv. in Neural Inf. Proc. Sys. **29** (2016)
7. Fang, J.J., Hung, C.L.: An improved parameterization method for b-spline curve and surface interpolation. Computer-Aided Design **45**(6), 1005–1028 (2013)
8. Floater, M.S.: Parameterization and smooth approximation of surface triangulations. Computer Aided Geometric Design **14**, 231–250 (1997)
9. Floater, M.S., Reimers, M.: Meshless parameterization and surface reconstruction. Computer Aided Geometric Design **18**(2), 77–92 (2001)
10. Gori, M., Monfardini, G., Scarselli, F.: A new model for learning in graph domains. In: Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., vol. 2, pp. 729–734 vol. 2 (2005)
11. Gross, J.L., Yellen, J., Anderson, M.: Graph theory and its applications. CRC (2018)
12. Harary, F., Norman, R.Z.: Some properties of line digraphs. Rendiconti del Circolo Matematico di Palermo **9**(2), 161–168 (1960)
13. Hoschek, J.: Intrinsic parametrization for approximation. Computer Aided Geometric Design **5**(1), 27–31 (1988)
14. Laube, P., Franz, M.O., Umlauf, G.: Deep Learning Parametrization for B-Spline Curve Approximation. In: 2018 International Conference on 3D Vision (3DV), pp. 691–699. IEEE Computer Society, Los Alamitos, CA, USA (2018)
15. Levie,R., Huang,W., Bucci,L., Bronstein,M., Kutyniok,G.: Transferability of spectral graph convolutional neural networks. J. Mach. Learn. Res.**22**(1), 12462–12520 (2022)
16. Levie, R., Monti, F., Bresson, X., Bronstein, M.: Cayleynets: Graph convolutional neural networks with complex rational spectral filters. IEEE Trans. on Sig. Processing **67**, 97–109 (2017)
17. Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., Bronstein, M.M.: Geometric deep learning on graphs and manifolds using mixture model cnns. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5425–5434. IEEE Computer Society, Los Alamitos, CA, USA (2017)
18. Piegl, L., Tiller, W.: The NURBS Book. Springer-Verlag, Berlin, Heidelberg (1995)
19. Saux, E., Daniel, M.: An improved hoschek intrinsic parameterization. Computer Aided Geometric Design **20**, 513–521 (2003)
20. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks **20**(1), 61–80 (2009)
21. Scholz, F., Jüttler, B.: Parameterization for polynomial curve approximation via residual deep neural networks. Computer Aided Geometric Design **85**, 101,977 (2021)
22. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph CNN for learning on point clouds. ACM Trans. On Graphics **38**(5), 1–12 (2019)
23. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. IEEE Trans. on NN. and Learn. Sys. **32**(1), 4–24 (2021)
24. Yavuz, E., Yazici, R.: A dynamic neural network model for accelerating preliminary parameterization of 3D triangular mesh surfaces. Neural Computing and Applications **31**(8), 3691–3701 (2019)