3. Parameterization for point cloud spline fitting

In the previous Chapter we have introduced and analyzed the fitting problem of data observations of the form $\mathcal{U} \times \mathcal{P}$ as in (23), where the points $\mathcal{P}$ (1) in $\mathbb{R}^N$ are provided together with their locations $\mathcal{U}$ (3) over a subdomain $\Omega \subset \mathbb{R}^D$. Nevertheless, in many contexts, e. g., real-world applications, the data sites $\mathcal{U}$ are unknown, and the task of approximating a point cloud without data sites results in a complicated non-linear problem.

Therefore, the first fundamental problem of any spline fitting method, commonly addressed as the *parameterization* problem, consists in defining a one-to-one mapping between the points $\mathcal{P}$ and their distinct *parametric* values belonging to a subdomain $\Omega \subset \mathbb{R}^D$. While this *parameterization* process plays a crucial role in the final reconstructed geometry, it is still an open research topic, and several solutions have been proposed that are not optimal in a universal way. Moreover, the complexity of the problem motivates the employment of DL as a viable option for tackling it.

In this Chapter, we propose different *data-driven* parameterization procedures depending on the nature of the input data, i. e. whether they consist of point sequences or point clouds, as well as whether they are organized or scattered. More specifically, deep CNNs have led to improvements in image processing, see e. g., [92, 93, 88], but they also thrive when any kind of data involving spatial-correlated patterns, as for example audio files [151] and geometric data processing [162], have to be processed. Therefore, CNNs are employed for the parameterization learning problem of points on a rectilinear grid, as illustrated in Section 1. On the other hand, CNNs are not suitable to handle scattered data because of the lack of structure among the input data; hence, we propose to employ methods from geometric deep learning to properly address the parameterization problem in this setting. In particular, the parameterization learning problem for unstructured data configurations is solved by means of GCNs, which are a generalization of CNNs to non-Euclidean data, such as discrete manifolds, graphs, and general point clouds, as shown in Section 2 and Section 3. In all the considered examples, the proposed data-driven parameterization methods overcome state-of-the-art standard schemes. The present Chapter is mostly based on [32, 60, 58].

Sofia Imperatore, Eindhoven University of Technology, the Netherlands, s.imperatore@tue.nl, 0009-0003-9116-9978

Referee List (DOI 10.36253/fup_referee_list)
FUP Best Practice in Scholarly Publishing (DOI 10.36253/fup_best_practice)

Book References DOI 10.36253/979-12-215-1002-7.references

3.1. PARCNN: parameterization of gridded data with Convolutional Neural Networks

Gridded data consists of (measurement) values obtained at regularly spaced points that form a rectilinear grid. This kind of geometric data with a regular structure arises in different scientific application settings, such as geographic information systems, meteorology, and medical imaging, when, for example, measurements at regular space or time intervals are acquired through different types of technological instruments. The result is a set of ordered points that can represent curvilinear, surface, volumetric, or even higher-dimensional data. Since the possibility of suitably extending the measurement information to regions where no data are available is often required, efficient parameterization and fitting schemes are usually needed.

In this Section we employ CNNs for the *parameterization* problem to assign parameter values at an arbitrary number of points on a rectilinear grid for their subsequent use in multivariate polynomial spline approximation schemes. We call the proposed model PARamerization with Convolutional Neural Network (PARCNN). We train our model using synthetic data generated by sampling random tensor-product polynomial curves and surfaces by considering univariate and bivariate polynomial least-squares approximations. In our experiments, we also add noise to the test data to simulate measurement errors in the input of the proposed networks. Moreover, we use the resulting parameterization to fit point clouds with non-uniform spline constructions and show that the accuracy of the model properly scales under adaptive mesh refinement. The choice of CNNs allows us to obtain a method that is agnostic to the size of the input point cloud and performs well with an input of a different size from the one considered in the training phase. The experiments highlight that our PARCNN model can also be effective when multivariate adaptive spline fitting with THB-splines is considered.

Over the last decades several approaches have been developed to handle the parameterization of gridded data. Standard methods rely either on a UNIform (UNI) distribution of parameter values over the parametric domain or on scaled configurations of the physical points on the parametric space, as for example the CHord-Length (CHL) or CENtripetal (CEN) parameterizations [95]. Alternatives to these standard methods were presented in the literature, with special focus on the univariate setting. In particular, variants of the scaled parameterizations were proposed in [114, 41], whereas an example of an iterative procedure of different kind can be found in [2]. In addition, the so–called universal parameterization method, closely related to the uniform parameterization, was presented in [26] in the context of univariate and bivariate spline interpolation. The potential of exploiting ML techniques for B-spline curve approximation was originally outlined in [90] and [89] by considering support vector machines and MLPs, respectively. While the focus of [90] was on identifying a suitable knot placement strategy, a scheme based on two interdependent deep neural networks to compute both the knot and parameter values was proposed in [89]. In particular, the neural network used to predict the parameters consists of a standard MLP that takes as input a planar point sequence of fixed length equal to 100 and

returns as output the corresponding parametric values. Therefore, if the given number of data points does not match 100, pre- and post-processing steps for super- or sub-sampling the input data need to be performed. To overcome the computational limitations of this method, the authors in [143] proposed a method for univariate polynomial curve approximation based on a residual neural network. In this case, the network takes as input a planar point sequence of length $2d + 1$, where $d \in \mathbb{N}$ is the polynomial degree. The residual building blocks of the proposed networks are characterized by fully connected layers, which constrain the architecture to deal with an input of fixed size. Consequently, this approach also requires multiple network evaluations and a post-processing step to handle point sequences of arbitrary size.

It should also be noted that, to the best of our knowledge, data-driven methods to address the parameterization problem of gridded data sets in a multivariate setting were not previously proposed. To properly process input datasets with varying dimensions without requiring additional computations, we here propose a deep convolutional approach.

Problem presentation

Let $\mathcal{P}$ be a point cloud, defined in (1) with items in $\mathbb{R}^N$ and characterized by a rectilinear grid structure along $D \leq N$ directions. More precisely, let $h = 1, \dots, D$ be the index associated with the parametric direction, and let m_h be the point sequence length along each direction h. We define the multi-index set

$$\mathbb{I} = \{I = (i_1, \dots, i_D) \mid i_h = 1, \dots, m_h,\ h = 1, \dots, D\},$$

where each multi-index I uniquely identifies a point $\boldsymbol{p}_I$ of the point cloud $\mathcal{P}$. Consequently, the point cloud $\mathcal{P}$ in (1) can be rewritten as

$$\mathcal{P} = \left\{\boldsymbol{p}_I \in \mathbb{R}^N \mid I = (i_1, \dots, i_D),\ i_h = 1, \dots, m_h,\ h = 1, \dots, D\right\}. \quad (44)$$

We denote the cardinality of the point cloud as $m \coloneqq \prod_{d=1}^{D} m_d$. Figure 27 shows an example of a point cloud $\mathcal{P}$, characterized by $D = 2$ parametric directions and elements belonging to $\mathbb{R}^N$, with $N = 3$. The point sequences in the two directions have lengths $m_1 = m_2 = 4$. Each item of the point cloud $\boldsymbol{p}_I$ is uniquely determined by the multi-index $I = (i_1, i_2)$, with $i_1, i_2 = 1, \dots, 4$.

To construct a gridded data approximation scheme, as for example the ones presented in Chapter 2, a parameterization of the point cloud $\mathcal{P}$ is needed. This problem requires to identify a set of suitable parametric values $\mathcal{U}$ as in (3), which can be rewritten as

$$\mathcal{U} = \{\boldsymbol{u}_I \in \Omega \mid I = (i_1, \dots, i_D),\ i_h = 1, \dots, m_h,\ h = 1, \dots, D\}, \quad (45)$$

so that any point $\boldsymbol{p}_I \in \mathcal{P}$ in (44) is associated to the parameter $\boldsymbol{u}_I \in \mathcal{U}$. Moreover, without loss of generality, we set $\Omega = [0, 1]^D$.

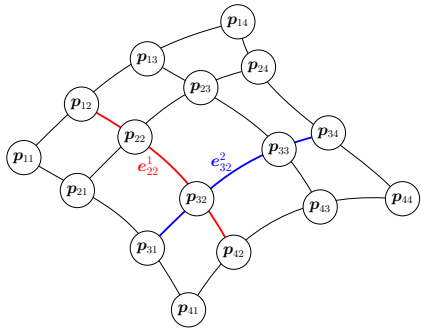


Figura 27. Example of a gridded point cloud with items in $\mathbb{R}^N$, with $N = 3$, and $D = 2$ parametric directions. The feature extraction step derives the edge vectors $\boldsymbol{e}^h_{ij}$ of the input grid.

The identification of a suitable parametrization method for the definition of the set $\mathcal{U}$ of parameter values is a challenging open problem within the polynomial spline fitting framework. In particular, it can naturally influence the quality of the final approximation, leading to sub-optimal approximation error results if not carefully handled; see, see e. g., [48, 52], which address the case of curve interpolation. In the tensor-product B-spline setting, see Section 1, once the univariate parameterization along each parametric direction is computed, the multivariate parameterization can be simply obtained by averaging across all the other parametric directions [128].

The data connectivity information on proximity and distance plays a fundamental role in geometric data processing. Within the data learning framework, modelling the interactions between data relies on their weighted sums, computed via NNs convolutional operators, has become a building block for deep learning architectures to process data with a grid-like topology, for example when reconstructing a free-form model starting from a set of gridded observations $\boldsymbol{p}_I \in \mathbb{R}^N$, for $I \in \mathbb{I}$. Architectures that involve convolutional operators within their layers are addressed as CNNs; see Section 2. The architecture we designed is a *pure* CNN, hence consisting only of convolutional blocks. This choice has been made to take advantage of the locality of convolutional operators in order to be able to support variable input sizes without any additional effort and/or pre- or post-processing of the data. In particular, since the convolutional neural network is characterized by local operators, the particular size of the considered point clouds does not play a fundamental role, whereas the filter dimension does. Consequently, even if a convolutional model is trained on point cloud of fixed size, it can be easily generalised to point cloud of different dimensions. In the following, we detail the proposed architecture, together with information on the data generation, the hyperparameter selection, and the training process.

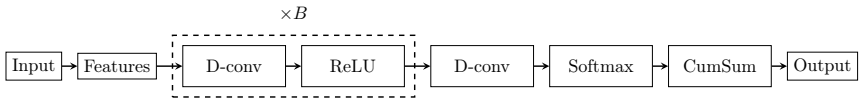


Figura 28. The CNN architecture.

Learning model architecture

The proposed architecture, presented in Figure 28, is a sequential CNN, namely, a progression of layers which consists of a feature extraction layer, B convolutional building blocks (characterized by convolutional layers coupled with ReLU activation functions), a final block of one convolutional layer, the softmax activation function, and a cumulative sum layer connected to the output. We now give a detailed description of each component of our architecture to illustrate how to obtain the output parameterization result.

Input The learning model takes as input a point cloud $\mathcal{P}$ of gridded data, organised in a tensor structure, as in (44).

Features The feature extraction layer produces a relational representation of the point cloud that is translation invariant. It consists in the computation of $m_h - 1$ edges in $\mathbb{R}^N$ between each pair of consecutive points along each direction $h = 1, \dots, D$. Figure 27 illustrates the feature extraction process for a structured point cloud with $N = 3$ and $D = 2$. In particular, the edge associated to the element $\boldsymbol{p}_{22}$ along the direction $h = 1$ is $\boldsymbol{e}^1_{22} \coloneqq \boldsymbol{p}_{32} - \boldsymbol{p}_{22}$ (shown in red in Figure 27), whereas the edge associated to the element $\boldsymbol{p}_{32}$ along the direction $h = 2$ is $\boldsymbol{e}^2_{32} \coloneqq \boldsymbol{p}_{33} - \boldsymbol{p}_{32}$ (shown in blue in Figure 27). More precisely, for any $D \geq 1$, let $I = (i_1, \dots, i_D)$ for $i_h = 1, \dots, m_h - 1$, $h = 1, \dots, D$, and let $I + h$ be the multi-index I augmented by 1 along the h-th direction, for $h = 1, \dots, D$, namely $I + h \coloneqq (i_1, \dots, i_h + 1, \dots, i_D)$. For all $I = (i_1, \dots, i_D)$ with $i_h = 1, \dots, m_h - 1$ and $h = 1, \dots, D$, we define the edges $\boldsymbol{e}^h_I$ as $\boldsymbol{e}^h_I \coloneqq \boldsymbol{p}_{I+h} - \boldsymbol{p}_I$. Note that $\boldsymbol{e}^h_I \in \mathbb{R}^N$. By concatenating the edges $\boldsymbol{e}_I = \left(\boldsymbol{e}^1_I, \dots, \boldsymbol{e}^D_I\right)$, the extracted features are defined as the collection of edges

$$\mathcal{E} = \left\{ \boldsymbol{e}_I \in \mathbb{R}^{N \times D} \mid I = (i_1, \dots, i_D),\ i_h = 1, \dots, m_h - 1,\ h = 1, \dots, D \right\}. \tag{46}$$

The extracted features are then passed as input to the convolutional building blocks.

D-conv The convolutional layers perform a D-dimensional convolution over the input, see e. g., Figure 16 in Chapter 1 for $1D$ and $2D$ convolution examples. Each block is characterized by discrete convolutional operators whose hyperparameters along each direction $h = 1, \dots, D$ are the number of input and output channels $c^h_{in}, c^d_{out} \in \mathbb{N} \setminus \{0\}$, the size of the convolving filter $\mathrm{ks}^h \in 2\mathbb{N} + 1$, the amount of padding

added to the boundaries of the input $\text{pad}^h \in \mathbb{N}$, the stride of the convolutional operator $\text{s}^h \in \mathbb{N} \setminus \{0\}$, and the spacing between the filter elements $\text{dil}^h \in \mathbb{N}$. By denoting with $\boldsymbol{u}^h \in \mathbb{R}^{c_{in}^h \times L_{in}^h}$ the input instance of any convolutional layer along the direction $h = 1, \dots, D$, and with $\boldsymbol{y}^h \in \mathbb{R}^{c_{out}^h \times L_{out}^h}$ its output, the relationship between the input (L_{in}^h) and the output (L_{out}^h) sizes is

$$L_{out}^h = \left\lfloor \frac{L_{in}^h + 2 \cdot \text{pad}^h - \text{dil}^h \cdot \left(\text{ks}^h - 1\right)}{\text{s}^h} + 1 \right\rfloor. \tag{47}$$

In order to fully convert the input along any direction $h = 1, \dots, D$, through the filter of dimension ks^h and obtain an output with the same sequence length, namely $L_{in}^h = L_{out}^h$, we specify $\text{s}^h = 1$ and $\text{dil}^h = 0$ and suitably pad the input of each convolutional layer by adding $\left\lfloor \text{ks}^h/2 \right\rfloor$ elements around the boundary of each item along any direction $h = 1, \dots, D$. In particular, we repeat the first and the last input element (reflect padding mode) $\left\lfloor \text{ks}^h/2 \right\rfloor$ times at the beginning and at the end of each input item, along each direction $h = 1, \dots, D$. Moreover, we set the hyperparameter values of the architecture along each direction $h = 1, \dots, D$ as $\text{ks}^h = 2k_h - 1$, where k_h is the spline order along direction h, $c_{in}^h = m$ (amount of points) in the first convolutional layer and $c_{out}^h = D$ in the last convolutional layer. The input/output channels of the hidden layers are fixed to 100. In particular, their values depends on the user choice, with the only constrain that $c_{out}^h = c_{in}^h$ for two consecutive layers.

ReLU The ReLU is applied element-wise to the output of all except the last convolutional hidden layer. It is defined as $\sigma(z) \coloneqq \max\{0, z\}$ and it is used for practical applications among most feedforward neural networks due both to its simple piecewise linear structure and its high performance [168].

Softmax As a final activation function we apply the softmax function along each direction h to generate parameter intervals that form a partition of unity along each $h = 1, \dots, D$. In particular, the output of the softmax are the elements $\Delta_I^h \in [0, 1]$ associated to each edge $\boldsymbol{e}_I^h$ computed in the feature extraction layer, for $I = (i_1, \dots, i_D)$, with $i_h = 1, \dots, m_h - 1$, $h = 1, \dots, D$.

CumSum In conclusion, the parametric values are recovered by applying the cumulative sum operation along each direction $h = 1, \dots, D$. More precisely, we define the parametric values as

$$\boldsymbol{u}_{(1,i_2,\dots,i_D)}^h = 0, \quad \boldsymbol{u}_{I+h}^h = \boldsymbol{x}_I^h + \Delta_I^h,$$

where $I + h$ is the multi-index I augmented by 1 along the h-th direction, for $h = 1, \dots, D$. Thanks to the application of the softmax

activation function, for each $h = 1, \dots, D$, it is also ensured that the remaining boundary points are mapped to the boundary of the parameter domain, namely $\boldsymbol{u}^h_{(i_1,i_2,\dots,i_{D-1},m_D)} = 1$.

Output The output of the learning model consists of the parametric values $\mathcal{U}$ as defined in (45), associated to the input point cloud $\mathcal{P}$.

Loss function

Once the parameterization as in (45) is computed with the PARCNN model, we estimate the control points of the approximating geometry by solving the least squares minimization problem, see Section 1, in Bernstein-Bézier polynomial form. The reconstructed geometry $\boldsymbol{s} : [0,1]^D \to \mathbb{R}^N$ is then used to defined the loss function of our *unsupervised* learning model.

In particular, let

$$\mathcal{L} : [0,1]^{m \times D} \times \mathbb{R}^{m \times N} \to \mathbb{R}$$

be the MSE between the values of the polynomial approximation model at the *learnt* parameters $\mathcal{U}$, i. e.

$$\widehat{\mathcal{P}} = \left\{ \widehat{\boldsymbol{p}}_I \in \mathbb{R}^N \mid \widehat{\boldsymbol{p}}_I = \boldsymbol{s}(\boldsymbol{u}_I),\ I \in \mathbb{I} \right\}.$$

and the corresponding data $\mathcal{P}$. The loss function is the residual

$$\mathcal{L}\left(\mathcal{U},\ \mathcal{P}\right) \coloneqq \frac{1}{2} \sum_{I \in \mathbb{I}} \| \widehat{\boldsymbol{p}}_I - \boldsymbol{p}_I \|_2^2 , \tag{48}$$

of the least-squares fitting problem (27) in Section 1, computed with the parameters $\mathcal{U}$ in output from the network. Note that our method falls into the unsupervised learning class, since the real parametric values are not considered to guide the optimization problem, hence our input data are unlabelled.

Curve and surface parameterization learning

In the following, we provide the information concerning the training of our models. The method we propose can be implemented for any spatial dimension $N \geq 1$ and parametric dimension $D \leq N$, thanks to the general definition of the convolutional operator. Note that we just set up and train a different model for any distinct pair of values N and D, independently of the considered point cloud. Moreover, since the input processed by the convolutional layers are the edges computed in (46), our parameterization method is affine invariant. Thereby, we focus on two cases: the parameterization of point sequences, when $N = 2$ or $N = 3$ and $D = 1$, and the parameterization of spatial gridded data, when $N = 3$ and $D = 2$, to subsequently compute polynomial and spline approximations in terms of (univariate) curves and (bivariate) surfaces, respectively.

Algorithm 5: Generation of rectilinear point clouds.

Input: Polynomial multi-order $\boldsymbol{k} \in \mathbb{N}^D$, number of samples m_h along each direction $h = 1, \dots, D$.

1 Generate a *random* parametric curve $\boldsymbol{s} : [0,1]^D \to \mathbb{R}^N$, by sampling its coefficients according to the normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$.

2 Sample

$$\{x_{i_h} \in [0,1] : i_h = 1, \dots, m_h\}$$

parameters according to the random uniform distribution $\mathrm{Uniform}(0,1)$, along each direction $h = 1, \dots, D$.

3 Normalize the parametric values x_{i_h} in $[0,1]$, so that $x_1 = 0, x_{m_h} = 1$, i. e.

$$x_{i_h} = \frac{x_{i_h} - \min_{j=1,\dots,m_h} x_j}{\max_{j=1,\dots,m_h} x_j - \min_{j=1,\dots,m_h} x_j}.$$

4 Define a tensor-product mesh of samples $\mathcal{X} \coloneqq \bigtimes_{h=1}^{D} \boldsymbol{x}_h$, namely

$$\mathcal{X} = \left\{ \boldsymbol{x}_I \in [0,1]^D : \boldsymbol{x}_I,\ I = (i_1, \dots, i_D)\,,\ i_h = 1, \dots, m_h,\ h = 1, \dots, D \right\}.$$

5 Evaluate the parametric curve $\boldsymbol{s}$ computed in step 1 on the parametric values $\mathcal{X}$, so that $\boldsymbol{p}_I = \boldsymbol{s}(\boldsymbol{x}_I)$ and collect them in $\mathcal{P}$.

6 Normalize $\mathcal{P}$ in $[0,1]^N$.

Output: Rectilinear point cloud $\mathcal{P}$ in $[0,1]^N$.

Deep learning has its roots in data-driven artificial intelligence and the considered datasets naturally play a fundamental role for the performance of the model. In view of the lack of public datasets for the problem at hand, the points used in the training phase have been randomly generated. We produced multiple synthetic data sets by sampling different kinds of parametric objects, either curves ($D = 1$, $N = 2, 3$) or surfaces ($D = 2$, $N = 3$). The proposed algorithm for data generation is Algorithm 5. The type of the parametric object in step 1 naturally characterizes the generated point cloud. In particular, for $N = 2, 3$ and $D = 1$ we deal with random polynomial curves of fixed degree d in Bernstein-Bézier form, see Remark 4 and 6 in Chapter 1, where the basis $\beta_{j,\boldsymbol{k}}$ for $j = 0, \dots, d$ are defined in terms of Bernstein polynomials, and $\boldsymbol{c}_j \in \mathbb{R}^N$ for $j = 0, \dots, d$ are the corresponding control points sampled from the standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$. For $N = 3$ and $D = 2$, we generate an initial polynomial tensor-product surface with control points $\boldsymbol{c}_j = (c_j^1, c_j^2, c_j^3)^\top$, where $(c_j^1, c_j^2)^\top$ are chosen as the tensor product Greville abscissae for degree d and c_j^3 are sampled randomly accordingly to the uniform distribution in $[0,1]$. The resulting surface is then also randomly rotated.

Training our neural network consists in using our synthetic data to minimize the loss function (48), so that the model weights tend to be

optimal for both seen and unseen input point clouds. In order to optimize the performance of the network, we set up an extensive grid search on optimizers, learning rates, filter sizes, and number of convolutional building blocks when applying the learning parameterization model in the context of planar polynomial curves, namely for $N = 2$ and $D = 1$. In particular, we tested three optimizers (Adam, SGD, RMSprop) with learning rates $\zeta = 1\mathrm{e} - i$, $i = 1, \ldots, 6$. Inspired by [143], the filter size is set to be $2d + 1$, i. e. twice the polynomial degree plus one, and finally the number of building blocks is $B = 4$. In addition, we randomly select the point sequence length of each training batch in the range $m \in [2d + 1, 100]$. We obtained the best results using *RMSprop* as optimizer, learning rate equals to $1\mathrm{e} - 5$, and momentum 0.9 [153]. We then used these hyperparameter configurations also to train the model designed for higher dimensions, in particular for $N = 3$ and $D = 2$ to address the surface fitting problem. In this case we increase B to 5 and we fix the point cloud size to $m = 100$. Note that we train a different model for any value of N and D without a fixed number of steps. By following the so-called early stopping criteria, the training phase continues until the validation loss stagnates, a strategy commonly used to avoid overfitting.

Alternative deep learning parameterization models

Even if the choice of the convolutional parameterization model is naturally inspired by the possibility of considering input data sets of variable dimensions, a comparison of its performance with alternative architectures is needed. We then also consider an MLP, a TRAnsformer encoder (TRA) and a RESidual neural network (RES) with convolutional layers. It should be noted that, except for MLP, both the TRA and RES architectures can also support point sequences of variable lengths. The performance comparison of these architectures presented in Example 1 motivates the choice of the CNN architecture, resulting in the proposed PARCNN model.

The MLP we considered is represented in Figure 29 (left). It consists of a feed forward model with 7 fully connected layers (Linear) alternated by ReLU activation function. Each hidden (fully connected) layer has 100 hidden units, while the first and last ones have $m - 1$ (corresponding to the relative edges of the point sequence) and m units, respectively. The output of the last fully connected layer is sequentially fed to the softmax activation function and the cumulative sum to obtain suitable parameter values.

By following recent advancements in sequence modelling for natural language processing with Bidirectional Encoder Representations from Transformers (BERT) [36], we applied a TRA to our point sequences. One of the most peculiar feature of Transformer is the so-called *Attention mechanism* [158], which introduces competition to select most valuable information from the input sequence. The structure of our transformer is illustrated in Figure 29 (center). In particular, we used a model architecture similar to BERT, which consists of 6 encoder layers with 2048-sized feed-forward layers, dropout probability 10% and N head attentions (instead of 8 as in BERT) to match our data dimension. The transformer takes as input

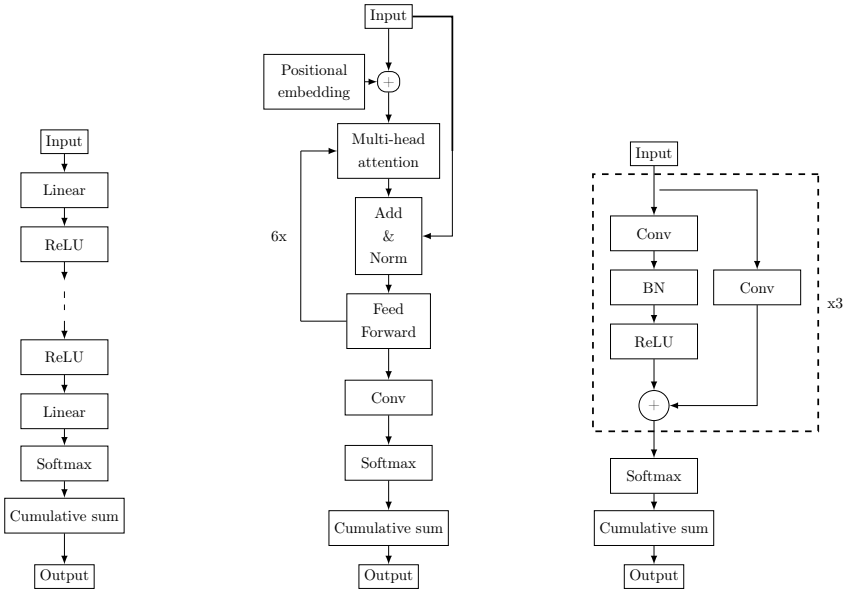


Figura 29. The MLP (left), TRA (center) and RES (right) architectures.

the relative distances of the point sequences and its output is sequentially fed to a convolutional layer, a softmax function and a cumulative sum in order to obtain a suitable parameterization. Even if TRAs, being very over-parametrized models characterized by quadratic time complexity on attention layers with respect to the input length, are more computational expensive than other models, they are capable to handle variable length point sequence like CNNs.

In order to design our convolutional RES, whose configuration is illustrated in Figure 29 (right), we follow the structure of the residual neural networks presented in [143] and [71]. In particular, we assemble 3 convolutional residual blocks, each of them consisting in a feed-forward CNN with 75 (hidden) channels and skip connections, i. e. the 1st layer output has an additional convolutional layer connected with the 4th layer. More precisely, each residual block (represented with the dashed box in Figure 29(c)) has one D-convolutional layer to which the batch normalization and the ReLU follow. Furthermore, there is a skip connection, implemented as a D-convolution with kernel size 1, that takes the current residual block input and sums its output to the convolutional layer output. Softmax activation function and cumulative sum are then applied on the output of the residual blocks. The other convolutional hyperparameters are chosen as in Section 1 for the CNN proposed in Section 1. The motivation for choosing this architecture is to prevent the vanishing gradient problem on very deep architectures during the back-propagation phase, when the chain rule of derivatives can tend to 0 on top layers. Therefore, by adding skip connections we can shorten the distance between the starting layers and

the last one. This architecture has similar advantages and disadvantages to a feed-forward CNN, i.e. can support variable lengths of point sequences but it is computationally slower compared to a feed-forward CNN due to the multiple branches.

Numerical results for PARCNN

In this section we present a selection of experiments to analyze the performance of the introduced learning parameterization model and its generalization capabilities with respect to a variety of structured data and different spline approximation schemes. The quality of the parameterization model is defined in terms of the final accuracy, measured in terms of MAX, MSE or HauSdorff Distance (HSD).

Firstly, a comparison with state-of-the-art neural networks in the case of polynomial data is presented in Section 6 and continues in Section 6, which shows the flexibility of the proposed CNN-based model on input sequences of variable lengths. Trigonometric and noisy data approximation is addressed in Section 6 in the context of adaptive spline curve approximation. The performance of the proposed CNN model on benchmark datasets for spline curve approximation is then illustrated in Section 6.

By moving to the surface case, Section 6 shows the results on polynomial data, together with the model robustness to noise. Subsequently, in Section 6, we provide numerically evidence of the generalization capabilities of the proposed parameterization model with respect to datasets significantly different (in nature and size) from the synthetic data used during the training phase. Finally, in Section 6 we investigate the performance of the learning parameterization model for tensor-product B-spline least squares fitting scheme.

Architectures comparison

In this first example, we analyze the different learning parameterization models illustrated in the previous sections. More specifically, we compare our convolutional model (CNN) with the three deep learning models (MLP, TRA, RES) of Section 6 as well as with the results reported in [143, Table 1], i. e. Parameterization for Polynomial curve Network (PPN). In particular, we assembled all the new networks in order to have about the same number of learnable parameters (#LP) as PPN. As additional term of comparison, for each considered metric we also report the best value obtained with the STandarD (STD) schemes of the form

$$u_{i+1} = u_i + \frac{\Delta \boldsymbol{p}_i}{\Delta \boldsymbol{p}} \quad \text{for} \quad i = 1, \dots, m-1,$$

with

$$u_0 = 0, \quad \Delta \boldsymbol{p}_i := \|\boldsymbol{p}_{i+1} - \boldsymbol{p}_i\|^{\alpha}, \quad \Delta \boldsymbol{p} = \sum_{i=1}^{m-1} \Delta \boldsymbol{p}_i,$$

among UNI, CHL and CEN parameterizations, that can be recovered by setting $\alpha = 0, 1, 1/2$ respectively.

Tabella 1. Comparison of the different parameterization methods on polynomial data sets of $2d+1$ points for $d = 2, 3, 4, 5$ in Section 6. The learnable parameters (#LP) for the different neural networks are also shown.

	STD	MLP	TRA	RES	PPN	CNN
$d = 2$						
#LP		5.19e + 4	6.17e + 4	3.02e + 4	5.45e + 4	2.31e + 4
MSE	2.74e − 3	3.78e − 5	1.58e − 2	2.82e − 4	7.8e − 5	**2.22e − 5**
MAX	4.62e − 2	7.60e − 5	2.60e − 2	5.56e − 4	1.0e − 2	**4.37e − 5**
HSD	6.17e − 2	7.50e − 3	1.78e − 1	2.27e − 2	8.2e − 3	**6.33e − 3**
$d = 3$						
#LP		5.25e + 4	6.17e + 4	4.19e + 4	5.47e + 4	3.23e + 4
MSE	1.86e − 3	1.17e − 4	9.94e − 3	3.40e − 4	1.3e − 4	**4.55e − 5**
MAX	6.09e − 2	2.83e − 4	1.53e − 2	6.63e − 4	1.5e − 2	**1.23e − 4**
HSD	5.88e − 2	1.45e − 2	1.51e − 1	2.69e − 2	1.2e − 2	**9.73e − 3**
$d = 4$						
#LP		5.31e + 4	6.17e + 4	5.36e + 4	5.50e + 4	4.15e + 4
MSE	8.96e − 4	1.13e − 4	6.69e − 3	2.48e − 4	1.4e − 4	**2.72e − 5**
MAX	2.11e − 2	2.87e − 4	1.07e − 2	5.32e − 4	1.7e − 2	**5.44e − 5**
HSD	4.47e − 2	1.58e − 2	1.31e − 1	2.50e − 2	1.4e − 2	**8.05e − 3**
$d = 5$						
#LP		5.37e + 4	6.17e + 4	6.52e + 4	5.52e + 4	5.07e + 4
MSE	4.75e − 4	1.60e − 4	4.94e − 3	2.09e − 4	1.5e − 4	**2.22e − 5**
MAX	1.16e − 2	6.52e − 4	9.49e − 3	3.63e − 4	1.9e − 2	**6.32e − 5**
HSD	3.41e − 2	1.96e − 2	1.17e − 1	2.41e − 2	1.6e − 2	**7.67e − 3**

In all the required cases, the filter dimension is set to $2d + 1$. We trained MLP, TRA, RES and CNN on polynomial point sequences of length $m = 2d + 1$ generated with the algorithm reported in Section 1, and performed the tests on 100 data sequences of the same kind of the training set for degrees $d = 2, \ldots, 5$. The results in terms of averaged MSE, MAX and Direct Hausdorff Distance (DHD) are reported in Example 1 together with #LP. The number of input units and the kernel size increase with the degree and, consequently, also the number of learnable parameters changes. As concerns TRA, the change is not evident within the first 3 significant digits, since the encoder architecture, which is the prevalent part of this network, is not affected by the input dimension and the kernel size. We can note that, except for TRA, the learning parameterization models outperform the standard parameterization techniques by gaining up to two (RES, PPN) or three (MLP, CNN) order of accuracy with respect to the different metrics. The best error results are highlighted in bold, in particular it should be noted that the CNN model always scores the lowest approximation errors, while also having the smallest amount of #LP.

Tabella 2. MSE error for Bézier approximation of degree $d = 2, 3, 4, 5$ on polynomial data for different input lengths $m = 20, 50, 80$ in Section 6.

	$m = 20$			$m = 50$			$m = 80$		
d	STD	PPN	CNN	STD	PPN	CNN	STD	PPN	CNN
2	2.97e − 3	5.3e − 4	**3.21e − 5**	1.79e − 3	7.3e − 4	**2.23e − 5**	1.46e − 3	7.9e − 4	**4.13e − 5**
3	2.20e − 3	6.7e − 4	**1.31e − 4**	1.43e − 3	8.7e − 4	**8.80e − 5**	1.41e − 3	8.5e − 4	**1.09e − 4**
4	1.66e − 3	3.3e − 4	**7.24e − 5**	1.06e − 3	4.3e − 4	**5.30e − 5**	9.48e − 4	4.1e − 4	**6.13e − 5**
5	1.35e − 3	4.8e − 4	**3.71e − 5**	7.65e − 4	5.8e − 4	**2.50e − 5**	5.09e − 4	5.6e − 4	**3.55e − 5**

Polynomial curve approximation

In this experiment, we exploit the PARCNN model on univariate polynomial data for different degrees d and variable point sequence lengths m. For each degree $d = 2, \ldots, 5$, and length $m = 20, 50, 80$, we generate 100 polynomial point sequences $\mathcal{P} = \left\{\boldsymbol{p}_i \in \mathbb{R}^2 \mid i = 1, \ldots, m\right\}$ as detailed in Section 1. Subsequently, we compute their parameterization $\mathcal{U} = \{u_i \in [0, 1] \mid i = 1, \ldots, m\}$ with the proposed CNN-based approach and with standard parameterization schemes. We then compare the averaged MSE obtained building the Bézier fitting model at the parametric values given by the best standard technique (STD), the results presented in [143, Table 2] (PPN), and the ones obtained with our convolutional model (CNN). The outcome of this analysis is reported in Table 2, where the best approximation errors are highlighted in bold. The CNN outperforms STD methods and the PPN model in terms of MSE. In particular, the MSE obtained with CNN is reduced up to one and two orders of magnitude as regards PPN and STD, respectively. The advantage of our PARCNN model lies not only in the better performance shown in Example 2, but also in the novelty of its self-contained nature. For each input point sequence of any length, our convolutional learning method directly computes the corresponding point parameterization, without any additional computation. The approach proposed in [143] instead requires multiple network evaluations and a post processing step to handle arbitrary sequence lengths.

Spline curve approximation of trigonometric and noisy datasets

In this example we investigate the generalization capabilities of our convolutional parameterization model to different datasets and adaptive B-spline curve fitting of various degrees ($d = 2, 3, 4, 5$), as well as its robustness to noise. By considering the algorithms presented in Section 1, we generate two distinct datasets of size 100 by sampling trigonometric and polynomial curves. The trigonometric curves are defined as

$$\boldsymbol{c}(t) = \boldsymbol{a}_0 + \sum_{j=1}^{r} \boldsymbol{a}_j \cos(jt) + \sum_{j=1}^{r} \boldsymbol{b}_j \sin(jt),$$

for a fixed degree r, where $\boldsymbol{a}_0, \boldsymbol{a}_j, \boldsymbol{b}_j \in \mathbb{R}^2$ are values sampled from the standard normal distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{1})$, for $j = 0, 1, \ldots, r$. In the polynomial case, we added a factor of $1\mathrm{e} - 2$ random Gaussian noise to the data points.

Tabella 3. MSE for adaptive B-spline approximation with 5 interior knots of degree $d = 2, 3, 4, 5$ on trigonometric data (top) and polynomial data with random Gaussian noise (bottom) for different input lengths $m = 20, 50, 80$ in Section 6.

	$m = 20$		$m = 50$		$m = 80$	
d	STD	CNN	STD	CNN	STD	CNN
2	3.01e − 4	**1.56e − 5**	2.17e − 4	**6.08e − 6**	1.71e − 4	**5.82e − 6**
3	1.35e − 4	**1.63e − 5**	1.10e − 4	**1.05e − 5**	3.16e − 5	**7.84e − 6**
4	1.74e − 4	**5.53e − 6**	2.48e − 5	**3.66e − 6**	3.14e − 5	**4.74e − 6**
5	3.09e − 5	**2.54e − 6**	1.17e − 5	**1.88e − 6**	4.03e − 5	**1.92e − 6**

	$m = 20$		$m = 50$		$m = 80$	
d	STD	CNN	STD	CNN	STD	CNN
2	1.27e − 4	**7.69e − 5**	1.32e − 4	**9.46e − 5**	1.54e − 4	**1.02e − 4**
3	9.54e − 5	**3.36e − 5**	1.21e − 4	**6.33e − 5**	1.12e − 4	**7.24e − 5**
4	7.37e − 5	**2.68e − 5**	1.66e − 4	**8.89e − 5**	8.41e − 5	**4.04e − 5**
5	9.02e − 5	**2.72e − 5**	1.68e − 4	**7.64e**−5	1.04e − 4	**6.11e − 5**

We test the CNN parameterization by considering adaptive B-spline least-squares approximation obtained by performing dyadic refinement of the knot intervals which exhibit the highest error values. The resulting MSE is shown in in Table 3 for trigonometric data of degree $r = 5$ (top) and noisy polynomial data (bottom), for different input sequence lengths ($m = 20, 50, 80$) and adaptive B-spline approximations with a final number of 5 interior knots. The proposed CNN parameterization model always obtains better results, highlighted in bold, than the best standard methods (STD). In particular, in the case of trigonometric data (Table 3 (top)), it gains up to two order accuracy. For the results on noisy polynomial data (Table 3 (bottom)), the accuracy obtained with CNN model is improved up to one order of magnitude when compared to STD.

To better understand the generalization capabilities of the network when an increasing number of knots is considered, we present a final test with fixed input length equals to $m = 20$. We then perform an adaptive B-spline curve fitting with dyadic refinement, starting from the polynomial case of 0 internal knots up to 10 internal knots. The results in terms of MSE for the CEN, CHL, UNI and CNN parameterizations for $d = 3, 4, 5$ are shown in Figure 30 (top) for trigonometric data of degree $r = 5$ and in Figure 30 (bottom) for noisy polynomial data. We may observe that in Figure 30 (top) for $d = 3, 5$ the MSE error gap between CNN and STD is maintained or even increased with respect to the number of interior knots inserted. In Figure 30 (top) for $d = 4$ and in Figure 30 (bottom) for all the degrees, the MSE error gap between CNN and STD tends to reduce with the number of interior knots inserted, but the CNN parameterization model always achieves the best results.

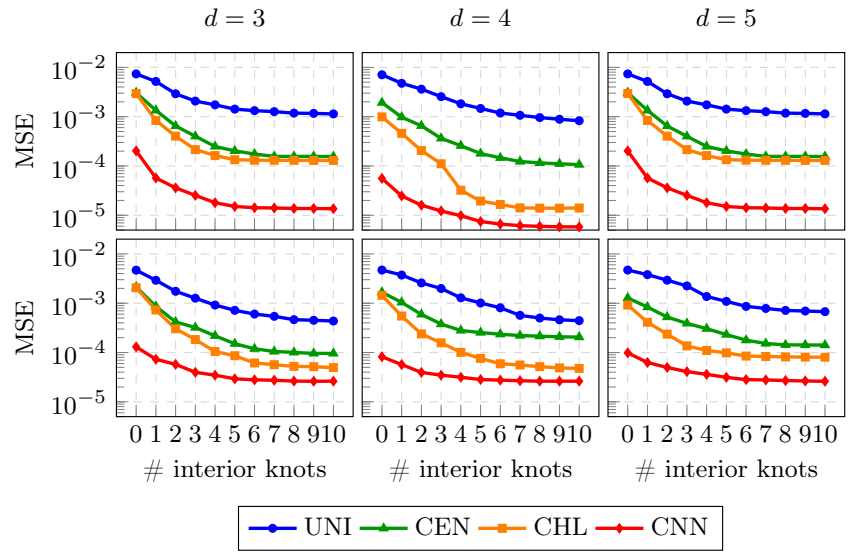


Figura 30. MSE for adaptive B-spline approximations with increasing number of knots tested on $m = 20$ trigonometric (top) and noisy (bottom) data for degree $d = 3$ (left), $d = 4$ (center) and $d = 5$ (right) in Section 6.

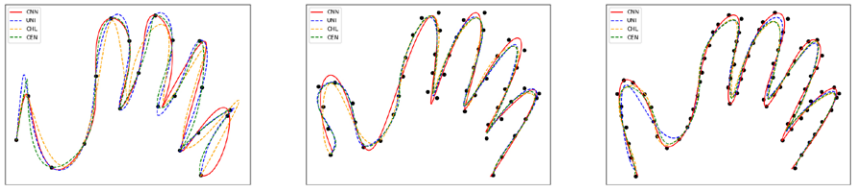


Figura 31. B-spline approximation of degree $d = 4$ with 15 interior knots for 20 (left), 50 (center), and 80 points (right) in Section 6.

Spline curve approximation of benchmark datasets

In this experiment, we test our parameterization model on point sequences sampled by the benchmark presented in [124]. By considering B-spline approximation of degree $d = 4$ with at most 15 adaptive interior knots, Figure 31 presents the results obtained employing the standard (UNI, CHL, CEN) and the CNN parameterizations on an increasing number of input points, namely $m = 20, 50, 80$. An analogous comparison on a sequence of $m = 65$ points, approximated with degree $d = 5$ B-splines and adaptive knot vectors with a varying number (4, 7, and 10) of interior knots is shown in Figure 32.

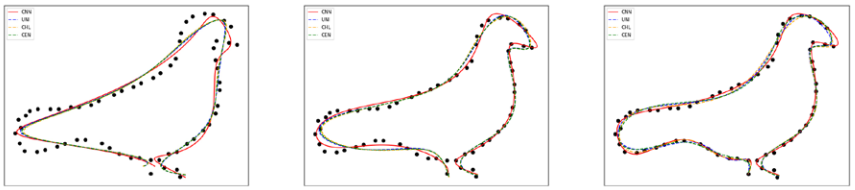


Figura 32. B-spline approximation of 65 points with $d = 5$ and 4 (left), 7 (center), 10 (right) interior knots in Section 6.

Tabella 4. MSE error for polynomial least-squares approximation of polynomial (left) and noisy (right) data of bi-degree $= (d, d)$, for $d = 2, 3, 4, 5$ in Section 6.

	exact		noisy	
d	STD	CNN	STD	CNN
2	1.20e − 3	**5.21e − 4**	1.26e − 3	**5.94e − 4**
3	6.07e − 4	**2.25e − 4**	6.45e − 4	**2.57e − 4**
4	3.91e − 4	**1.46e − 4**	4.62e − 4	**1.94e − 4**
5	2.19e − 4	**8.97e − 5**	2.75e − 4	**1.35e − 4**

Polynomial surface approximation

In this experiment we illustrate the performance of the proposed parameterization model on both exact and noisy data sampled from polynomial patches in the bivariate case. In particular, we collect structured point clouds by sampling Bézier surfaces of bi-degree $\boldsymbol{d} = (d, d)$. For each $d = 2, 3, 4, 5$, we generate 100 point clouds consisting of $m = 144$ gridded points following the algorithm described in Section 1. In addition, we also create a noisy version of this dataset by perturbing each point cloud with Gaussian noise, considering a factor $\epsilon = 5\mathrm{e}-3$. The results of the parametric polynomial fitting of bi-degree $\boldsymbol{d} = (d, d)$ of the aforementioned generated point clouds are reported in Table 4 in terms of MSE for the best standard parameterization technique (STD), among UNI, CHL and CEN, and the proposed convolutional model (CNN). The best approximation MSE values are highlighted in bold. The errors give numerically evidence of the generalization capabilities of the trained model with respect to datasets which are somehow similar but at the same time independent from the ones used in the training phase. In particular, the proposed model avoids potential over-fitting problems, it is robust to noise, and it always increases the accuracy of the polynomial approximation more than 50% over STD parameterization techniques in all the considered configurations.

Polynomial approximation of geometric models

In this experiment, we analize the generalization capabilities of the parameterization learning model for data which are different both in dimension and nature from the one used in the training phase. More precisely, we sample

Tabella 5. MSE for polynomial least-squares surface approximation with different bi-degrees $\boldsymbol{d} = (d, d)$ of the car shell, the ship hull, and the wind turbine point clouds of size $m = 1089$ and $m = 3025$ in Section 6.

		car		ship hull		wind turbine	
	d	GT	CNN	GT	CNN	GT	CNN
$m = 1089$	2	1.38e − 3	**8.28e − 4**	2.53e − 4	**1.11e − 4**	1.95e − 3	**1.30e − 3**
	3	4.58e − 4	**1.94e − 4**	7.28e − 5	**6.01e − 5**	1.18e − 4	**3.94e − 5**
	4	1.19e − 4	**7.63e − 5**	2.78e − 5	**1.88e − 5**	4.21e − 5	**3.54e − 5**
	5	7.82e − 5	**6.73e − 5**	**5.89e − 6**	1.19e − 5	2.67e − 5	**1.38e − 5**
$m = 3025$	2	1.32e − 3	**8.10e − 4**	2.34e − 4	**8.44e − 5**	1.90e − 3	**1.55e − 4**
	3	4.42e − 4	**2.03e − 4**	6.76e − 5	**4.95e − 5**	1.14e − 4	**7.63e − 5**
	4	1.14e − 4	**5.54e − 5**	2.62e − 5	**2.05e − 5**	**4.14e − 5**	5.55e − 5
	5	7.47e − 5	**5.13e − 5**	**5.67e − 5**	1.17e − 4	2.62e − 5	**2.14e − 5**

point clouds of dimension either $m = 1089$ or $m = 3025$ from 3 different B-spline geometries of bi-degree $\boldsymbol{d} = (d, d)$, representing a car, a ship hull, and a wind turbine. The corresponding point clouds are illustrated in Figure 33 for the case of 1089 (top) and 3025 (bottom) samples. Moreover, for this experiment we have also stored the true parametric values $\mathcal{X}$ using during the sampling phase and we will refer to them as the ground truth (GT) in the comparisons. Once the data have been parametrized with the convolutional method (CNN), we performed polynomial approximation of bi-degree $\boldsymbol{d} = (d, d)$ and compare the MSE and MAX obtained with GT and CNN parameterizations. Note that the GT parameters come from a specific B-spline geometry with different degrees and knot lines configurations, therefore if used for different spline models they not necessarily lead to zero error at the evaluation sites.

The results are reported in Table 5, where the best approximation values are highlighted in bold. They numerically show the generalization capabilities of the proposed model with respect to the size of the input point clouds and to the nature of the data. For the car geometry, in all the considered configurations, the MSE error of the final reconstructed polynomial approximation is smaller in the case of the CNN parameterization, with a gain in accuracy up to more than 50% for $d = 3$. When the ship hull geometry is considered, the best performances are achieved if the CNN parameterization is applied in combination with lower polynomial bi-degrees $\boldsymbol{d} = (d, d)$, namely $d = 2, 3$, and for $d = 2$ the final reconstructed geometry of the bigger point cloud gains more than 60% of accuracy. As concerns the final wind turbine geometry, the best advantage is registered on the point cloud of size $m = 1089$ among all the different bi-degrees, where the MSE is improved up to more than 60% for bi-degree $\boldsymbol{d} = (3, 3)$. The same bi-degree $\boldsymbol{d} = (3, 3)$ leads also to the best error results for the bigger wind turbine point cloud.

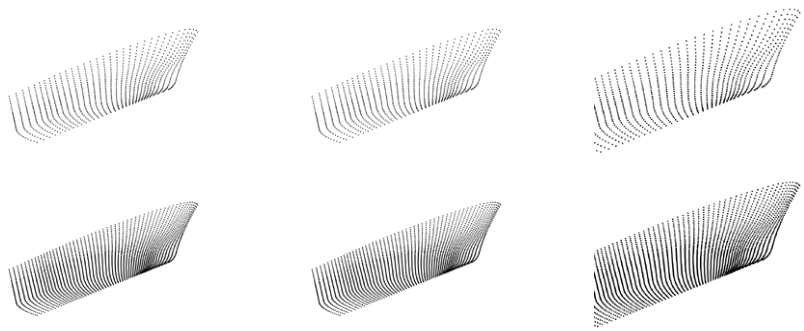


Figura 33. Point clouds obtained from a car (left), a ship hull (center), and a wind turbine (right) model of 1089 (top) and 3025 (bottom) points.

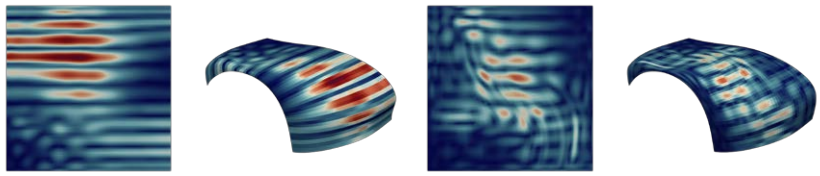
Tensor product B-splines approximation

In this example we show the generalization capabilities of the convolutional parameterization for tensor product B-spline models. In particular, we consider the point clouds obtained from the car shell geometry introduced in Section 6 for which we perform a tensor product B-spline approximation of bi-degree $\boldsymbol{d} = (3, 3)$ with 3 levels of uniform refinement, both considering the given parameterization (GT) and the convolutional parameterization (CNN).

Concerning the point cloud of dimension 1089, the MSE error registered using GT parameterization is $8.18\text{e} - 6$. If the CNN parameterization is considered, the MSE is more than halved and reduces to $3.54\text{e}{-6}$. Analogous results are achieved when considering the point clouds of dimension 3025 for which the GT parameterization leads to MSE of $8.21\text{e} - 6$, whereas by using the CNN parameterization the accuracy improves more than 40% obtaining an MSE equals to $4.31\text{e} - 6$.

The scaled error distributions of the final approximation plotted on the parametric and physical domain, for both GT and CNN parameterizations, are illustrated in Figure 34 for the point cloud of size 3025. More precisely, plots (a) and (b) in Figure 34 are the error distributions for the GT parameterization, whereas the corresponding results for the CNN parameterization are shown in plots (c) and (d) of the same figure.

3.2. PARGCN: parameterization of scattered data with Graph Convolutional neural Networks



(a) (b) (c) (d)

Figura 34. Point-wise error on the parametric (a-c) and geometric (b-d) domain of the tensor product B-spline approximation of bi-degree $\boldsymbol{d} = (3, 3)$ for the car shell point cloud of size 3025 using the given GT (a-b) and the CNN (c-d) parameterizations in Section 6.

The standard paradigms of deep learning have been designed in order to process data characterized by a regular multiple array structure, as for the rectilinear point clouds analyzed in the previous Section. However, CNNs are not suitable for applications whose data has a graph or mesh structure or is completely unorganized, e. g., 3D shapes, chemical molecules, social/relational networks, citation networks, scattered point clouds, among others. In order to handle this kind of data, a new learning theory and related architectures have been developed.

GCNs have become the counterpart of CNNs to process data belonging to discrete manifolds, graphs, or general point clouds, and many different approaches to defining convolutional operators on graph domains have been proposed [167]. In particular, the translation of standard filter operators to graph operators relies on suitable aggregations of vertex and neighbour features; see for more details Section 2. GCNs are employed in the next two Sections, i. e. Section 2 and Section 3, to address the parameterization problem of scattered point clouds.

In this Section, we propose a deep learning approach called PARGCN for parameterizing an *unorganized* or *scattered* point cloud in $\mathbb{R}^3$ with GCNs. The proposed parameterization learning model builds upon a GCN that predicts the weights (called *parameterization weights*) of certain convex combinations that lead to a mapping of the physical points into a planar parameter domain, after solving a sparse linear system. This is a novel learning approach that goes beyond closed-form heuristic choices of the parameterization weights [51], thus introducing a geometry-informed learning procedure that produces parameterizations of high quality.

While graph neural networks have been successfully applied to classification tasks such as shape recognition, segmentation and registration [164, 135, 69, 147], regression tasks where the network should predict a (potentially vector-valued) function on the input geometry are much less studied [5, 40, 60]. One difficulty is that GCNs aim to perform classification, segmentation or regression on the vertices $\mathcal{V}$ of a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. However, we are interested in performing regression on the edges $\mathcal{E}$ instead of the vertices $\mathcal{V}$ of $\mathcal{G}$. To overcome this problem, given the directed graph $\mathcal{G}$, we

extract its line graph (dual-graph) [70, 66], which represents the adjacencies between edges of the original graph. The line graph is used as input to a geometry-informed graph convolutional neural network trained to learn optimal parameterizations.

Meshless barycentric parameterization

Let $\mathcal{P}$ be a scattered point cloud, as in (1). The adjective *scattered* usually indicates that the points do not have a regular structure. Thereby, the only assumption we make consists in assuming that the unorganized point cloud $\mathcal{P}$ can be partitioned into two disjoint subsets, i. e. $\mathcal{P} = \mathcal{P}_I \dot{\cup} \mathcal{P}_B$, with

$$\mathcal{P}_I = \left\{ \boldsymbol{p}_i \in \mathbb{R}^3 \mid i = 1, \ldots, n \right\}, \tag{49}$$

the set of interior points, and

$$\mathcal{P}_B = \left\{ \boldsymbol{p}_i \in \mathbb{R}^3 \mid i = n+1, \ldots, m \right\}, \tag{50}$$

the set of boundary points. Our objective consists in finding a suitable parameterization $\mathcal{U}$ as in (3) for the unorganized data in $\mathcal{P}$.

In [51], the authors propose a method for parameterizing an unstructured point cloud over a convex polygonal domain $\Omega \subset \mathbb{R}^2$. It consists of two steps:

1. *Boundary point cloud parameterization.* The boundary points $\mathcal{P}_B$ in (50) are parameterized over the boundary of the parameter domain Ω, resulting in parameters

$$\mathcal{U}_B = \{ \boldsymbol{u}_i \in \partial\Omega \mid i = n+1, \ldots, m \}, \tag{51}$$

that lie in anticlockwise order on $\partial\Omega$.

2. *Interior point cloud parameterization.* A directed graph $\mathcal{G}$, whose vertices are the points in $\mathcal{P}$, is determined. The parameters that correspond to the points $\mathcal{P}_I$ in (49) are assumed to be convex combinations of their neighbours in the graph with certain weights. This leads to a sparse linear system whose solution is the parameterization of $\mathcal{P}_I$, i. e.

$$\mathcal{U}_I = \{ \boldsymbol{u}_i \in \Omega \mid i = 1, \ldots, n \}. \tag{52}$$

As far as the first step of the method is concerned, several approaches are available for the parameterization of point sequences, as already discussed in Section 1. We then assume that the parametric values $\mathcal{U}_B$ in (51) corresponding to the boundary points $\mathcal{P}_B$ in (50) are known.

In the second step, for each point in $\boldsymbol{p}_i \in \mathcal{P}_I$, $i = 1, \ldots, n$, a neighbourhood $N_i \subset \{1, \ldots, m\} \setminus \{i\}$ is determined. In particular, we use the ball neighbourhood

$$N_i = \{ j \in \{1, \ldots, m\} : 0 < \|\boldsymbol{p}_i - \boldsymbol{p}_j\| < r \}, \quad i = 1, \ldots, n, \tag{53}$$

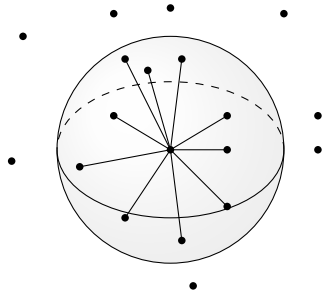


Figura 35. A radius neighbourhood.

for some radius $r > 0$, as illustrated in Figure 35. Choosing 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 $j \in N_i$. 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 [3] from $\mathcal{P}_I$ into $\mathcal{P}$ or, for short, the *radius graph* of the points $\mathcal{P}$.

The unknown parameter $\boldsymbol{u}_i$ for each point $\boldsymbol{p}_i \in \mathcal{P}_I$ is assumed to be a convex combination

$$\boldsymbol{u}_i = \sum_{j \in N_i} \lambda_{ij} \, \boldsymbol{u}_j, \quad i = 1, \ldots, n, \tag{54}$$

of its neighbours in $\mathcal{G}$, where the *parameterization weights* fulfill $\lambda_{ij} > 0$ and $\sum_{j \in N_i} \lambda_{ij} = 1$. Figure 36 shows an example for two interior points $\boldsymbol{p}_i, \boldsymbol{p}_k \in \mathcal{P}_I$ (in blue) and their corresponding neighbourhoods N_i and N_k, where we identify the vertices in $\mathcal{G}$ with the corresponding points in $\mathcal{P}$.

We collect the linear equations (54) in a linear system

$$A \, U_I = B, \tag{55}$$

where A is the sparse $n \times n$-matrix

$$A_{ij} = \begin{cases} 1 & \text{if } i = j, \\ -\lambda_{ij} & \text{if } j \in N_i, \\ 0 & \text{otherwise,} \end{cases}$$

B is the $n \times 2$ matrix $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)^\top$ with

$$\boldsymbol{b}_i = \sum_{\{1,\ldots,n\} \cap N_i} \lambda_{ij} \boldsymbol{u}_j$$

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.

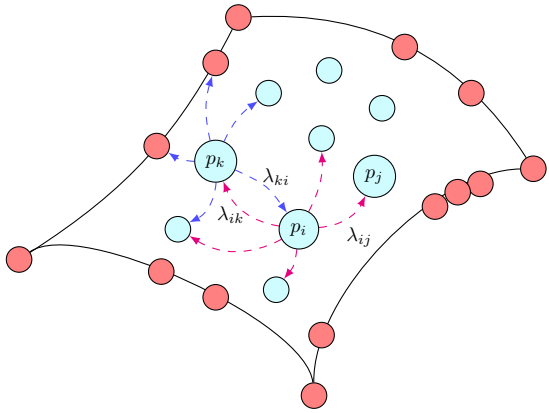


Figura 36. Examples of two directed neighbourhoods for the interior points $\boldsymbol{p}_i$ and $\boldsymbol{p}_k$, for any $i, k = 1, \ldots, n$. The interior points $\mathcal{P}_I$ in (49) are cyan, while the boundary points $\mathcal{P}_B$ in (50) are red.

In order to guarantee that the matrix A in (55) has full rank, the choice of the radius r is crucial. As proved in [51, 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.

What remains to be determined is the specific choice of the parameterization weights λ_{ij} for $i = 1, \ldots, n$ and $j \in N_i$. The latter determine the parameterization of $\mathcal{P}_I$ and it is therefore important to choose them carefully in order to obtain good results. The particular choice of λ_{ij} is still open, while some heuristics have been proposed. More precisely, in [51] two heuristic choices are suggested: UNIForm parameterization weights (UNIF) and RECiprocal Distance parameterization weights (RECD). The uniform weights aim at generalizing the concept of uniform parameterization of curves to surfaces, hence the parameterization weights are simply chosen as

$$\lambda_{ij} = \frac{1}{|N_i|}.$$

The reciprocal distance weights are meant to generalize the concept of chord-length parameterization to surfaces, thus, the weights are chosen as

$$\lambda_{ij} = \frac{\|\boldsymbol{p}_j - \boldsymbol{p}_i\|^{-1}}{\sum_{k \in N_i} \|\boldsymbol{p}_k - \boldsymbol{p}_i\|^{-1}}.$$

A third choice that is proposed in [51] are the Local Projection Shape-Preserving parameterization weights (LPSP), which are not defined on the radius graph. Instead, for all interior points $\boldsymbol{p}_i$ the neighbourhood (53) is projected onto its best-approximating plane. The resulting planar point cloud is then triangulated by a Delaunay triangulation [34, 67]. The

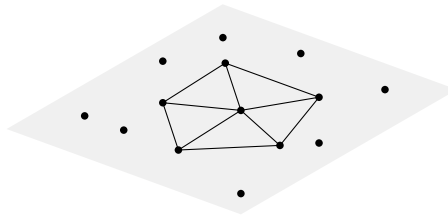


Figura 37. A Delaunay neighbourhood.

neighbours of $\boldsymbol{p}_i$ in the Delaunay triangulation are its neighbours in the *local projection* graph $\mathcal{G}$, see Figure 37. The corresponding weights are found based on the shape preserving weights for triangulated surfaces presented in [49]. For each interior point $\boldsymbol{p}_i$, the neighbours are ordered counter-clockwise and intermediate local parameters $\tilde{\boldsymbol{u}}$ are determined such that for all $j \in N_i$

$$\|\tilde{\boldsymbol{u}}_i - \tilde{\boldsymbol{u}}_j\| = \|\boldsymbol{p}_i - \boldsymbol{p}_j\|$$

and

$$\angle(\tilde{\boldsymbol{u}}_j, \tilde{\boldsymbol{u}}_i, \tilde{\boldsymbol{u}}_{j+1}) = \frac{2\pi \angle(\boldsymbol{p}_j, \boldsymbol{p}_i, \boldsymbol{p}_{j+1})}{\sum_{k,(k+1)\in N_i} \angle(\boldsymbol{p}_k, \boldsymbol{p}_i, \boldsymbol{p}_{k+1})},$$

i. e. the lengths and the angle ratios around $\boldsymbol{p}_i$ are preserved. Finally, corresponding weights λ_{ij} are computed from the parameters $\tilde{\boldsymbol{u}}_j$. For details about the implementation we refer to [49].

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, i. e.

$$\lambda_{ij} > 0, \quad \sum_{j \in N_i} \lambda_{ij} = 1, \quad \text{for} \quad i = 1, \ldots, n, \quad \text{and} \quad j \in N_i.$$

Consequently, 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.

In the following, we present the details of our neural network based method for predicting the optimal parameterization weights λ_{ij} in (54). Our method takes as input an unstructured point cloud with parameterized boundary curves. The feature extraction step proceeds as follows. First, the radius graph associated to the unstructured point cloud is computed, and then its line graph, together with appropriate vertex features suitably identified as described in Section 2, is derived. The line graph is then given as input to our graph convolutional neural network, detailed in Section 2. The output of the network is a set of parameterization weights λ_{ij}, for any interior point, $i = 1, \ldots, n$ and neighbouring point, $j \in N_i$. Those are mapped back to the radius graph, and finally, a system of the form (55) is solved to obtain the parametric values.

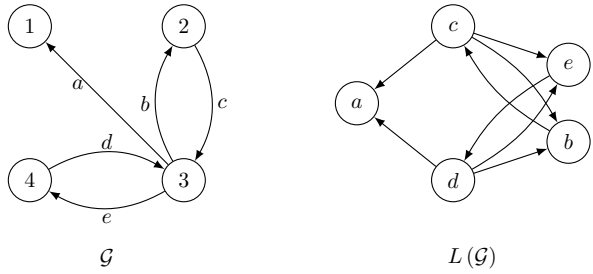


Figura 38. Examples of a directed graph $\mathcal{G}$ (left) and its corresponding directed line graph $L(\mathcal{G})$ (right).

Preprocessing and feature extraction

As a preprocessing step, the input point cloud is normalized by translation and scaling so that it is contained inside the unit cube $[0,1]^3$. As outlined in Section 2, the interior points in $\mathcal{P}_I$ are parameterized using neighborhood relationships in a directed graph $\mathcal{G}$, which is the radius graph from $\mathcal{P}_I$ into $\mathcal{P}$, based on the ball neighborhoods (53). Therefore, predicting optimal parameterization weights corresponds to predicting edge weights on the directed graph. However, while a number of graph neural network architectures can incorporate given *edge weights* in their convolution, their predictions live on the vertices of the graph. In order to predict an output on the directed edges of $\mathcal{G}$, we transform $\mathcal{G}$ into a *line graph* $L(\mathcal{G})$ whose vertices are the directed edges of $\mathcal{G}$. The formal definition of the line graph follows.

Definition 12. *The line graph of a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is the directed graph $L(\mathcal{G}) = (\mathcal{V}', \mathcal{E}')$ whose vertex set $\mathcal{V}'$ corresponds to the directed edges $\mathcal{E}$. $\mathcal{E}'$ contains all directed edges from a vertex $(i,j) \in \mathcal{V}'$ to a vertex $(k,\ell) \in \mathcal{V}'$ such that $j = k$, $j, k \in \mathcal{V}$.*

An example of a directed graph $\mathcal{G}$ and its corresponding directed line graph $L(\mathcal{G})$ is shown in Figure 38.

In order to define the features on the vertices $L(\mathcal{G})$ that the network will process, we define features on the edges of $\mathcal{G}$ and transfer them to the vertices of $L(\mathcal{G})$. For each edge of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, i.e. $(i,j) \in \mathcal{E}$, we define the edge feature

$$\boldsymbol{e}_{ij} \coloneqq (\boldsymbol{p}_i, \boldsymbol{p}_i - \boldsymbol{p}_j)^\top \in \mathbb{R}^6, \tag{56}$$

which are attached to the vertices of $L(\mathcal{G})$. This choice of features was suggested in [162], and motivated by the aim to achieve partial translation-invariance.

Architecture

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

convolutional operators introduced in [33]. The mathematical foundations of spectral convolutional graph neural networks are rooted in graph *signal* processing and graph Fourier transforms. In view of the convolution theorem [111], spectral convolutions are defined as linear operators that diagonalize self-adjoint operators of the graph in the Fourier (i. e. eigenvector) basis of the spectral space. State-of-the art graph convolutional operators are implemented by means of functional calculus.

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we are interested in performing vertex regression on $\mathcal{G}$, hence in processing the features (also called signals) defined on $\mathcal{V}$. Assuming $\mathcal{V}$ to be finite, namely $|\mathcal{V}| = v$, let $W \in \mathbb{R}^{v \times v}$ be the (weighted) adjacency matrix describing the graph connections between pair of vertices, i. e.

$$W_{ij} = \begin{cases} 0 & \text{if } (i,j) \notin \mathcal{E}, \\ w_{ij} > 0 & \text{if } (i,j) \in \mathcal{E}. \end{cases}$$

Moreover, let $D \in \mathbb{R}^{v \times v}$ be the degree matrix, which is a diagonal matrix whose elements are

$$D_{ii} \coloneqq \sum_{j=1, j \neq i}^{v} w_{ij}, \text{ for } i = 1, \dots, v$$

and 0 otherwise. The frequency (i.e. spectral) domain of a graph can be determined by the self-adjoint Laplacian operator Δ, either considering its unnormalized definition

$$\Delta \coloneqq D - W$$

or its normalized form

$$\Delta \coloneqq I_v - D^{\frac{1}{2}} W D^{\frac{1}{2}},$$

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 ordered 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 $\theta \in \mathbb{R}$. We can apply g_θ on Δ, resulting in

$$g_\theta(\Delta)\, \boldsymbol{s} = \sum_{i=1}^{v} g_\theta(\mu_i) \langle \boldsymbol{s}, \boldsymbol{x}_i \rangle \boldsymbol{x}_i.$$

For suitable choices of g_θ, its application to Δ, reported in the above equation, has an explicit formulation. In particular, this is the case of [33], where g_θ is defined as a polynomial filter, so that

$$g_\theta(\mu_i) = \sum_{j=0}^{d} \theta_j \mu_i^j, \quad \text{for each} \quad i = 1, \dots, v,$$

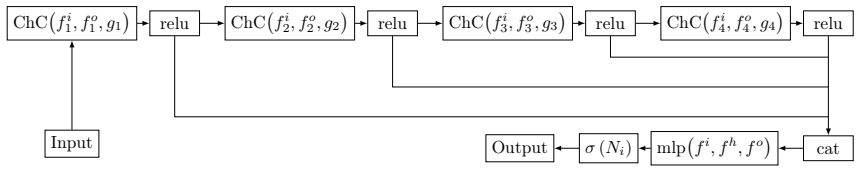


Figura 39. PARGCN Architecture design.

where θ_j are polynomial coefficients. To develop a recursive and fast formulation of spectral filter, the authors in [33] exploit the Chebychev polynomials. During the training phase, described later in Section 2, optimal values for θ_j will be computed. For further details about spectral convolutional operators and their properties we refer to [33] and [99].

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 39. It consists of the input (Input), 4 spectral convolutional layers (ChC), ReLU activation functions after each convolution (relu), a concatenation layer (cat), an MLP (mlp), composed by 3 fully connected layers, the softmax activation function (σ) and, finally, the output (Output).

In the following, we give a detailed description of each component of our architecture and the identification of the resulting parameterization.

Input The directed line graph $L\,(\mathcal{G}) = (\mathcal{V}', \mathcal{E}')$ together with the (line graph) vertex features (56) form the input of the neural network. The line graph is built on the radius graph of the unorganized point cloud $\mathcal{P}$, for which we want to compute the parameterization.

ChC The spectral graph convolutional layers are characterized by the Chebychev polynomial filters proposed in [33], based on the normalized graph Laplacian Δ. For each layer $\ell = 1, \ldots, 4$, the corresponding convolutional layer ChC$\big(f_\ell^{in}, f_\ell^{out}, g_\ell\big)$ is defined by declaring the size of input and output features $f_\ell^{in}, f_\ell^{out}$ and the dimension of the convolving filter g_ℓ. More precisely, for each $\ell = 1, \ldots, 4$, we choose $g_\ell = 2d + 1$, depending on the polynomial bi-degree $\boldsymbol{d} = (d, d)$ used in the loss function in (57). As concerns the dimension of the layer features, for the first layer $f_1^{in} = 6$, accordingly to (56), whereas we choose $f_1^{out}, f_\ell^{in}, f_\ell^{out} = 64$ for $\ell = 2, \ldots, 4$.

relu The ReLU activation function is applied element-wise to the output of each spectral convolutional layer, in order to obtain a non-linear learning model.

cat In order to prevent the degradation problem, due to numerical instabilities related to a potentially too high number or learnable parameters [71, 150], we introduce short-cut connections by concatenating the

output of each ReLU function, enabling the network to skip the sequence of layers in between.

mlp The content of the concatenation layer is processed by an MLP characterized by three hidden layers: the input layer, the hidden layer, and the output layer, of dimensions $f^{in} = 262$, $f^h = 64$, and $f^{out} = 1$, respectively. Note that f^{in} is a constrained dimension due to the dimension of the previous concatenation layer, f^h is an author's choice, and f^{out} 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 $\mathcal{G}$.

σ Finally, we apply the softmax function to each local neighbourhood identified by N_i, for each $i = 1, \dots, n$, in order to guarantee the parameterization weights to be positive and to form a partition of unity for each neighbourhood N_i. More precisely, for $j \in N_i$ and $i = 1, \dots, n$,

$$\sigma(x_j) = \frac{\exp(x_j)}{\sum_{k \in N_i} \exp(x_k)}.$$

Output The 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 $j \in N_i$ to be used to assemble and solve the linear system in (55). Due to the final softmax activation function σ, they are positive and we have $\sum_{j \in N_i} \lambda_{ij} = 1$, for all $i = 1, \dots n$.

Loss function

There are multiple strategies for training the neural network on our data set, consisting of synthetically generated data. One possibility is to compute the exact parameterization weights λ_{ij} from (54) and use them as labels, minimizing the MSE of the output weights predicted by our network. We can alternatively solve the linear system (54) based on the predicted parameterization weights and minimize the MSE with respect to the exact parameters. These two approaches result in a supervised learning method, as the exact parameterization for the training data needs to be available.

In this Thesis, we instead pursue an *unsupervised* learning approach: from the predicted parameterization weights, we first solve the linear system (54) to obtain a suitable parameterization $\mathcal{U}$ as in (3) of the input point cloud $\mathcal{P}$ as in (1). When training the network, we use this parameterization to fit a tensor product Bézier surface $\boldsymbol{s}$ to the data by solving the linear least-squares problem, see (27) in Section 1. The loss function is the residual

$$\mathcal{L}(\lambda_{ij}, \mathcal{P}) = \sum_{i=1}^{m} \|\boldsymbol{s}(u_i) - \boldsymbol{p}_i\|_2^2, \tag{57}$$

of the least-squares fitting problem corresponding to the parameter $\boldsymbol{u}_i$ that were obtained from the predicted parameterization weights λ_{ij}, by solving the system in (54). Note that in the implementation of the method it is necessary to obtain the gradients of the solution to the linear problem (54) and the linear least-squares problem, with respect to the input λ_{ij} and $\mathcal{U}$.

One advantage of this unsupervised learning approach is that the network can be trained on arbitrary point clouds, even if the points are 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 loss function (57) leads to better experimental results than minimizing the MSE with respect to the parameterization weights or the parameters.

Shape preserving correction

From the predicted parameters λ_{ij}, we obtain the parameters $\mathcal{U}$ by solving the linear system (55). While we can directly use these parameters for fitting a polynomial surface to the input point cloud by solving the least-squares problem, we also investigate the use of a further correction step inspired by the shape preserving parameterization from [49]. To this end, we first obtain a global Delaunay triangulation of the planar parameters $\boldsymbol{u}_i$, $i = 1, \dots, m$. Then, for each interior point $\boldsymbol{p}_i$, $i = 1, \dots, n$, we determine the shape preserving weights λ_{ij} with respect to the corresponding neighbours $\boldsymbol{p}_j$ in the planar triangulation, as described in Section 2. By solving the linear system (55) once more based on the neighbourhoods defined by the Delaunay triangulation and the *shape preserving corrected weights*, we obtain the corrected parameterization. In Section 5, we empirically analyze the difference between the parameters obtained using the predicted parameterization weights and the shape preserving corrected weights.

Learning meshless parameterization

In this section, we present the necessary steps for implementing the PAR-GCN method described in the previous sections. Since our method follows a data-driven approach, it is necessary to obtain a large enough data set, which we generate synthetically, similarly to the procedure described in Section 1. In what follows, we also summarize the hyperparameters used for the training process.

Data-driven methods naturally depend on data, and in particular on their availability, amount, and nature. The network architecture takes as input a directed line graph based on the radius graph of the point cloud. Since there are no public data sets available that are suitable for the parameterization of unorganized point clouds, we generate synthetic data by randomly sampling points from randomly generated polynomial parametric surfaces of bi-degree $\boldsymbol{d} = (d, d)$. The precise algorithm for data generation follows, see Algorithm 6. In particular, as concerns step 1, we generate an initial polynomial tensor product surface with control points $\boldsymbol{c}_j = [c_j^1, c_j^2, c_j^3]^T$, where $[c_j^1, c_j^2]^T$ are chosen as the tensor product Greville abscissae for bi-degree $\boldsymbol{d} = (d, d)$, and c_j^3 are sampled randomly

Algorithm 6: Generation of scattered point clouds.

Input: Polynomial bi-degree $\boldsymbol{d} = (d_1, d_2)$, number of interior samples n

1 Generate a *random* polynomial tensor product surface of bi-degree $\boldsymbol{d}$.

2 Sample random *interior* parameters $\boldsymbol{u}_i = \left(u_i^1, u_i^2\right)$ for $i = 1, \ldots, n$ according to the uniform distribution on $(0,1)^2$.

3 Set $m = n + 4\left(\lceil\sqrt{n}\rceil + 1\right)$ and sample random *boundary* parameters $\boldsymbol{u}_i$ for $i = n+1, \ldots, m$ according the uniform distribution on $\partial[0,1]^2$.

4 Evaluate the surface on the *interior* parameters $\boldsymbol{u}_i$, and define the *interior* points $\boldsymbol{p}_i \in \mathcal{P}_I$, i.e. $\boldsymbol{p}_i = \boldsymbol{s}\left(\boldsymbol{u}_i\right)$ for $i = 1, \ldots, n$.

5 Evaluate the surface on the *boundary* parameters $\boldsymbol{u}_i$, and define the *boundary* points $\boldsymbol{p}_i \in \mathcal{P}_B$, i.e. $\boldsymbol{p}_i = \boldsymbol{s}\left(\boldsymbol{u}_i\right)$ for $i = n+1, \ldots, m$.

Output: Scattered point cloud $\mathcal{P} = \mathcal{P}_I \dot{\cup} \mathcal{P}_B$

accordingly to the uniform distribution in $[0,1]$. We then choose a random rotation axis $\boldsymbol{x} \in \mathbb{R}^3$ by sampling each of its components with respect to the random uniform distribution on $[0,1]$ and a random angle ψ according to the random uniform distribution on $[0,\pi]$. The initial surface is rotated around $\boldsymbol{x} \in [0,1]^3$ by ψ.

The resulting point cloud is the ordered union of $\mathcal{P}_I$ and $\mathcal{P}_B$, i. e.

$$\mathcal{P} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n, \boldsymbol{p}_{n+1}, \ldots, \boldsymbol{p}_m\}.$$

Note that the surfaces created by this algorithm are graph surfaces of tensor product polynomial functions of arbitrary orientations in space. While this may seem like a restriction of the training data, the network has no problems generalizing to arbitrary surface data, as we will demonstrate in the numerical experiments in Section 5.

We use Algorithm 6, to generate a training data set of 10.000 point clouds sampled from polynomial parametric surfaces of bi-degree $\boldsymbol{d} = (2,2)$, 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 2. For computing the radius graph, we set the radius to $r = 0.2$ and se 32 as the maximum number of neighbours for each vertex in the graph, to reduce the computational complexity and the size of the line graph computed from the radius graph. This choice of hyperparameters leads to a preprocessed training set of size 29 GB.

We additionally generated a validation data set of 2.500 (7.2 GB) preprocessed point clouds with the same properties as 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 $1\mathrm{e}-3$ and momentum 0.9. In order to compute the loss function described in Section 2,

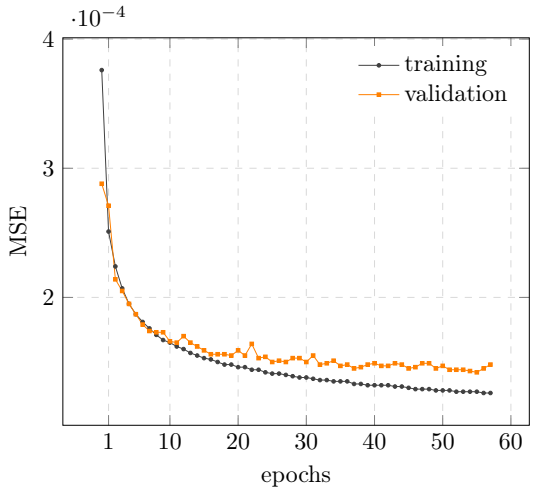


Figura 40. Training and validation error converngence rate across the epochs.

we solve the polynomial least-squares problem with bi-degree $\boldsymbol{d} = (2, 2)$, namely the same polynomial degree that characterizes the training and validation data. According to the learning curves in Figure 40, showing the training and validation errors across the training epochs, we choose the learning model corresponding to the 50th epoch. The performance on the test set is reported in Section 5.

Remark 20. *Our network architecture can also be trained on the graph obtained by local projections that is used for the shape preserving parameterization. In our experiments, this resulted in similar training and validation errors as for training on the radius graph.*

Numerical results for PARGCN

In this section, we present a selection of experiments in order to analyze the performance and generalization capability of the learning meshless parameterization model on a variety of different scattered point cloud data sets.

The quality of the proposed meshless learning parameterization model PARGCN is evaluated in terms of geometric model reconstruction accuracy based on MSE and the one-sided Direct Hausdorff Distance (DHD). For each test, we compare the error measures MSE and DHD of the geometric models obtained with PARGCN and the parameterization choices introduced in [51] and briefly discussed in Section 2 (UNIF, RECD and LPSP). 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.

The numerical experiments reported in Section 5, show the performance of the predicted meshless parameterization on a test set of the same nature

as the training and validation sets described in Section 2. In Example 5, we show the generalization capabilities of the learning model on polynomial data sets of varying bi-degree $\boldsymbol{d} = (d, d)$, together with its robustness when tested on noisy data configurations. Subsequently, in Section 5, we illustrate the generalization capability of PARGCN with respect to non-polynomial point clouds of arbitrary dimension, together with its suitability for polynomial least-squares fitting. Finally, in Section 5, we numerically demonstrate that the trained meshless parameterization model is capable of properly generalizing to tensor product B-spline fitting of arbitrary point clouds.

Test error

In this experiment, we analyze the quality of the learning model by its evaluation on the so called *test set*, i. e. a data set of the same nature as training and validation sets, but whose items have never been seen by the network during the training phase. Furthermore, we motivate the choice of introducing the shape preserving correction step described in Section 2.

More precisely, we generate 100 unorganized point clouds of $m = 264$ points, with $n = 200$ interior points as detailed in Section 2, and we preprocess them as described in Section 2, building their radius graph for a fixed radius $r = 0.2$ and allowing each interior point to have at most 32 neighbours.

After evaluating the network on this data set and solving the system in equation (55), without performing the shape preserving correction step, the MSE resulting from the polynomial least-squares fitting with bi-degree $\boldsymbol{d} = (2, 2)$, averaged on 100 point clouds, is $1.59\text{e}-4$ and the DHD, averaged on the whole data set, is $1.90\text{e}-2$. These results prove the transferability of the network with respect to unseen data of the same complexity as the training and validation sets, since they are in line with the values obtained during the training phase, already discussed in Figure 40. By adding the shape preserving correction step explained in Section 2, we obtain MSE and DHD equal to $4.45\text{e}-5$ and $1.12\text{e}-2$, respectively. This improvement motivates us to include the shape preserving correction as the final step in the PARGCN method. Figure 41 shows the Delaunay triangulation for the parametric and spatial domains built from the PARGCN parameterizations without (b-c) or with (d-e) the parameter correction step for an item from our test set.

Polynomial approximation of synthetic data

In this section, we numerically prove the generalization capability of the proposed learning model with respect to the approximation of data characterized by different attributes. More precisely, we consider polynomial fitting for different bi-degrees, and the robustness of the model in presence of noise.

We evaluate the PARGCN model on unorganized point clouds sampled from polynomial surfaces of various bi-degrees. We generate 100 unorganized point clouds by sampling $m = 264$ points, of which $n = 200$ are interior

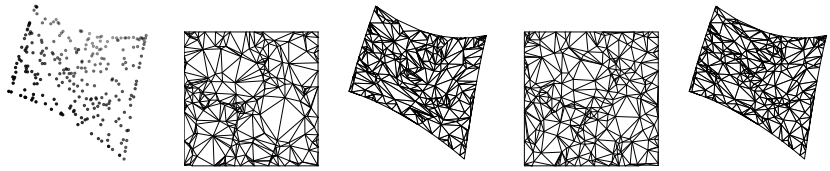


Figura 41. An input point cloud (a) and the Delaunay triangulations for the parametric (b,d) and spatial (c,e) domain related to the PARGCN parameterization obtained with (d-e) or without (b-c) the shape preserving correction step for Section 5.

points and 64 are boundary points, from tensor product polynomial surfaces of bi-degree $\boldsymbol{d} = (d, d)$ 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 $\epsilon = 1\text{e} - 2$. For each point cloud we perform the feature extraction described in Section 2, setting the radius to $r = 0.2$ and allowing each interior point to have at most 64 neighbours. Finally we run the least-squares fitting scheme for the corresponding polynomial bi-degree $\boldsymbol{d} = (d, d)$.

The results for exact and noisy data are shown in Table 6. For each bi-degree and for each parameterization methods, we report the MSE and the DHD obtained on the two test sets. The best approximation results are highlighted in bold. We observe that the accuracy gained by PARGCN with respect to LPSP when considering the MSE, is between 52% ($d = 3$) and 70% ($d = 2$) on exact data. For noisy data, PARGCN is able to outperform the LPSP method gaining on average for each degree 41% of accuracy with respect to MSE. 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. Furthermore, PARGCN's results are suitable for solving the parameterization problem of scattered data for a variety of degrees being in addition robust to noise.

Polynomial approximation of a ship hull

In this example we show the capabilities of the PARGCN model to generalize with respect to data sets of different nature and size. We consider the point cloud shown in Figure 42 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 2 building a radius graph with $r = 0.1$ while allowing a maximum number of 72 neighbours. Finally, we parameterize the input data with the RECD, LPSP, and PARGCN methods and compute the least-squares tensor product polynomial approximation of bi-degree $\boldsymbol{d} = (d, d)$ with $d = 2, 3, 4, 5$. The average MSE and DHD errors are reported in Table 7, whereas the polynomial reconstructed models are shown in Figure 43. The best approximation error values are highlighted in bold. While the approximations obtained with

Tabella 6. Exact and noisy data polynomial least squares fitting for different bi-degree $\boldsymbol{d} = (d, d)$ with $d = 2, 3, 4, 5$. Average MSE and DHD on 100 point clouds with $m = 264$ points, of which $n = 200$ interiors, and a fixed number of maximum neighbours 64, parameterized with UNIF, RECD, LPSP, and PARGCN parameterization weights for Example 5.

	exact		noisy	
	MSE	DHD	MSE	DHD
$d = 2$				
UNIF	1.09e − 3	4.18e − 2	1.09e − 3	4.59e − 2
RECD	5.01e − 4	2.42e − 2	5.00e − 4	2.80e − 2
LPSP	5.75e − 5	9.90e − 3	5.31e − 5	1.58e − 2
PARGCN	**1.69e − 5**	**7.80e − 3**	**3.43e − 5**	**1.48e − 2**
$d = 3$				
UNIF	8.84e − 4	3.68e − 2	9.09e − 4	4.69e − 2
RECD	3.44e − 4	2.91e − 2	3.67e − 4	3.92e − 2
LPSP	2.93e − 5	1.08e − 2	1.15e − 4	2.91e − 2
PARGCN	**1.40e − 5**	**9.05e − 3**	**6.41e − 5**	**2.58e − 2**
$d = 4$				
UNIF	6.41e − 4	3.68e − 2	7.37e − 4	4.45e − 2
RECD	2.38e − 4	2.98e − 2	2.64e − 4	3.86e − 2
LPSP	3.51e − 5	1.39e − 2	9.14e − 5	2.63e − 2
PARGCN	**1.34e − 5**	**9.70e − 3**	**5.15e − 5**	**2.49e − 2**
$d = 5$				
UNIF	4.48e − 4	3.67e − 2	4.71e − 4	4.18e − 2
RECD	1.34e − 4	2.97e − 2	1.58e − 4	3.58e − 2
LPSP	2.60e − 5	1.19e − 2	7.60e − 5	2.49e − 2
PARGCN	**1.06e − 5**	**9.93e − 3**	**4.53e − 5**	**2.31e − 2**

Figura 42. Point cloud sampled from the model of a ship hull.

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.

Tabella 7. Polynomial least-squares fitting of bi-degree for $\boldsymbol{d} = (d, d)$ with $d = 2, 3, 4, 5$ for Section 5 parameterized with UNIF, RECD, LPSP, and PARGCN parameterization weights.

	RECD	LPSP	PARGCN	RECD	LPSP	PARGCN
		$d = 2$			$d = 3$	
MSE	6.80e − 4	1.32e − 4	**7.55e − 5**	2.55e − 4	4.10e − 4	**2.51e − 5**
DHD	4.02e − 2	3.39e − 2	**3.71e − 2**	4.73e − 2	2.39e − 2	**2.32e − 2**
		$d = 4$			$d = 5$	
MSE	1.74e − 4	2.27e − 5	**1.10e − 5**	1.20e − 4	1.68e − 5	**6.39e − 6**
DHD	3.20e − 2	1.39e − 2	**1.08e − 2**	3.14e − 2	1.07e − 2	**8.94e − 3**

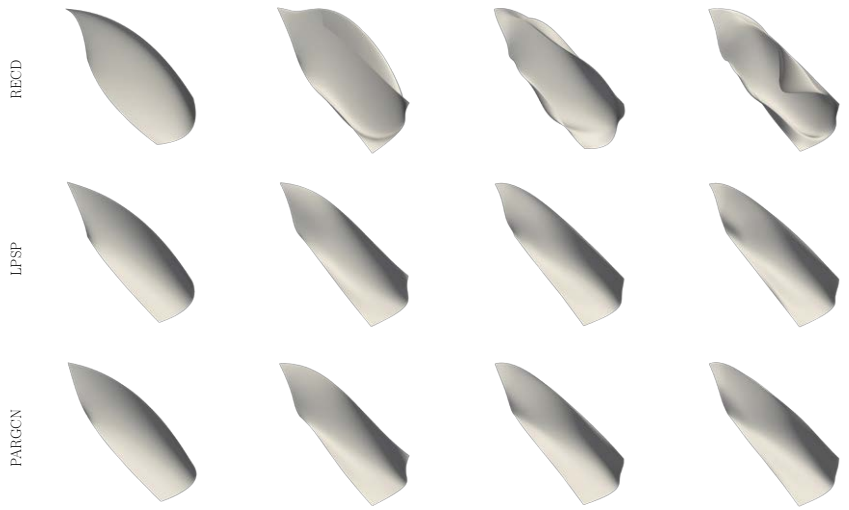


Figura 43. Polynomial least-squares approximation of the ship-hull point cloud shown in Figure 42 for RECD (top), LPSP (center), and PARGCN (bottom) parameterization weights and bi-degree (d, d) with $d = 2, 3, 4, 5$ from left to right for Section 5.

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 scattered data fitting. In particular, we show the suitability of the output parameterization for tensor product B-spline penalized least-squares fitting.

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 45 (left). In this case, we build a radius graph with $r = 0.2$, allowing a maximum number of 64 neighbours for each interior point. The parametric values in $[0, 1]^2$, resulting from different parameterization methods, namely

RECD, LPSP, and PARGCN, together with the resulting triangulations, are shown in Figure 44. 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. The low quality results obtained with RECD and LPSP in this example are visible also by analyzing the triangulations in Figure 44 (left and center). This leads to mesh distortion phenomena and the presence of artifacts when these parameterizations are considered for polynomial and spline surface reconstruction. The MSE for polynomial approximation for different bi-degrees $\boldsymbol{d} = (d, d)$, with $d = 2, 3, 4, 5$, is shown in the left column of Table 8, while the right column reports the MSE obtained when performing a (penalized) least-squares fitting with tensor product B-splines characterized by 4 levels of uniform refinement. Across all degrees, when PARGCN is used in order to perform polynomial least-squares approximation, we are able to gain on average 32% of accuracy with respect to LPSP for MSE. Moving to the B-spline setting, the MSE decreases due to the growth of degrees of freedom for all the methods, but PARGCN registers a gain in accuracy of 70% on average for each degree. Figure 45 shows the geometric model obtained by computing the (penalized) B-spline least-squares fitting with two (center) and four (right) levels of uniform refinement when the PARGCN parameterization model is considered. Moreover, Figure 46 illustrates the parameter distribution of the B-spline surface resulting from PARGCN and 4 levels of uniform refinement.

Beyond PARGCN

The PARGCN parameterization method is a novel learning approach for the meshless parameterization problem of scattered datasets. It goes beyond closed-form heuristic choices of the parameterization weights, thus injecting geometric deep learning into a fundamental process of surface modeling and computer-aided design to improve performance. In particular, the numerical results show the performance of the proposed model and its capabilities to suitably generalize with respect to different data configurations, such as the size of the point clouds, the presence of noise, different polynomial degrees, and data sampled from non-polynomial surfaces. In particular, we applied the method to real-world data, and we also investigated the generalization of the method to fitting B-spline surfaces of varying refinement levels.

A key step of our PARGCN method is the novel local feature extraction scheme by means of a directed line graph that is computed from the radius graph of an unstructured point cloud, see Section 2. This enabled us to design a new network architecture that has the ability to predict barycentric parameterization weights for each edge in the radius graph, which are then used to obtain a suitable parameterization.

On the other hand, the computation of the line-graph $L(\mathcal{G})$ as in Definition 12 represents the bottleneck of this approach in terms of computational time and memory. In particular, for point clouds with a high number of

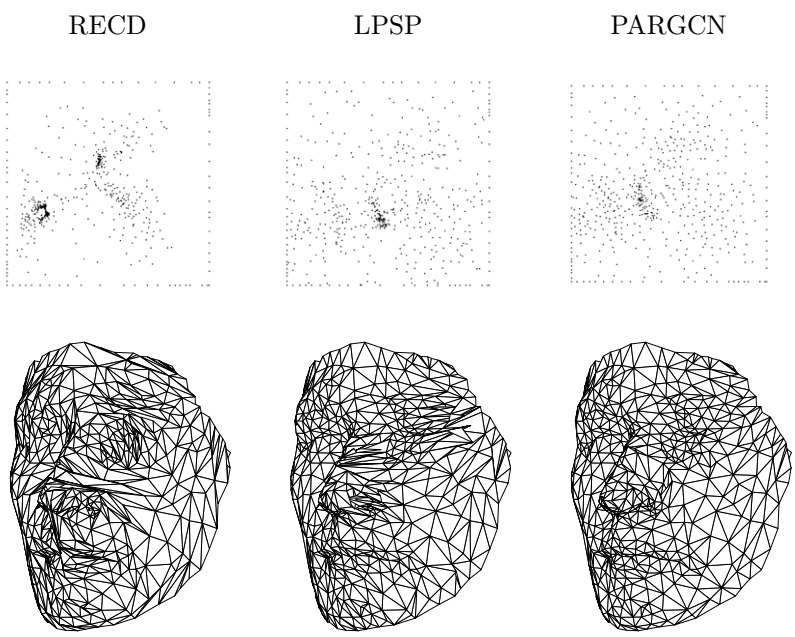


Figura 44. Parametric values on $[0,1]^2$ (top) for the face example, and 3D Delaunay triangulations for different methods (bottom) for Section 5.

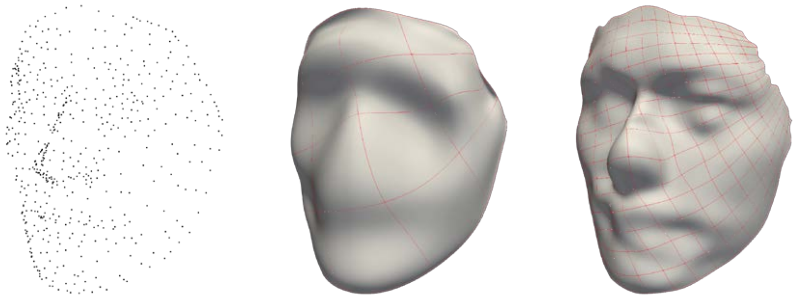


Figura 45. Point cloud considered in Section 5 (left) and penalized B-spline least-squares fitting of bi-degree $\boldsymbol{d} = (5,5)$ obtained with PARGCN parameterization for 2 (center) and 4 (right) levels of uniform refinement.

vertices ($> 10^3$), the number of edge connections can potentially be very high, and therefore, the dimension of the dual graph drastically increases. This led us to work with datasets of the magnitude of hundreds of data points. In particular, we report the dimension of the training and validation sets in Section 2: to store 12.500 point clouds, each consisting of 264 items in $\mathbb{R}^3$, and their radius graph and line graph, about 36.2GB are needed. In addition, a crucial role is played by the choice of the radius r to define

Tabella 8. Polynomial least-squares fitting and B-Spline fitting after 4 steps of uniform refinement, with bi-degree $\boldsymbol{d} = (d, d)$ for $d = 2, 3, 4, 5$ in Section 5, with $m = 543$ points, of which $n = 458$ interiors, parameterized with UNIF, RECD, LPSP, and PARGCN parameterization weights.

	Polynomial surface		B-Spline surface	
	MSE	DHD	MSE	DHD
$d = 2$				
RECD	3.54e − 3	1.71e − 1	2.68e − 4	8.65e − 2
LPSP	1.55e − 3	1.55e − 1	8.43e − 5	3.32e − 2
PARGCN	**8.62e − 4**	**1.24e − 1**	**2.76e − 5**	**4.00e − 2**
$d = 3$				
RECD	2.25e − 3	1.28e − 1	2.42e − 4	7.98e − 2
LPSP	7.10e − 4	1.10e − 1	8.97e − 5	3.52e − 2
PARGCN	**4.31e − 4**	**1.09e − 1**	**2.52e − 5**	**3.73e − 2**
$d = 4$				
RECD	1.71e − 3	1.24e − 1	2.43e − 4	7.91e − 2
LPSP	4.30e − 4	9.96e − 2	8.13e − 5	3.40e − 2
PARGCN	**3.57e − 4**	**1.05e − 1**	**2.48e − 5**	**3.91e − 2**
$d = 5$				
RECD	1.21e − 3	1.06e − 1	2.31e − 4	7.90e − 2
LPSP	3.37e − 4	9.01e − 2	8.48e − 5	3.51e − 2
PARGCN	**2.42e − 4**	**8.87e − 2**	**2.39e − 5**	**3.74e − 2**

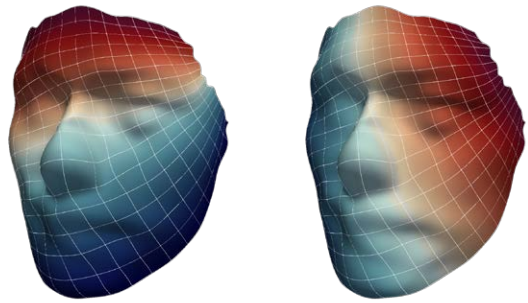


Figura 46. Parameter distribution along the two parametric dimensions for the tensor product B-spline model with 4 levels of uniform refinement in Section 5.

the graph $\mathcal{G}$. As highlighted in Section 2, the existence, uniqueness, and suitability of the solution of (55), i. e. of a parameterization for the input point cloud, depend on the choice of r. If r is *relatively* too small or too big, a suitable parameterization for the input point cloud at hand does not exist. The fine-tuning of r is therefore fundamental and unavoidable for the proposed method. Finally, post-processing steps, e. g., the solution of the linear system in (55), are required in order to recover the final parameterization after evaluating the network.

These considerations motivate the development of a data-driven method

that performs a fully automatic regression on the input point cloud. In particular, it is desirable to avoid the computation of the radius graph $\mathcal{G}$, hence the translation into its dual counterpart $L(\mathcal{G})$. In addition, we also want to avoid further computations after the network evaluation, i. e. we seek a learning method that takes as input a scattered point cloud and gives directly in output a suitable parameterization. In the next Section, we propose a graph neural network architecture, characterized by a novel boundary enforcing layer that is applied directly to point cloud data.

3.3. BIDGCN: parameterization of scattered data with boundary information

Most current graph neural network architectures are classified as *message passing neural networks* [63], meaning that hidden states at each vertex are computed by aggregating the output of message functions applied to the features of the vertex and its neighbours connected by an edge. GCNs that are employed for processing discrete surface data have mostly been applied so far to *closed* surfaces, i. e. surfaces without boundary. This simplifies the design of suitable graph convolutional operators significantly, since all vertices of the discrete manifold or point cloud can be handled in the same way and no explicit distinction between interior and boundary vertices needs to be made. However, in many applications in geometric modelling, geometry processing, and numerical analysis, the input geometries are given as discrete surfaces *with* boundary. For a variety of problems, *boundary conditions* are imposed on the corresponding vertices, and often the solution to a problem is only uniquely determined up to the boundary conditions. For example, this is the case for boundary value problems of elliptic partial differential equations on surfaces, as well as for the problem of scattered point cloud parameterization. In order to apply deep learning-based methods to this kind of problems, it is therefore necessary to devise a network architecture that takes into account the boundary conditions in addition to the standard vertex features. Since boundary conditions can be regarded as additional features that are defined only on the boundary vertices but not on the interior vertices, this means that such a network architecture needs to be able to handle data with varying feature dimensions.

In this Section, we propose the new model, called BIDGCN, for processing scattered point clouds. The key idea of BIDGCN is to treat boundary conditions as additional features at the boundary vertices of the point cloud. These features are propagated into the whole point cloud by a novel graph convolution operator that contains two separate trainable message functions: the first one for edges between interior and boundary vertices, and the second one for edges between interior vertices. In the subsequent hidden layers, the information stemming from the boundary conditions is further processed and used to predict the solution for the problem at hand. The two main properties of the proposed network layer are: (i) the ability to incorporate boundary conditions in its prediction (due to the boundary input layer) (ii) the dynamic prediction of the graph used for message passing (due to the dynamic edge convolution approach as in [162]).

As a use case for our new network architecture, we apply it to the

problem of scattered point cloud parameterization. In particular, the new BIDGCN parameterization method overcomes the limitations of the classical methods and of the PARGCN method, discussed in Section 2 in terms of efficiency, robustness, and sensitivity to parameter-dependent graph connectivity.

In our approach, we also assume that the boundary is already parameterized, as in the standard methods, see Section 2. We then use a network architecture based on our new BIDGCN to predict the parameterization of the interior vertices. Using our boundary informed dynamic graph convolutional network leads to a number of advantages compared to the classical methods for scattered point cloud parameterization:

1. *Computational efficiency.* After training the network its evaluation is computationally much more efficient than applying the classical methods in [51] and the learning-based method in [60]. In particular, once the training process is completed, large advantages in computational time can be observed, ranging from 4 times upto a speedup of 180 times.

2. *Robustness with respect to noise.* Our method is robust with respect to noise and results in much better approximations when approximating noisy point clouds with polynomial surfaces. In particular, an improvement in the accuracy from 60% up to 80% is observed with respect to existing algorithms.

3. *Robustness with respect to adjacency graph.* While for existing methods, e. g., [51, 60], it is necessary to construct a graph by suitably choosing the local neighbourhoods of each interior vertex, our method automatically predicts a suitable graph without the need of free parameter selection. Note that an improper graph selection can even lead to failure of classic parameterization algorithms due to non-invertible linear systems in case of sparse neighbourhoods.

After we determine a parameterization of the scattered point cloud using our proposed method, we use that parameterization to approximate the point cloud with a smooth spline surface.

BIDGCN: Boundary Informed Dynamic Graph Convolutional Network

The graph convolutional neural network we propose is characterized by its ability to handle and propagate point cloud boundary information. This is achieved by the development of a *new* boundary informed dynamic edge convolutional layer, which is an extension of the dynamic edge convolution operator originally proposed in [162]. We first briefly describe the original dynamic edge convolution operator, and then we present our new boundary informed dynamic convolutional layer and the additional layers of the network architecture.

We assume to be given a point cloud $\mathcal{P}$ together with vertex features $\boldsymbol{x}_i \in \mathbb{R}^p$, where $p \in \mathbb{N}$ is the input dimension. The dynamic edge convolution operator defined in [162] computes new features $\boldsymbol{y}_i \in \mathbb{R}^q$ for all points in

$\mathcal{P}$, where $q \in \mathbb{N}$ is the output dimension. To this end, first the k-nearest neighbour graph $\mathcal{G}$ is computed based on the input features. This is a directed graph where the existence of a directed edge (j, i) implies that j is among the k nearest neighbours of i with respect to the Euclidean distance $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2$ of the input features. In the next step, edge features are computed for each directed edge in $\mathcal{G}$: for all $(j, i) \in \mathcal{G}$

$$\boldsymbol{e}_{ji} = h_\Theta(\boldsymbol{x}_i, \boldsymbol{x}_j - \boldsymbol{x}_i),$$

is evaluated, where h_Θ is a feed-forward neural network

$$h_\Theta : \mathbb{R}^{2p} \to \mathbb{R}^q$$

with trainable weights Θ. Finally, at each vertex of $\mathcal{G}$, the edge contributions are aggregated as

$$\boldsymbol{x}_i' = \square_{(j,i)\in\mathcal{G}} \boldsymbol{e}_{ji},$$

where $\square$ computes the mean value. In a *dynamic* edge convolution network, the k-nearest neighbour graph is recomputed after each layer with respect to the new features $\boldsymbol{x}_i'$, thereby allowing the network to pass information arbitrarily fast across the point cloud. The dynamic approach enables the network to automatically predict a suitable graph for message passing instead of using a fixed one. This implies that information can travel arbitrarily far in each layer, as it is determined by the training process. In [162] it was shown that updating the graph after each layer leads to a significantly improved accuracy compared to performing edge convolution on a static graph.

As a slight modification, while in [162] the k-nearest-neighbour graph was used, to emphasize locality we propose to use the radius graph also in the hidden layers. In particular, instead of specifying the number of neighbours k, we specify a radius $r > 0$ and add directed edges (i, j) and (j, i) for all $i, j \in V$ such that

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 \leq r.$$

By dynamically recomputing the radius graph instead of the k-nearest neighbour graph, we reduce the dependence of the network architecture on the local density of the input point clouds. In particular, the radius graph results in neighbourhoods with a fixed maximum distance, while a k-nearest graph might associate points with very large distances if the point cloud is locally sparse.

Based on the dynamic edge convolution, we introduce a new boundary informed input layer that takes as input the point cloud with its vertex features as well as the boundary conditions and propagates the boundary conditions into the new features of the interior points. The output of this layer consists of all *interior* points together with new vertex features. We name the resulting neural network architecture Boundary Informed Dynamic Graph Convolutional neural Network (BIDGCN).

Assume that the input point cloud is decomposed like in Section 2 as $\mathcal{P} = \mathcal{P}_I \dot\cup \mathcal{P}_B$ into interior and boundary points and that each vertex $\boldsymbol{p}_i \in \mathcal{P}_I$

comes with features $\boldsymbol{x}_i \in \mathbb{R}^p$ and every vertex $\boldsymbol{p}_j \in \mathcal{P}_B$ comes with features $(\boldsymbol{x}_j, \boldsymbol{u}_j) \in \mathbb{R}^p \times \mathbb{R}^s$, were we regard $\boldsymbol{u}_j \in \mathbb{R}^s$ as boundary conditions.

We first compute two different radius graphs

$$\mathcal{G}_{I\to I} \quad \text{and} \quad \mathcal{G}_{B\to I},$$

where $\mathcal{G}_{I\to I}$ contains all directed edges

$$(j,i) \quad \text{with} \quad \boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{P}_I \;:\; \|\boldsymbol{x}_i - \boldsymbol{x}_j\| \leq r,$$

while $\mathcal{G}_{B\to I}$ contains all directed edges

$$(k,i) \quad \text{with} \quad \boldsymbol{p}_i \in \mathcal{P}_I,\ \boldsymbol{p}_k \in \mathcal{P}_B \;:\; \|\boldsymbol{x}_i - \boldsymbol{x}_k\| \leq r.$$

This means that even in $\mathcal{G}_{B\to I}$, the edges do not depend on the boundary conditions $\boldsymbol{u}_j$. Note that vertices in $\mathcal{P}_B$ only have outgoing edges and no incoming ones.

We then compute edge contributions for all edges using two separate neural networks. For all $(j,i) \in \mathcal{G}_{I\to I}$, we compute

$$\boldsymbol{e}_{ji} = h_\Theta\left(\boldsymbol{x}_i, \boldsymbol{x}_j - \boldsymbol{x}_i\right),$$

where h is a feed-forward neural network

$$h_\Theta : \mathbb{R}^{2p} \to \mathbb{R}^q$$

with output feature size q and learnable weights Θ.

For edges $(k,i) \in \mathcal{G}_{B\to I}$, the boundary conditions at the boundary vertices are concatenated with the vertex features and we compute

$$\boldsymbol{e}_{ik} = g_\Phi\left(\boldsymbol{x}_i, \boldsymbol{x}_k - \boldsymbol{x}_i, \boldsymbol{u}_k\right),$$

where g is another feed-forward neural network

$$g_\Phi : \mathbb{R}^{2p+s} \to \mathbb{R}^q$$

with the same output feature size and independent learnable weights Φ.

Finally, the edge contributions are aggregated in the target vertices of the directed edges. By construction, all target vertices are contained in $\mathcal{P}_I$. For $\boldsymbol{p}_i \in \mathcal{P}_I$, we have

$$\boldsymbol{x}'_i = \left(\square_{j:(j,i)\in\mathcal{G}_{I\to I}} \boldsymbol{e}_{ji}\right) \square \left(\square_{k:(k,i)\in\mathcal{G}_{B\to I}} \boldsymbol{e}_{ki}\right),$$

where the aggregation operator $\square$ can be the sum, the mean or the component-wise maximum value. The output of the layer consists of features of dimension q for the interior point cloud $\mathcal{P}_I$. During training Θ and Φ are optimized simultaneously. The flow of the input layer is depicted in Figure 47 and its application to an interior vertex is illustrated in Figure 48.

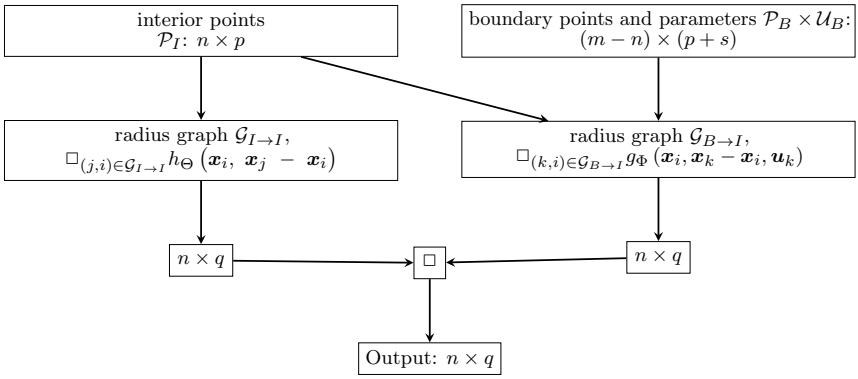


Figura 47. Boundary informed input layer.

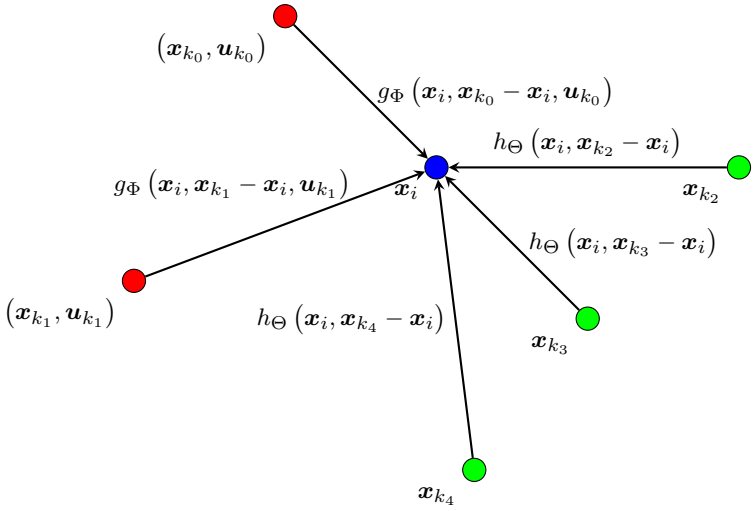


Figura 48. Edge contributions of the input layer for $\boldsymbol{x}_i$ (blue) computed from the features at the neighbouring boundary vertices (red) and interior vertices (green).

The input layer gives as output new features in $\mathbb{R}^m$ for the interior point cloud $\mathcal{P}_I$. These features can then be processed by further hidden layers, e. g., based on dynamic edge convolution. Through the application of the different layers, the information that was transported by the input layer from the boundary vertices to their neighbours in $\mathcal{G}_{B \to I}$ is propagated further into the interior of the point cloud.

The novel BIDGCN layer gives as output new features in $\mathbb{R}^q$ for the interior point cloud $\mathcal{P}_I$. To further process these features, we proceed like in the original Dynamic Graph Convolutional Neural Network (DGCNN) by computing a new graph for $\mathcal{P}_I$ based on the new features $\boldsymbol{x}_i' \in \mathbb{R}^m$. As a slight modification, while in [162] the k nearest-neighbour graph was

used, we propose to use the radius graph also in the hidden layers. This makes BIDGCN architecture more independent on the sampling density of the point cloud. Since the number of neighbours of the vertices is not constant, we use the mean value as the aggregation of the edge features, also motivated by the goal of achieving independence of the sampling density.

Through the application of the different layers, the information that was transported by the input layer from the boundary vertices to their neighbours in $\mathcal{G}_{B\to I}$ is propagated further into the interior of the point cloud. Since the radius graph is recomputed after each layer, the information can travel arbitrarily as far as determined by the training process.

Point cloud parameterization using BIDGCN

In the following, we present how a network architecture based on BIDGCN can be used to address the parameterization problem of scattered point clouds. As detailed in the previous section, the task for our trained neural network is to predict parameters $\boldsymbol{u}_i \in [0,1]^2 \subset \mathbb{R}^2$ for the given point cloud. The neural network should then approximate an operator that assigns to each sufficiently large set of input features a parameterization that is optimal for fitting a surface $\boldsymbol{s} : [0,1]^2 \to \mathbb{R}^3$ to the input points. Without fixing some of the parameter values a priori, this operator is not uniquely defined, making it difficult to directly train a neural network for this task. Therefore, the parameters $\boldsymbol{u}_j$ for the boundary points needs to be suitably computed. This determines a well-defined specific parameterization operator that takes as input the positions of all points, together with the boundary parameterization, and gives as output the unique optimal parameterization of the interior points. We train our neural network to approximate this operator.

In order to predict the optimal parameterization with respect to a specific choice of boundary parameters, the neural network needs to take into account the boundary parameters as boundary conditions. This motivates our choice to apply BIDGCN to the parameterization problem.

We design a neural network based on the new boundary informed input layer described in Section 3. The architecture of our BIDGCN is shown in Figure 49, and it has an overall number of trainable parameters of 192,130. It consists of the boundary informed input layer (BIDGC layer), four hidden dynamic edge convolution layers (DGC layers), and an MLP as output layer.

Input The learning model takes as input the interior points $\mathcal{P}_I$, whose features correspond to their Cartesian coordinates, together with the boundary points $\mathcal{P}_B$ and their features, i. e. their Cartesian coordinates and their parameters $\mathcal{U}_B$.

BIDGC In the Boundary Informed Convolutional layer, we have two MLPs, one for the edges in $\mathcal{G}_{I\to I}$ and one for the edges $\mathcal{G}_{B\to I}$. For $\mathcal{G}_{I\to I}$ we train an MLP with layer sizes $\{6, 64, 64\}$, while for $\mathcal{G}_{B\to I}$ we train an MLP with layer sizes $\{8, 64, 64\}$. Note that the input dimension corresponds to the edge features in the two different graphs.

DGCN The hidden layers are assembled as ordinary Dinamic Edge Convolutional layers, where the k-nearest neighbour graph has been replaced by the radius graph. In all hidden dynamic edge convolution layers, the MLP has size $\{128, 64\}$. The output feature dimension of the hidden layers is deliberately chosen to be moderate because a new radius graph is computed after each layer with respect to the output features, which can be costly for high dimensions.

cat Finally, the output of the last hidden layer is concatenated with the output of all hidden layers and fed into a final MLP of size $\{320, 256, 256, 2\}$.

Output The parameterization $\mathcal{U}_I$ for the interior points $\mathcal{P}_I$.

After each hidden layer, the ReLU activation function is applied. Finally, after the output layer, the sigmoid activation function is applied to enforce that the predicted parameters lie in $[0, 1]^2$. Since all layers in BIDGCN are convolutional except for the last layer that is applied vertex-wise, the network can be applied to any point cloud, independent of its size.

The overall number of trainable parameters in this network architecture is 192.130. We remark that the choice of using four hidden layers takes into account a suitable trade-off between computational time and error. Indeed adding more hidden layers slightly increased the computation times without effective gains in accuracy.

Loss function

To train the neural network for the prediction of optimal fitting parameters, we follow an unsupervised strategy and use the predicted parameters for the interior points $\mathcal{U}_I$ as well as the prescribed parameters for the boundary points $\mathcal{U}_B$ to fit a bi-quadratic polynomial surface to the point cloud. More precisely, we assemble the collocation matrix, as defined in (26), Section 1, i. e.

$$A = \begin{pmatrix} \beta_0(\boldsymbol{u}_1) & \dots & \beta_8(\boldsymbol{u}_1) \\ \vdots & \ddots & \vdots \\ \beta_0(\boldsymbol{u}_m) & \dots & \beta_8(\boldsymbol{u}_m) \end{pmatrix} \in \mathbb{R}^{m \times 8},$$

where, β_j are the bi-quadratic tensor-product Bernstein polynomials and $\boldsymbol{u}_i \in [0, 1]^2$ are the prescribed parameters if $\boldsymbol{p}_i \in \mathcal{P}_B$ and the parameters predicted by the neural network if $\boldsymbol{p}_i \in \mathcal{P}_I$.

The right hand side $\boldsymbol{b} \in \mathbb{R}^{m \times 3}$ is given by the features of the point cloud, i. e.

$$\boldsymbol{b}_i = \boldsymbol{x}_i.$$

Consequently, we solve the linear system, representing the least-squares problem,

$$\min_{\boldsymbol{c} \in \mathbb{R}^{9 \times 3}} \|A\boldsymbol{c} - \boldsymbol{b}\|_2^2 \tag{58}$$

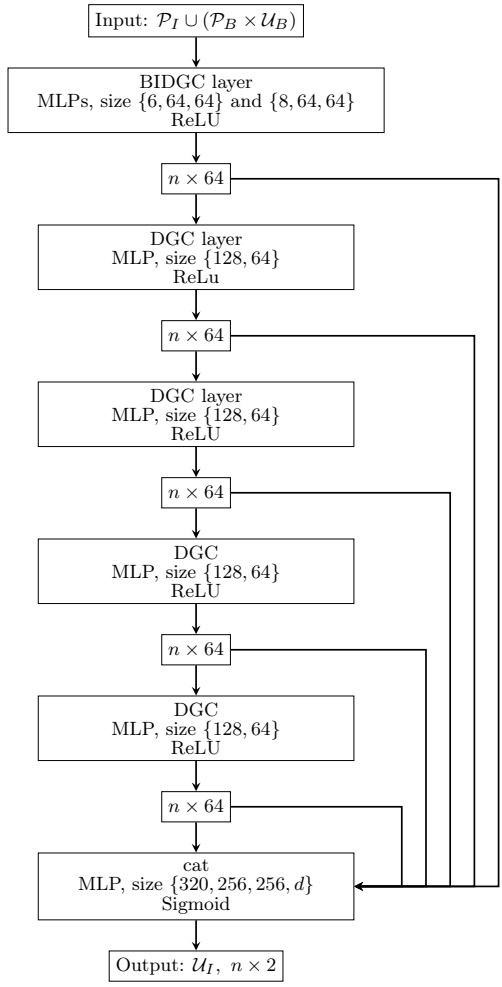


Figura 49. BIDGCN architecture for $p = 3$ (dimension of the point cloud) and $s = 2$ (parametric dimension).

using QR decomposition and keeping track of the gradients with respect to the learnable weights of the neural network. The loss for the predicted parameterization of the interior points is the residual of (58).

Learning boundary informed parameterization

In order to train the network for parameterizing scattered point cloud data, we generate a data set consisting of $100,000$ point clouds by following Algorithm 6. The points are sampled from bi-quadratic tensor-product

Bézier surfaces whose control points $\boldsymbol{c}_{ij}$ are randomly sampled from

$$\mathcal{C}_{ij} = \left[\frac{i}{2} - \frac{1}{4}, \frac{i}{2} + \frac{1}{4}\right] \times \left[\frac{j}{2} - \frac{1}{4}, \frac{j}{2} + \frac{1}{4}\right] \times [-1, 1]$$

for $i, j = 0, \dots 2$, according to the uniform distribution. This choice of sampling space ensures a large variation in the complexity of the surfaces while avoiding self-intersections. Finally, we rotate each surface around a randomly sampled axis in $\mathbb{R}^3$ by a random angle. Besides the vertex features $\boldsymbol{x}_i \in \mathbb{R}^3$ for all interior and boundary points, we store the exact parameters $\boldsymbol{u}_i \in \mathbb{R}^2$ for all boundary points. Each point cloud $\mathcal{P} = \mathcal{P}_I \dot{\cup} \mathcal{P}_B$ contains 1000 interior points and 34 boundary points, and thus the ratio of interior points to boundary points is equivalent to a uniformly distributed point cloud.

Remark 21. *Note that the choice of point cloud size in the training data set does not mean that the trained network is limited to point clouds of this size. The network described in the previous section can be applied to any point cloud, and we will observe in the numerical experiments that it performs well for a large range of point cloud sizes.*

Before evaluating the network on a point cloud from the training data set, from the validation and data sets, or from a real-world sample, we normalize the vertex features $\boldsymbol{x}_i \in \mathbb{R}^3$ by translating and scaling them so that they lie in $[0, 1]^3$. Due to the affine invariance of Bézier and B-Spline surfaces this does not affect the optimal choice of parameters $\boldsymbol{u}_i \in \mathbb{R}^2$.

Numerical results for BIDGCN

We implemented our method using the PyTorch [126] and PyG [46] libraries. The code for training and testing is available at `https://github.com/felixfeliz/BIDGCN`.

We trained the network architecture described in Section 3 and visualized in Figure 49 on our data set consisting of 100.000 point clouds sampled from bi-quadratic surfaces according to Section 3. Referring to Section 3, the loss function during training is the fitting error. We trained the network using stochastic gradient descent, starting with learning rate 0.1 and decreasing whenever the loss plateaus.

Moreover, we also consider *noisy* data, where we added Gaussian noise with varying standard deviation.

In order to test the performance of our method, we generate several test data sets as described in Section 3. Each data set consists of 100 point clouds sampled from tensor-product polynomial surfaces. We generated distinct test data sets by varying the polynomial degree, the amount of noise and the number of sample points. In particular, we used polynomial bi-degrees 2, 3, 4 and 5, Gaussian noise with standard deviation $5\mathrm{e}-3$, $1\mathrm{e}-2$ and $5\mathrm{e}-2$, and sample sizes between 200 and 2000 inner points. The number of boundary points was always set so that the ratio between inner points and boundary points is equivalent to the one of a uniform mesh. Moreover, we also consider real-world data sets.

Tabella 9. Summary of hyperparameters.

No. trainable parameters	192.130
No. DGCNN layers	4
Optimizer	Stochastic gradient descent
Learning Rate	0.1, decreased on plateaus
Batch size	1
Training set size	100.000
GPU	NVIDIA GeForce GTX 1060
Floating point precision	double
Degree in test data	2, 3, 4, 5
Noise in test data	0, 5e − 3, 1e − 2, 5e − 2

We start in Section 3 with an ablation study of our new boundary informed layer to demonstrate its fundamental role in tackling the parameterization problem. We then show the comparison of the proposed method with the three standard approaches described in Section 2 in Section 3–3.

The training, as well as all evaluations of the different methods, was performed on a standard workstation computer with an NVIDIA GeForce GTX 1060 GPU. The hyperparameters for training and testing our method are summarized in Table 9.

Ablation study and comparison with learning-based methods on exact and noisy data

In our first experiment, we compare the performance of BIDGCN with two other learning methods based on graph neural networks. The first method we compare with is the standard dynamic graph convolutional neural network proposed in [162]. In order to ensure a fair comparison, we trained a network with the same architecture as our BIDGCN shown in Figure 49, with the only difference being that instead of our novel BIDGCN input layer, we use as input layer another standard dynamic edge convolution layer. Thus, this comparison serves at the same time also as an ablation study for our novel boundary informed input layer. We trained this network on the same dataset that we used for training BIDGCN, as described in Section 3. In our plots, we will denote this method by DGCNN. The second learning-based method that we compare with is the PARGCN method proposed in [60] and described in Section 2. We recall that PARGCN is based on the standard parameterization methods but it uses a graph convolutional neural network to predict parameterization weights instead of determining them heuristically. Since the parameterization weights are defined edge-wise, PARGCN generates a *line graph* of the initial radius graph, which can be costly. Moreover, it applies a post-processing step called *shape-preserving correction*, which is similar to the shape-preserving parameterization described in Section 2. PARGCN was trained on a dataset consisting of point clouds of size 200, sampled from randomly generated bi-quadratic surfaces.

We first apply the three methods to a test data set consisting of point clouds sampled from bi-quadratic surfaces with sample size varying between 200 and 2000. Note that while BIDGCN was trained only on point clouds of size 1000, it is important to ensure that it gives good results for point clouds of any size. We use each predicted parameterization to fit the input point cloud with a bi-quadratic surface and compute the MSE of the approximation. The computation time as well as the fitting MSE is shown in the left column of Figure 50. All the plots are semi-log plots with a logarithmic scale used for the time and error axes.

Our first observation is that the MSE resulting from DGCNN is prohibitively large while the network based on our novel BIDGCN input layer results in a very good approximation. This is expected since, as described earlier, the parameterization problem cannot be solved without taking into account the parameterization of the boundary curves and a boundary-informed input layer is therefore necessary. This ablation study demonstrates this fact and moreover shows that our novel BIDGCN layer is indeed able to correctly process the given boundary information. In order to better visualize the difference of the predicted parameterizations of BIDGCN and DGCNN, we plot two examples from our test data set in Figure 51. We observe that the parameterization predicted by DGCNN contains large voids and is far from the original. This is expected, since the optimal parameterization is not uniquely defined by the positional vertex features only. On the other hand, BIDGCN correctly takes the boundary conditions into account and predicts parameters that are very close to the original ones, leading to a much better approximation with a bi-quadratic surface.

A second observation from Figure 50 is that the performance of BIDGCN does not depend significantly on the size of the scattered data set, even if the network was trained exclusively on data with 1000 points per point cloud.

Comparing our method with PARGCN, we observe that the MSE is of the same order with PARGCN, which was trained with sample size 200, having a slight edge for smaller point clouds. However, the difference in computation time is very large and the speed-up of BIDGCN over PARGCN is over two orders of magnitude. We note that a large part of PARGCN's computational complexity is due to its reliance on an expensive post-processing step. The BIDGCN-based parameterization method does not need an expensive post-processing and the predicted parameters can directly be used for fitting a surface to the point cloud.

Measured real-world data is always subjected to noise. For this reason, we study the behaviour of BIDGCN, DGCNN, and PARGCN when applied to noisy data. In particular, we evaluate all methods on 100 point clouds sampled from bi-quadratic surfaces with added Gaussian noise of standard deviation $5e-3$ and $1e-2$. In the middle and right columns of Figure 50 we show the MSE and computation time when applying the methods to noisy data with Gaussian noise of standard deviation $5e-3$ and $1e-2$, respectively. As in the non-noisy case, the plain DGCNN does not result in acceptable fitting errors in the presence of noise. We observe that BIDGCN is more robust than PARGCN with respect to noise and results in better

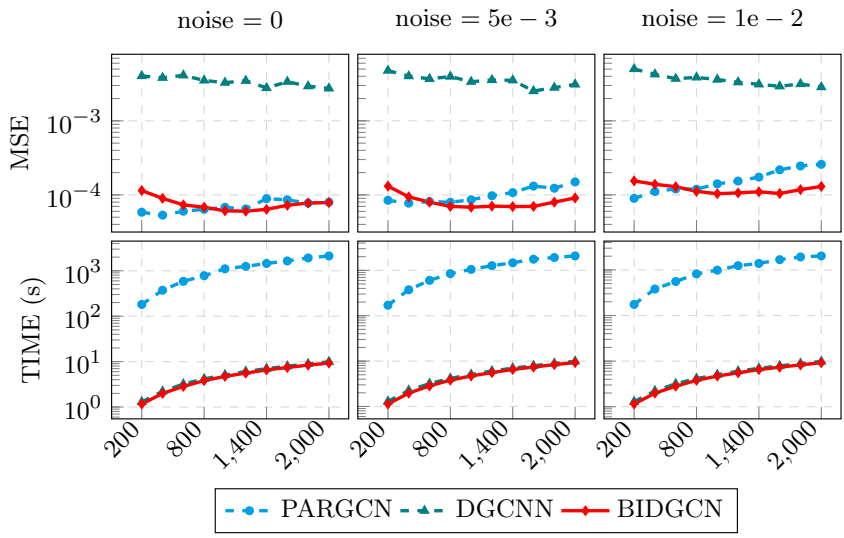


Figura 50. Comparison between learning parameterization methods of Section 3. MSE (top) and computation time (bottom) when applying the three learning-base methods BIDGCN, DGCNN, and PARGCN to point clouds of different sizes sampled from 100 bi-quadratic surfaces without noise (left), with Gaussian noise of standard deviation 5e − 3 (middle) and with Gaussian noise of standard deviation 1e − 2 (right).

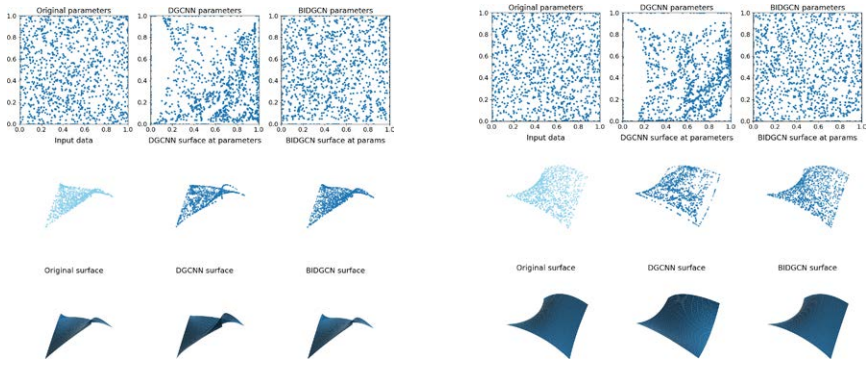


Figura 51. Comparison of the BIDGCN network with a pure DGCNN trained for the point parameterization problem considered in Section 3. Top row: Original parameters and parameters predicted by the networks. Middle row: Input data and the evaluation of the fitted surfaces at the predicted parameters. Bottom row: The original surface and the fitted bi-quadratic surfaces.

approximations for larger point clouds while needing much smaller amount of computation time. In particular, this means that BIDGCN is able to predict good parameterizations even for data that it is not of the same class as the data that it was trained on.

Comparison with standard methods on exact and noisy data

In this example, we study the performance of our neural network when applied to data from the same class as the training data, i. e. data sampled from bi-quadratic surfaces. As a benchmark, we use the three parameterization methods presented in Section 2: parameterization with UNIF, RECD, and LPSP weights. For these methods, we choose the radius for defining the local neighbourhoods adaptively depending on the number of points per point cloud as

$$r = \frac{3}{\sqrt{m}}. \tag{59}$$

Choosing the optimal radius r for these methods is a complicated manual task that, for general data, cannot be solved efficiently. Here, the factor $\sqrt{m}$ is derived from the number of points on an axis-aligned line in a uniformly distributed point cloud. The factor 3 was determined empirically to be close to optimal for the standard methods when parameterizing point clouds of 200 points.

In order to compare the methods, we evaluate them on 100 point clouds and report the MSE as well as the total computational time needed to parameterize and approximate all 100 point clouds. In particular, we study the dependence of accuracy and computation time on the number of points in each point cloud. Note that while BIDGCN was trained only on point clouds of size 1000, it is important to ensure that it gives good results for point clouds of any size. The left column of Figure 52 shows the comparison of the methods on point clouds sampled from surfaces that were generated in the same way as the training data set for our neural network, see Section 3. All the plots are semi-log plots, with a logarithmic scale used for the time and error axes. We observe that the MSEs produced by our method are much smaller than the ones resulting from the UNIF and RECD parameterizations. On the other hand, while the accuracy of our method on this synthetic data is similar to that of LPSP parameterization, the computational time of the neural network-based method is much lower than the time needed for all three other methods. In particular, LPSP is very costly, with a computational time that is about two orders of magnitude higher than the one of BIDGCN.

We perform the comparison of BIDGCN with the previously considered standard methods on noisy data. The middle column of Figure 52 shows the behaviour of all four methods when applied to data with Gaussian noise of standard deviation $5\mathrm{e}-3$, while the right column of Figure 52 shows their behaviour on surfaces with Gaussian noise of standard deviation $1\mathrm{e}-2$. We observe that when applied to noisy data, BIDGCN performs much better than the three other methods.

As in the non-noisy data case, we observe that the evaluation of our method is much faster than the evaluations of the other methods. In particular, the speed-up with respect to LPSP is significant.

A further advantage of BIDGCN is that it is very robust with respect to the presence of noise, even if the noise becomes very large. For the standard methods suitably choosing the radius that determines the local

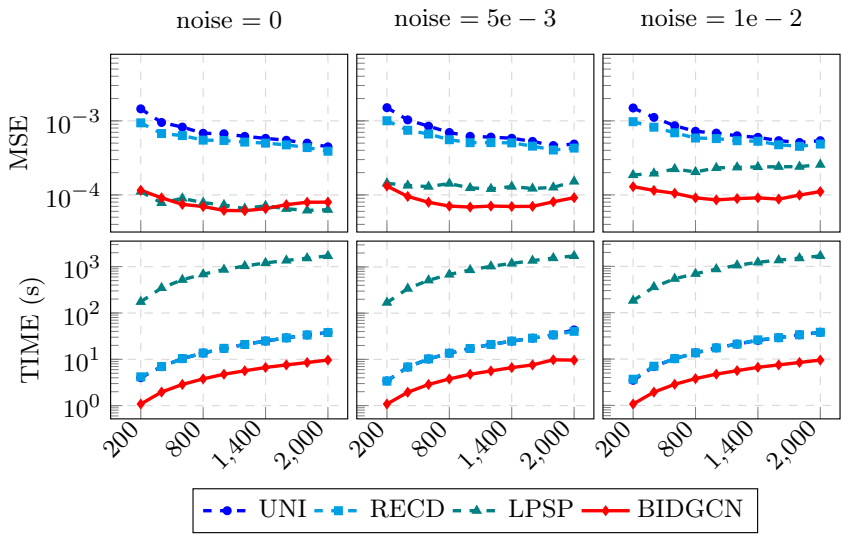


Figura 52. Comparison between BIDGCN and standard meshless parameterization methods of Section 3. MSE (top) and computation time (bottom) when applying the four methods to point clouds of different sizes sampled from 100 bi-quadratic surfaces from the same class as the training data set without noise (left), with Gaussian noise of standard deviation $5e-3$ (middle) and with Gaussian noise of standard deviation $1e-2$ (right).

neighbourhood becomes near impossible when the data is non-regularly distributed. For a fixed choice of radius or our simple adaptive choice (59), this means that these methods often fail due to neighbourhoods that are too small. In particular, LPSP needs enough points in each neighbourhood to generate a Delaunay triangulation. Figure 53 (left) shows how many surfaces out of 100 surfaces with Gaussian noise of standard deviation $5e-2$ could be successfully parameterized by the four methods. We observe that BIDGCN was always successful, while the number of surfaces that could be parameterized using LPSP strongly decreases with the number of points per point cloud. One possible way to tackle this problem could be by suitably choosing a different radius r_i for each point $i \in \mathcal{P}$; however, there is no obvious way how to realize such an adaptive local choice in a robust way. Moreover, a local choice, if one exists, would further increase the computational complexity and overall efficiency risks to deteriorate even further.

Finally, we report the MSEs of the surfaces that were successfully parameterized in Figure 53 (right). Also in this case the neural network results in significantly smaller errors.

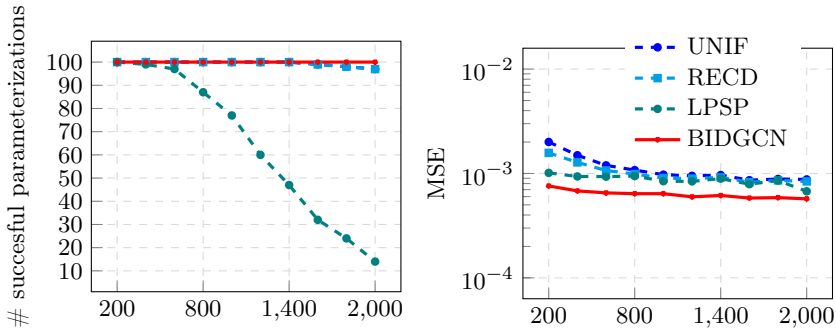


Figura 53. Number of successfully parameterized surfaces out of 100 bi-quadratic surfaces of magnitude m with added Gaussian noise of size $5e-2$ analyzed in Section 3 (left). MSE when applying BIDGCN and the three standard methods to point clouds of different sizes sampled from 100 bi-quadratic surfaces with added Gaussian noise of standard deviation $5e-2$ analyzed in Section 3 (right); all cases where a method did not succeed were removed from the computation of the MSE.

Generalization to higher degrees

In order to test the generalization properties of the neural network in terms of different degrees, we apply it to point clouds sampled from surfaces of higher bi-degree, namely $\boldsymbol{d} = (d, d)$ with $d = 3, 4, 5$, with and without Gaussian noise and we use tensor-product Bézier surfaces of the same degree for fitting the parameterized point clouds. Figure 54 shows the MSEs as well as the timings for point clouds without noise. We observe that BIDGCN results in similar MSEs compared to the LPSP parameterization. However, the computation time for BIDGCN is more than one order of magnitude smaller with respect to LPSP.

Figure 55 shows the results on point clouds with added Gaussian noise of standard deviation $1e-2$. We observe that in this case, BIDGCN results in improved MSEs compared to the LPSP parameterization, while the computation time is still more than one order of magnitude smaller.

Visual comparison of the methods

While the error in the previous examples was averaged over a large number of point clouds, we will now present particular examples that visually demonstrate the performance of the neural network in terms of quality of the reconstructed model. We applied BIDGCN as well as LPSP to the first two point clouds of size 1000 in our test data set, with and without noise, sampled from bi-quadratic surfaces. For LPSP, we compare two choices of the radius: $r = 0.09 \approx 3/\sqrt{m}$ and $r = 0.2$. The parameters, the fitted surface for both methods, the evaluation of the resultant surface at the parameters, and the input data are displayed in Figure 56. The point cloud displayed in Figure 56 (a) is noise free, whereas we added Gaussian noise with standard deviation $1e-2$ to the point cloud displayed in Figure 56 (b). We observe that the neural network predicts parameters that are visually

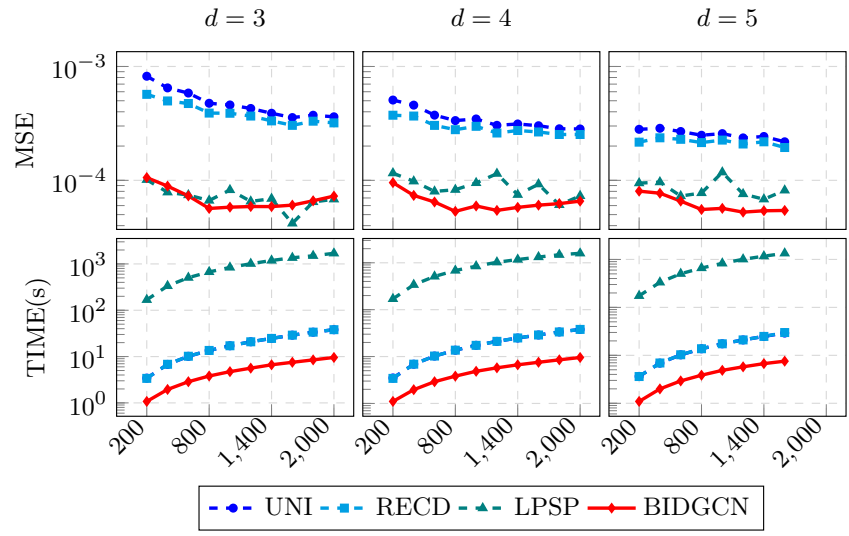


Figura 54. MSE (top) and computation time (bottom) when applying the four methods to point clouds of different sizes sampled from 100 surfaces of bi-degree $\boldsymbol{d} = (d, d)$ with $d = 3$ (left), $d = 4$ (middle) and $d = 5$ (right) considered in Section 3.

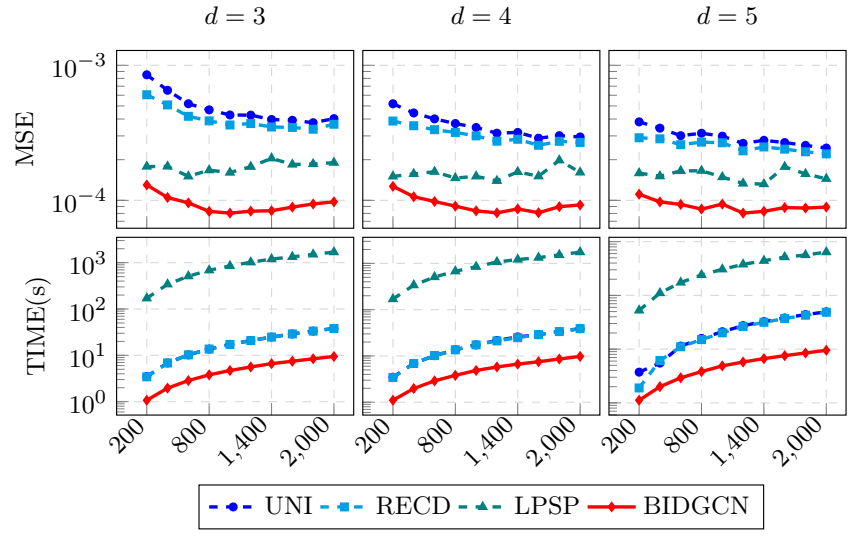


Figura 55. MSE (top) and computation time (bottom) when applying the four methods to point clouds of different sizes sampled from 100 of bi-degree $\boldsymbol{d} = (d, d)$, with $d = 3$ (left), $d = 4$ (middle) and $d = 5$ (right), with added Gaussian noise of standard deviation $1e-2$ analyzed in Section 3.

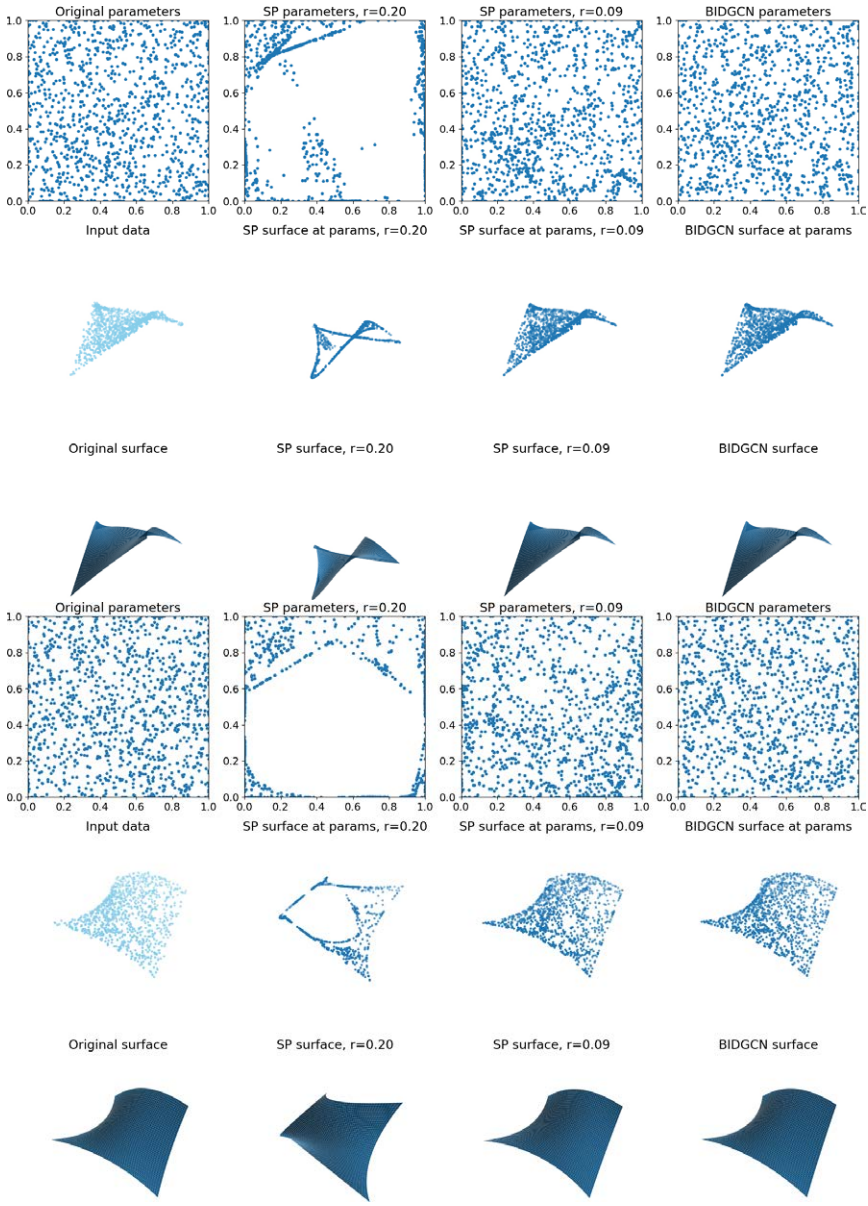


Figura 56. Visual comparison described in Section 3 of the BIDGCN and LPSP for radius $r = 0.2$ and $r = 0.09$ on a point cloud of size 1000 sampled from a bi-quadratic surface without noise (a) and with Gaussian noise of standard deviation $1e-2$ (b).

very close to the original parameters, both for the noisy and the non-noisy cases. On the other hand, the behaviour of LPSP largely depends on the choice of the radius r. For $r = 0.09$, LPSP parameterization performs

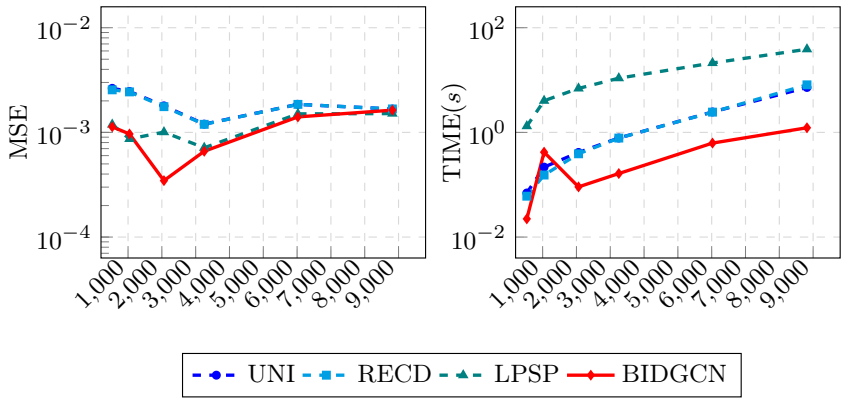


Figura 57. Quantitative comparison described in Section 3 of MSE (left) and computational time (s) (right) of the approximating bi-quadratic surface when parameterizing the point clouds of different size m sampled from the Nefertiti bust model using BIDGCN and the standard methods.

well but appears to result in larger deviations from the original parameters compared to BIDGCN. For the choice $r = 0.2$, the parameterization contains large gaps in the interior of the parameter domain. This shows that for using LPSP effectively, one needs to carefully choose the radius r, while BIDGCN is able to predict the optimal graph in the first layers of the network architecture and therefore does not require any fine-tuning.

Real-world data

In this experiment, we process real world point clouds of different sizes, representing a Nefertiti face model, and we lead a comparison between the standard parameterization methods and the proposed BIDGCN, in terms of accuracy and computational time. As concerns the standard parameterization methods, a suitable choice of the radius r needs to be selected. To produce fair comparisons and avoid unreasonable parameter value distributions, as illustrated in Figure 56, we execute a heuristic search for $r = \frac{3}{\sqrt{m}}, 0.05, 0.075, 0.1, \ldots, 0.25, 0.275, 0.3$ on each dataset by computing the polynomial approximation of bi-degree $\boldsymbol{d} = (2, 2)$, and selecting the radius r giving the best MSE. By analyzing the value of the error with respect to r, we decide to fix the radius as defined in (59) since it leads to the best approximation results for almost all the real data point clouds. Subsequently, we collect 6 data sets of $m = 532, 1043, 2062, 3253, 6024, 8818$ points, acquired from the same Nefertiti face model. We then compute the parametric values for each Nefertiti point cloud with the standard and the proposed BIDGCN methods to construct a polynomial bi-quadratic least-squares approximation. For each method, we report the MSE associated with the reconstructed polynomial surface and the total computational time

needed to parameterize and approximate each point cloud in Figure 57. In line with the trends observed on the synthetic data investigated in the previous sections, the uniform and inverse distance parameterizations lead to MSEs that are, in general, much higher than LPSP and BIDGCN. On the other hand, the error values obtained with BIDGCN and LPSP are similar, but our method scales favourably with the dimension of the point cloud always having the lowest computational time.