BIDGCN: Boundary informed dynamic graph convolutional network for adaptive spline fitting of scattered data

Carlotta Giannelli, Sofia Imperatore, Angelos Mantzaflaris, Felix Scholz

BIDGCN: Boundary informed dynamic graph convolutional network for adaptive spline fitting of scattered data

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

[1]Dipartimento di Matematica e Informatica, Università degli Studi di Firenze, Italy.
[2]Centre Inria of Université Côte d'Azur, Sophia Antipolis, France.
[3*]Institute of Applied Geometry, Johannes Kepler University Linz, Austria.

*Corresponding author(s). E-mail(s): felix.scholz@jku.at;
Contributing authors: carlotta.giannelli@unifi.it; sofia.imperatore@unifi.it;
angelos.mantzaflaris@inria.fr;

Abstract

In this work, we propose a Boundary Informed Dynamic Graph Convolutional Network (BIDGCN) characterized by a novel boundary informed input layer, with special focus on applications related to adaptive spline approximation of scattered data. The newly introduced layer propagates given boundary information to the interior of the point cloud, in order to let the input data be suitably processed by successive graph convolutional network layers. We apply our BIDGCN model to the problem of parameterizing three-dimensional unstructured data sets over a planar domain. The parameterization problem is a key step in the solution of different geometric modeling tasks and in particular for the design of surface reconstruction schemes with smooth spline surfaces. A selection of numerical examples shows the effectiveness of the proposed approach for adaptive spline fitting with (truncated) hierarchical B-spline constructions.

Keywords: Geometric deep learning, Parameterization, Surface fitting, Adaptive spline approximation, THB-splines

1 Introduction

Geometric deep learning refers to the generalization of convolutional neural networks to non-Euclidean data such as discrete manifolds, graphs and general point clouds [1]. While for data represented on a regular grid in a Euclidean space the convolution operator is uniquely defined for arbitrary dimensions, in the case of unstructured non-Euclidean data many different approaches for the definition of operators that mimic the behavior of standard convolution have been proposed [2].

Most current graph neural network architectures are classified as *message passing neural networks* [3], 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 neighbors, the adjacent vertices directly connected by an edge.

While graph neural networks have been successfully applied to classification tasks such as shape recognition, segmentation, and registration [4–7], regression tasks where the network

should predict a (potentially vector-valued) function on the input geometry are much less studied [8, 9]. Moreover, graph neural networks that are used for processing discrete surface data have mostly been applied to *closed* surfaces, i.e. surfaces without boundary. This simplifies the design of suitable graph convolution 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 these 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 problem it is then necessary to devise a network architecture that takes into account the boundary conditions in addition to the standard vertex features of the discrete surface. 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 paper, we propose a new *Boundary Informed Dynamic Graph Convolutional Network (BIDGCN)* for processing scattered point clouds based on the dynamic edge convolution introduced in [10]. The distinctive feature of this *dynamic* edge convolution is that the graph used for message passing is recomputed after each layer according to the new features, thereby enabling the network to predict the graph and to transport the features arbitrarily far in the point cloud. The key idea of the new BIDGCN is that we regard the 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. To summarize, the main advantages of the proposed network layer are: (i) the ability to incorporate boundary conditions in its prediction, (ii) the dynamic prediction of the graph used for message passing. The second property is directly inherited from the dynamic edge convolution approach [10].

As a use case for our new network architecture, we consider its application to the problem of scattered point cloud parameterization. The parameterization problem is a key step in different geometric modeling tasks and its solution is usually the starting point of various data processing algorithms. Among others, it is essential for approximating a scattered three-dimensional point cloud with high-order geometries, such as B-splines, NURBS or locally refined (hierarchical) B-splines. Industrial applications include, for example, a variety of reverse engineering problems, as the adaptive re-construction of geometric models from a 3D scan [11–13]. A class of methods for scattered point cloud parameterization was proposed in [14] building upon the methods from [15] for the related problem of parameterizing triangulated surface data. In these methods, the one-dimensional boundary of the point cloud is parameterized separately, which is a much easier problem. The parameterization of the interior vertices is then found by assuming that the parameter for each interior vertex is a convex combination of the parameters of a certain vertex neighborhood. The specific choices of neighbors and coefficient values considered in the convex combinations determine the particular method from this class, see [14]. Recently, machine learning tools have been applied to the problem of determining these coefficients [16, 17]. In particular, [16] builds upon [15] and trains a network for a fixed triangular mesh, whereas [17] provides a more general meshless data-driven approach based on [14]. While these methods lead to an improved performance compared to the standard choices of coefficients, they also inherit their limitations in terms of efficiency, robustness, and sensitivity to parameter-dependent graph connectivity.

In our approach, we assume that the boundary is already parameterized, as in the standard methods. We then use a network architecture based on

the new BIDGCN model to predict the parameterization of the interior vertices. This leads to three key 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. Since the training process only has to be completed once, this results in a large advantage in terms of computation time.
2. *Robustness*: Our method is robust with respect to noise and results in much better approximations when approximating noisy point clouds with polynomial surfaces. In addition, it overcomes failing issues in the solution of the involved linear systems in case of sparse neighborhoods.
3. *Automatic graph prediction*: While for the classical methods it is necessary to construct a graph by suitably choosing the local neighborhoods of each interior vertex, our method automatically predicts a suitable graph without the need of free parameter selection.

In order to achieve an effective scattered data approximation scheme, we suitably exploit the BIDGCN parameterization as starting point for adaptive fitting with truncated hierarchical B-splines (THB-splines) [18]. The performance of the approximation scheme are then enhanced by introducing a parameter correction step [19] within each iteration of the adaptive strategy. A similar idea has been investigated in the case of *structured* data fitting with T-splines in [20, 21] and for template mapping with adaptive splines in [22].

The structure of the paper is as follows. Section 2 introduces the new boundary informed edge convolution. In Section 3, we summarize the point cloud parameterization problem, that is then addressed in Section 4 by developing a boundary informed dynamic graph convolutional network devoted to scattered data parameterization. Section 5 presents the adaptive fitting algorithm with moving parameters. A selection of numerical examples on synthetic and real data sets is presented in Section 6. In particular, the generalization capabilities of the proposed parameterization model are highlighted within the framework of an adaptive surface reconstruction scheme based on THB-splines. Finally, Section 7 concludes the paper.

2 Boundary informed edge convolution

In this section, after briefly describing the classic dynamic edge convolution operator, we present our new boundary informed dynamic convolutional network architecture.

2.1 Dynamic edge convolution

We assume to be given a point cloud $\mathcal{P}$ together with vertex features $x_i \in \mathbb{R}^n$, where $n \in \mathbb{N}$ is the input feature dimension. The dynamic edge convolution operator defined in [10] computes new features $y_i \in \mathbb{R}^m$ for all points in $\mathcal{P}$, where $m \in \mathbb{N}$ is the output feature dimension. To this end, first the k-nearest neighbor graph $\mathcal{G}_k$ 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 neighbors of i with respect to the Euclidean distance

$$\|x_i - x_j\|$$

of the input features. In the next step, edge features are computed for each directed edge in $\mathcal{G}_k$ as follows. For all $(j, i) \in \mathcal{G}_k$

$$e_{ji} = h_\Theta(x_i, x_j - x_i),$$

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

$$h_\Theta : \mathbb{R}^{2n} \to \mathbb{R}^m$$

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

$$x_i' = \square_{(j,i)\in\mathcal{G}_k} e_{ji},$$

where $\square$ can be the sum, the mean, or the maximum value. In a *dynamic* edge convolution network, the k-nearest neighbor graph is recomputed after each layer with respect to the new features 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. We refer to these graph neural networks characterized by dynamic edge convolution operators as DGCNN (Dynamic Graph Convolutional Neural Networks).

Finally, in the examples presented in this paper, we replace the k-nearest neighbor graph by the radius graph, meaning that instead of specifying the number of neighbors k, we specify a radius $r > 0$ and add directed edges (i, j) and (j, i) for all $i, j \in V$ such that

$$\|x_i - x_j\| \leq r.$$

By dynamically recomputing the radius graph instead of the k-nearest neighbor 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 neighborhoods with a fixed maximum distance, while a k-nearest graph might associate points with very large distances if the point cloud is locally sparse.

2.2 BIDGCN: Boundary Informed Dynamic Graph Convolutional Network

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. It then 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 Network (BIDGCN)*.

More precisely, we assume that the input point cloud is decomposed as $\mathcal{P} = \mathcal{P}_\mathrm{B} \cup \mathcal{P}_\mathrm{I}$ into boundary and interior points. Moreover, we assume that each vertex $i \in \mathcal{P}_\mathrm{I}$ comes with features $x_i \in \mathbb{R}^n$ and every vertex $j \in \mathcal{P}_\mathrm{B}$ comes with features $(x_j, u_j) \in \mathbb{R}^n \times \mathbb{R}^d$, were we regard $u_j \in \mathbb{R}^d$ as boundary conditions.

We first compute two different radius graphs

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

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

$$(j, i) \quad \text{with} \quad i, j \in \mathcal{P}_\mathrm{I} : \|x_i - x_j\| \leq r,$$

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

$$(k, i) \quad \text{with} \quad i \in \mathcal{P}_\mathrm{I},\ k \in \mathcal{P}_\mathrm{B} : \|x_i - x_k\| \leq r.$$

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

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

$$e_{ji} = h_\Theta(x_i, x_j - x_i),$$

where h is a feed-forward neural network

$$h_\Theta : \mathbb{R}^{2n} \to \mathbb{R}^m$$

with output feature size m and learnable weights Θ.

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

$$e_{ik} = g_\Phi(x_i, x_k - x_i, u_k),$$

where g is another feed-forward neural network

$$g_\Phi : \mathbb{R}^{2n+d} \to \mathbb{R}^m$$

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 $\mathcal{P}_\mathrm{I}$. For $i \in \mathcal{P}_\mathrm{I}$, we have

$$x_i' = \left(\square_{j:(j,i)\in\mathcal{G}_{\mathrm{I}\to\mathrm{I}}} e_{ji}\right) \square \left(\square_{k:(k,i)\in\mathcal{G}_{\mathrm{B}\to\mathrm{I}}} e_{ki}\right),$$

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 m for the interior point cloud $\mathcal{P}_\mathrm{I}$. During training Θ and Φ are optimized simultaneously. The flow of the input layer is depicted in Fig. 1 and its application to an interior vertex is illustrated in Fig. 2.

2.3 Hidden and output layers

The input layer gives as output new features in $\mathbb{R}^m$ for the interior point cloud $\mathcal{P}_\mathrm{I}$. To further process these features, we proceed like in the original dynamic edge convolution network by computing a new graph for $\mathcal{P}_\mathrm{I}$ based on the new features

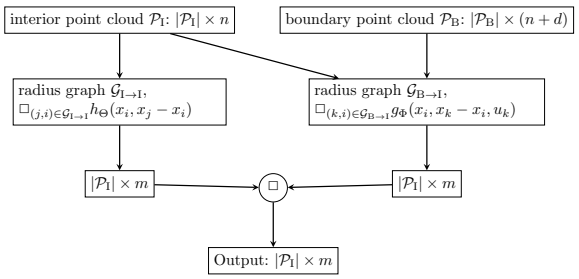

Fig. 1: Boundary informed input layer.

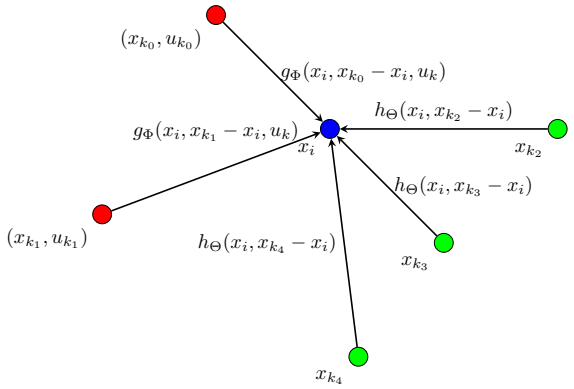

Fig. 2: Edge contributions of the input layer for x_i (blue) computed from the features at the neighboring boundary vertices (red) and interior vertices (green).

$x_i' \in \mathbb{R}^m$. As a slight modification, while in [10] the k nearest-neighbor graph was used, we propose to use the radius graph also in the hidden layers. This makes our network architecture more independent of the sampling density of the point cloud. Since the number of neighbors 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.

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

3 Scattered point cloud parameterization for surface fitting

In this section, we introduce the problem of scattered point cloud parameterization for approximation with smooth surfaces. Moreover, we shortly summarize the standard methods for this problem. In the following section, we will then report on how we use a network architecture based on BIDGCN to tackle this problem.

3.1 Problem formulation

For a given point cloud $\mathcal{P}$ such that each point $i \in \mathcal{P}$ is equipped with a feature $x_i \in \mathbb{R}^3$, a *parameterization* is defined as a mapping

$$u : \mathcal{P} \to \Omega,$$

where $\Omega \subset \mathbb{R}^2$ is called the *parameter domain*. We write $u_i = u(i)$ for the parameter of $i \in \mathcal{P}$.

Finding a suitable parameterization of a point cloud is an important step in different geometry processing tasks. In particular, we consider the task of approximating a point cloud sampled from a surface with a smooth parametric surface

$$S : [0,1]^2 \to \mathbb{R}^3$$

with

$$S(u) \coloneqq \sum_{j=0}^{N} c_j B_j(u), \text{ for } u \in \Omega \qquad (1)$$

defined in the spline space spanned by the basis (B_j). In this paper, we consider either the tensor-product Bernstein polynomials or (truncated hierarchical) B-splines as basis functions B_j.

The objective of surface fitting is to find the best approximating surface, i.e.

$$\min_{c_0,\dots,c_N,u_1,\dots,u_{|\mathcal{P}|}} \sum_{i=1}^{|\mathcal{P}|} \|S(u_i) - x_i\|^2 .$$

Optimizing the parameterization u as well as the surface S at the same time is a complicated non-linear problem that can be simplified by separating the optimization with respect to the

parameterization $u_i \in [0,1]^2$ and the optimization with respect to the control points $c_j \in \mathbb{R}^3$.

Once a parameterization u of $\mathcal{P}$ has been determined, the best-approximating surface S can be found by solving the linear least-squares problem

$$\min_{c_0,\dots,c_N} \sum_{i=1}^{|\mathcal{P}|} \left\| \sum_{j=0}^{N} c_j B_j(u_i) - x_i \right\|^2 . \tag{2}$$

using standard techniques from computational linear algebra.

Our goal is then to look for the parameterization u that minimizes the residual of (2).

3.2 Standard methods

In order to demonstrate the efficiency and accuracy of our proposed method, we will compare its performance with the three parameterization methods presented in [14]: *uniform* parameterization, *inverse distance* parameterization and *shape preserving* (SP) parameterization. In these three methods the parameterization of the boundary of the point cloud is first determined separately. This can be done using standard methods such as uniform, chord length and centripetal parameterization [23] and their improvements [24, 25], using iterative schemes [26, 27] as well as using feed-forward neural networks [28, 29].

Then, a graph of the input point cloud $\mathcal{P} = \mathcal{P}_\mathrm{I} \cup \mathcal{P}_\mathrm{B}$ is constructed by choosing neighborhoods $N_i \subset \mathcal{P}$ for all $i \in \mathcal{P}_\mathrm{I}$. Once the parameterization weights $\lambda_{ik} > 0$ with $\sum_{k\in N_i} \lambda_{ik} = 1$ are chosen, the parameterization is determined by the system of equations

$$u_i = \sum_{k\in N_i} \lambda_{ik} u_k, \tag{3}$$

where the given parameters u_j for $j \in \mathcal{P}_\mathrm{B}$ serve as boundary conditions.

For the *uniform* and the *inverse distance* parameterization, the neighborhoods N_i are chosen as $N_i = \{k \in \mathcal{P} : \|x_k - x_i\| \le r\}$ for some radius r and the weights are set to

$$\lambda_{ik} = \frac{1}{|N_i|} \quad \text{(uniform)}$$

or to

$$\lambda_{ik} = \frac{\frac{1}{\|x_k - x_i\|}}{\sum_{\ell\in N_i} \frac{1}{\|x_\ell - x_i\|}} \quad \text{(inverse distance)}.$$

For the shape preserving weights, the neighborhoods N_i are chosen by first projecting all points x_k in a large ball neighborhood of x_i onto their best approximating plane and then finding a Delaunay triangulation of the resulting planar point cloud. The neighbors of i in the Delaunay triangulation form the neighborhood N_i and the weights λ_{ij} are the shape preserving weights for triangulations presented in [15]. Their defining property is that they preserve the distribution of the angles of the triangulation around x_i.

4 Point cloud parameterization using BIDGCN

In order to predict the optimal parameterization we propose a network architecture based on BIDGCN. As detailed in the previous section, the task for our trained neural network is to predict parameters $u_i \in [0,1]^2 \subset \mathbb{R}^2$ for the given point cloud such that the resulting solution $c_0, \dots c_N \in \mathbb{R}^3$ of (2) results in the smallest possible residual.

The neural network should then approximate an operator that assigns to each sufficiently large set of input features a parameterization that is optimal with respect to (2). Without fixing some of the parameter values a priori, this operator is not uniquely defined, making it difficult to directly train neural network for this task. Inspired by the standard methods summarized in Section 3.2, we fix the parameters u_j for the boundary points using the standard one-dimensional parameterization algorithms. 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.

4.1 Network architecture

We design a neural network based on the new boundary informed input layer described in Section 2. More precisely, our network consists of the boundary informed input layer, four hidden dynamic edge convolution layers, and a multi-layer perceptron as the output layer. In the input layer, we have two multi-layer perceptrons (MLP), one for the edges in $\mathcal{G}_{\mathrm{I}\to\mathrm{I}}$ and one for the edges $\mathcal{G}_{\mathrm{B}\to\mathrm{I}}$. For $\mathcal{G}_{\mathrm{I}\to\mathrm{I}}$ we train an MLP with layer sizes $\{6, 64, 64\}$, while for $\mathcal{G}_{\mathrm{B}\to\mathrm{I}}$ we train a MLP with layer sizes $\{8, 64, 64\}$. Note that the input dimension corresponds to the edge features in the two different graphs. 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. 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 $\{320, 256, 256, 2\}$.

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 our network 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. The architecture of our point cloud parameterizing network is shown in Fig. 3.

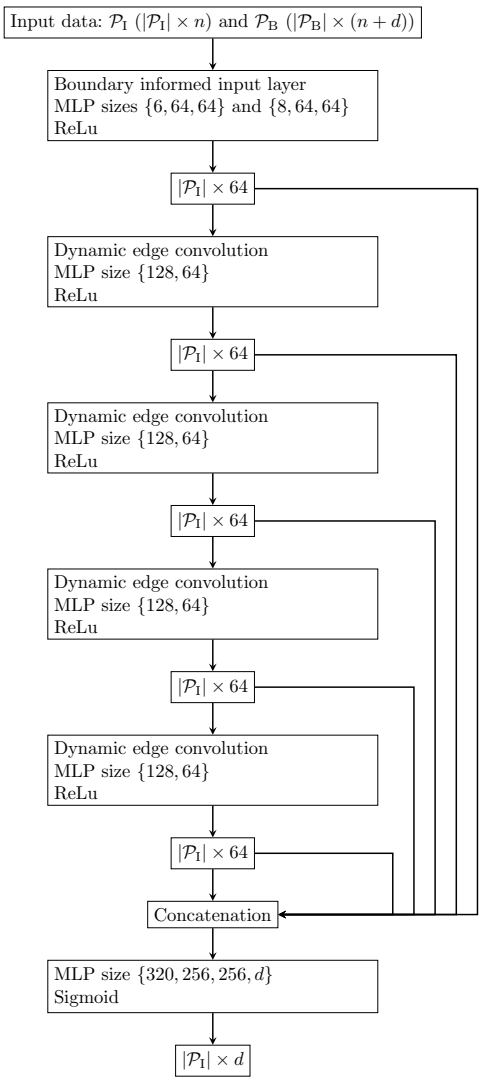

Fig. 3: Network architecture of the point cloud parameterizing network, $n = 3$ is the dimension of the point cloud and $d = 2$ is the parametric dimension.

4.2 Training data

In our training, we employ tensor-product Bézier surfaces, meaning that $B_j(u) = B_{ab}(u_1, u_2)$ are the tensor-product Bernstein polynomials of bidegree (p_1, p_2), defined as

$$B_{ab}(u_1, u_2) = \binom{p_1}{a}\binom{p_2}{b}(1-u_1)^{p_1-a}(u_1)^a(1-u_2)^{p_2-b}(u_2)^b,$$

where $a(j), b(j)$ is an ordering of $\{0, \ldots, p_1\} \times \{0, \ldots, p_2\}$. This choice of the discrete space has the advantage that for a sufficiently large point cloud the best approximation with respect to a fixed parameterization of the point cloud is well defined without further regularization.

In order to train the network for parameterizing scattered point cloud data, we generate a data set consisting of 100,000 point clouds. Each point cloud $\mathcal{P} = \mathcal{P}_\mathrm{I} \cup \mathcal{P}_\mathrm{B}$ 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. The

point clouds are sampled from biquadratic tensor-product Bézier surfaces whose control points c_{ij} were 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 of complexity of the surfaces while avoiding self-intersections. In order to achieve rotation-invariance, we rotate each surface around a randomly sampled axis in $\mathbb{R}^3$ by a random angle. Besides the vertex features $x_i \in \mathbb{R}^3$ for all interior and boundary points, we store the exact parameters $u_i \in \mathbb{R}^2$ for all boundary points.

Note that the choice of point cloud size in the 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.

4.3 Normalization

Before evaluating the network on a point cloud from the training, validation and test data sets or from a real-world sample, we normalize the vertex features $x_i \in \mathbb{R}^3$ 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 $u_i \in \mathbb{R}^2$.

4.4 Loss function

As we want to train the neural network to predict parameters that minimize (2), we follow an unsupervised strategy and use the predicted parameters for the interior points as well as the prescribed parameters for the boundary points to fit a biquadratic polynomial surface to the point cloud. More precisely, we assemble the collocation matrix

$$A_{ij} = B_j(u_i),$$

for $i = 1, \dots, |\mathcal{P}|$ and $j = 0, \dots, 8$. Here, B_j are the biquadratic tensor-product Bernstein polynomials and $u_i \in [0, 1]^2$ are the prescribed parameters if $i \in \mathcal{P}_\mathrm{B}$ and the parameters predicted by the neural network if $i \in \mathcal{P}_\mathrm{I}$.

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

$$b_i = x_i.$$

We then have reformulated (2) into the linear least squares problem

$$\min_{c \in \mathbb{R}^{9 \times 3}} \|Ac - b\|^2 \tag{4}$$

that we solve using QR decomposition, 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 (4).

We train the network using stochastic gradient descent. We start the training with learning rate 0.1 and decrease the learning rate whenever the loss plateaus.

5 Adaptive THB-spline fitting with moving parameters

The need of automatic representations in terms of free-form spline models which ensure sufficient flexibility, effective geometric modeling options, high approximation accuracy, while simultaneously preserving low computational costs, led to the developement of (truncated) hierarchical B-spline (THB-spline) adaptive approximation schemes, see e.g. [30] among others. The truncated basis for hierarchical B-splines was originally introduced in [18] to recover the partition of unity property in the hierarchical spline setting and reduce the interaction of hierarchical B-splines inserted at different refinement levels.

We consider a hierarchical spline model $S : [0, 1]^2 \to \mathbb{R}^3$ of the form (1) that approximates the input point cloud $\mathcal{P}$, where $B_j : [0, 1]^2 \to \mathbb{R}$ are bivariate THB-splines [18]. State of the art of scattered point cloud fitting schemes address the data parameterization and the design of the hierarchical spline model separately, see [11–13]. Given a point cloud $\mathcal{P}$, a suitable parameterization $\{u_1, \dots, u_{|\mathcal{P}|}\}$ is initially computed and, subsequently, on such a *fixed* parameterization, the hierarchical spline space is iteratively refined within the adaptive approximation scheme. We here consider as a starting point the adaptive

global THB-spline least squares approach, originally proposed in [11]. Let the features $x_i \in \mathbb{R}^3$ be the Cartesian coordinates of the items $i \in \mathcal{P}$, and $\{u_1, \dots, u_{|\mathcal{P}|}\}$ a suitable parameterization. By starting with the initial polynomial approximation, a THB-spline approximation is iteratively computed until the error satisfies a given tolerance or a maximum number of iterations is reached. At each iteration of the adaptive loop a set of elements is marked for refinement based on the error computation, and the hierarchical mesh is subsequently refined. More precisely, the iterative approximation scheme is characterized by four main steps. First, a solution of the current fitting problem is computed. In particular, for a fixed (TH)B-spline space, the control points in (1) are computed by solving the penalized least squares problem

$$\min_{c_0,\dots,c_N} \frac{1}{2} \sum_{i=1}^{|\mathcal{P}|} \|S(u_i) - x_i\|_2^2 + \lambda J(c_0, \dots, c_N), \tag{5}$$

where the regularization term J is the thin-plate energy functional, i.e.

$$J(c_0, \dots, c_N) = \int_{[0,1]^2} \left\| \frac{\partial^2 S}{\partial u_1^2} \right\|_2^2 + \left\| \frac{\partial^2 S}{\partial u_1 \partial u_2} \right\|_2^2 + \left\| \frac{\partial^2 S}{\partial u_2^2} \right\|_2^2 du_1 du_2,$$

whose influence is tuned by the weight $\lambda \geq 0$.

In the second step of each iteration, the THB-spline approximant is evaluated on the parameter sites $\{u_1, \dots, u_{|\mathcal{P}|}\}$ related to the data points $\mathcal{P}$ and the pointwise distance $\|S(u_i) - x_i\|_2^2$ is measured for each $i \in \mathcal{P}$. The pointwise error constitutes the error indicator on which the last two steps (marking and refinement) of the adaptive scheme are developed. The parameter values $u_i \in [0,1]^2$ associated to the points $i \in \mathcal{P}$ whose error exceeds a certain input threshold $\epsilon > 0$ are marked to be refined, since they identify regions of the parametric domain that need to be refined. The hierarchical mesh elements that contain at least one marked parameter $u_i \in [0,1]^2$ are dyadically refined, together with a certain number of neighboring cells, see [11] for detailed information. The iteration of these four steps is performed until the maximum point wise errors is within an input tolerance or a maximum number of iterations is reached.

Note that any adaptive fitting scheme of this kind is characterized by an iterative enrichment of the approximation space to progressively improve the accuracy of the result. Consequently, in our setting, even if the the parametric values associated to the input observations are initially (quasi-)optimal, there is no evidence that optimality is preserved when the hierarchical spline space is adaptively refined. We then introduce a parameter correction step [19] at every iteration of the fitting scheme, after the (local) refinement step.

The parameter correction approach addresses an optimization problem to identify the optimal intrinsic parameterization of a given spline model. The points $\mathcal{P}$ are projected on the current spline model $S(u)$ and for each $i \in \mathcal{P}$, its footpoint $S(\hat{u}_i)$ is identified. Hence, if $\|S(\hat{u}_i) - x_i\|_2 < \|S(u_i) - x_i\|_2$, the new (corrected) parameter for the point $i \in \mathcal{P}$ corresponds to $\bar{u}_i = \hat{u}_i$, otherwise $\bar{u}_i = u_i$. Note that, for each $i = 1, \dots, |\mathcal{P}|$, the projection is performed by considering the problem

$$\hat{u}_i = \min_{u \in [0,1]^2} \|S(u) - x_i\|_2^2, \tag{6}$$

which can be solved, for example, with a Newton-like method. Once the updated parameter values $\{\bar{u}_1, \dots, \bar{u}_{|\mathcal{P}|}\}$ are computed, the surface is also updated, and a new projection can be performed. Very few iterations of parameter correction are usually enough to obtain higher accuracy. Since we here repetitively applied the parameter update within the adaptive loop, a single correction step at any iteration is considered to suitably update the fitting result with respect to the current hierarchical spline configuration.

Note that, as shown in the last examples of the next section, the initial parameterization plays a key role to obtain high quality results.

6 Numerical results

We implemented our method using the PyTorch [31] and PyG [32] libraries. We tested the deep learning model on unseen data sets, generated in the same way as the training data, as described in Section 4.2, as well as on a number of real-world data sets. Moreover, we also consider *noisy* data, where we added Gaussian noise with varying standard deviation. We start with an

ablation study (Section 6.1) 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 3.2 (see Sections 6.2, 6.3, 6.4, and 6.5). In order to further test the generalization capabilities of BIDGCN on complex data sets and adaptive spline constructions, we consider in Sections 6.6 and 6.7 the adaptive THB-spline approximation scheme with moving parameters introduced in the previous section.

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.

6.1 Ablation study

As previously described, the parameterization problem cannot be solved without taking into account the parameterization of the boundary curves. In order to demonstrate this necessity empirically and also to show that it is fulfilled by our new boundary informed layer, we first compare the performance of our network with a trained DGCNN, see Section 2, whose architecture is equal to the one of our point cloud parameterizing network shown in Fig. 3, but with a standard dynamic edge convolution layer as input layer instead of the boundary informed input layer. This means that for boundary vertices, this network only acts on the positional features in the same way as it acts on interior layers.

In Fig. 4, we show the results on two surfaces from our test data set. 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 biquadratic surface. Quantitatively, DGCNN results in a mean squared error of $7.49 \cdot 10^{-4}$ for the surface on the left and $5.25 \cdot 10^{-4}$ for the surface on the right while BIDGCN results in a mean squared error one order of magnitude smaller, namely $5.26 \cdot 10^{-5}$ and $3.50 \cdot 10^{-5}$, respectively.

When looking the average mean squared error over 100 point clouds that were sampled from quadratic surfaces in our test set, we observe that DGCNN resulted in a mean squared error of $2.93 \cdot 10^{-3}$ and BIDGCN resulted in a mean squared error of $6.38 \cdot 10^{-5}$. This shows that the boundary informed layer results in a large improvement of the accuracy. This is in line with the theoretical consideration, that the optimal parameterization that is to be predicted is only uniquely defined when taking into account boundary conditions. As the network architectures of the two networks is similar, their evaluation times does not differ significantly.

6.2 Point clouds belonging to the test data set

In our first experiment, 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 biquadratic surfaces. As a benchmark, we use the three parameterization methods presented in [14], parameterization with *uniform*, *inverse distance* and *shape preserving* weights. For these methods, we choose the radius for defining the local neighborhoods adaptively depending on the number of points per point cloud as

$$r = \frac{3}{\sqrt{|\mathcal{P}|}}. \tag{7}$$

Choosing the optimal radius r for these methods is a complicated manual task that for general data cannot be solved efficiently. Here, the factor $\sqrt{|\mathcal{P}|}$ 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 mean squared error as well as the total computation time needed to parameterize and approximate all 100 point clouds.

In particular, we study the dependence of the accuracy and the computation time on the number of points in each point cloud. Note that while our network 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

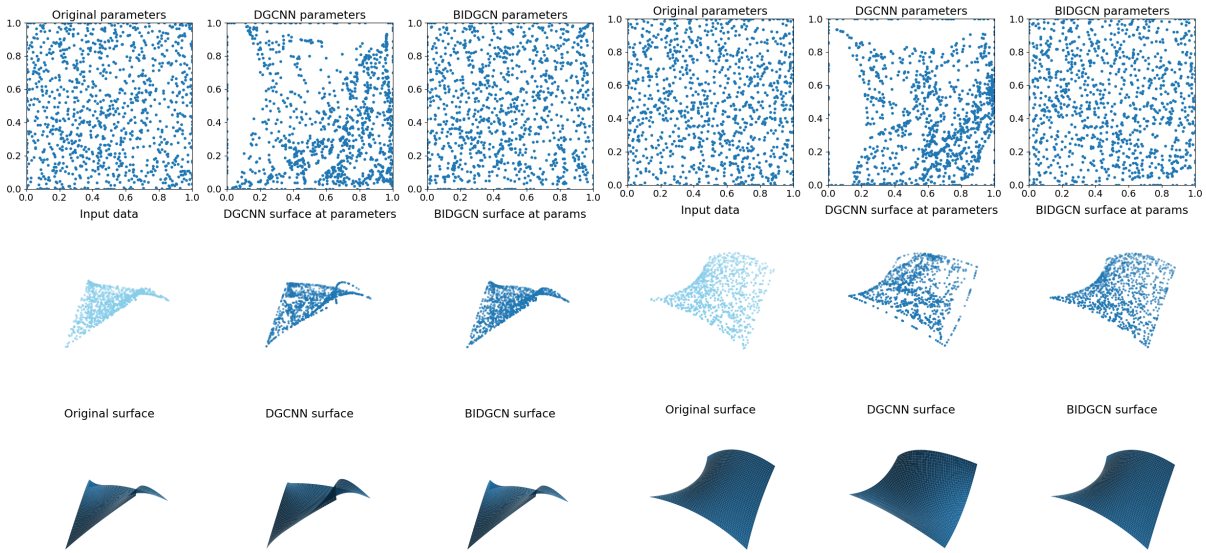

Fig. 4: Comparison of our BIDGCN network with a pure dynamic edge convolution network trained for the point parameterization problem. Top row: Original parameters and parameters predicted by the networks. Middle row: Input data and the evaluation of the fitted surfaces at the predicted parameters. Bottom row: The original surface and the fitted biquadratic surfaces.

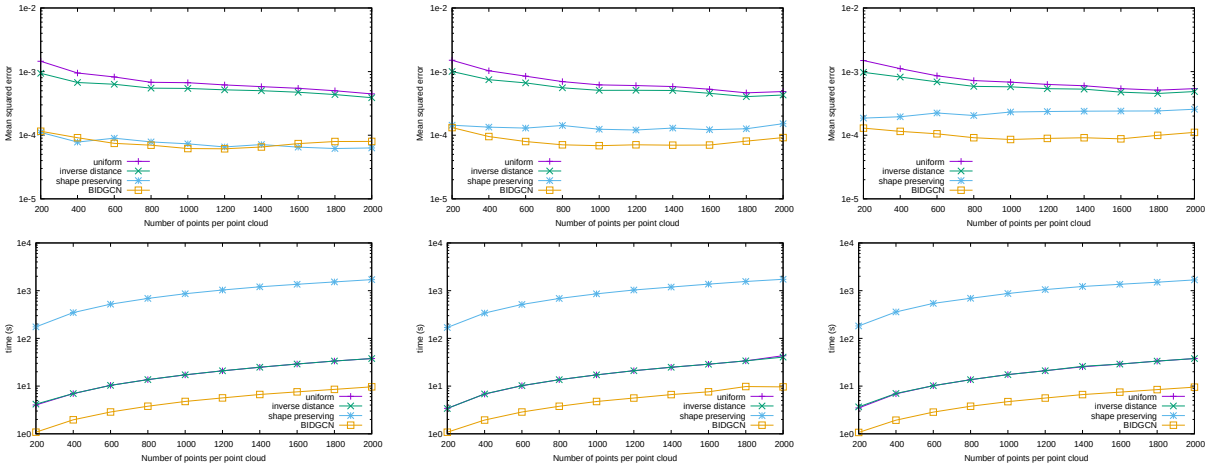

Fig. 5: Mean squared error (top) and computation time (bottom) when applying the four methods to point clouds of different sizes sampled from 100 biquadratic surfaces from the same class as the training data set without noise (left), with Gaussian noise of standard deviation 0.005 (middle) and with Gaussian noise of standard deviation 0.01 (right).

of Fig. 5 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, described in Section 4.2. All the plots are semi-log plots with a logarithmic scale used for the time and error axes. We observe that the mean squared errors produced by our method are much smaller than the ones resulting from the uniform and inverse distance parameterization. On the other hand, while the accuracy of our method on this synthetic data is similar to the one of the shape preserving parameterization, the computation time of the neural network based method is much lower than the

time needed for all three other methods. In particular, SP is very costly, with a computation time which is about two orders of magnitude higher than the one of BIDGCN. Moreover we note that, even if the network was trained exclusively on data with 1000 points per point cloud, the performance of BIDGCN does not depend on the size of the scattered data set.

6.3 Evaluations on noisy data

Measured real-world data are always subjected to noise. For this reason, we study the behavior of the trained network and the benchmark methods when applied to noisy data. We evaluate all methods on 100 point clouds sampled from surfaces that were generated as described in Section 4.2. We added Gaussian noise of standard deviation 0.005 and 0.01 to all surfaces.

The middle column of Fig. 5 shows the behavior of all four methods when applied to data with Gaussian noise of standard deviation 0.005, while the right column of Fig. 5 shows their behavior on surfaces with noise of size 0.01. We observe that when applied to noisy data, BIDGCN performs much better than the three other methods. This means that the network is able to predict good parameterizations even for data that it was not trained on. 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 SP 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 neighborhood becomes near impossible when the data is non-regularly distributed. For a fixed choice of radius or our simple adaptive choice (7), this means that these methods often fail due to neighborhoods that are too small. In particular, SP needs enough points in each neighborhood to generate a Delaunay triangulation. Fig. 6 shows how many surfaces out of 100 surfaces with Gaussian noise of standard deviation 0.05 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 SP strongly decreases with the number of points per point

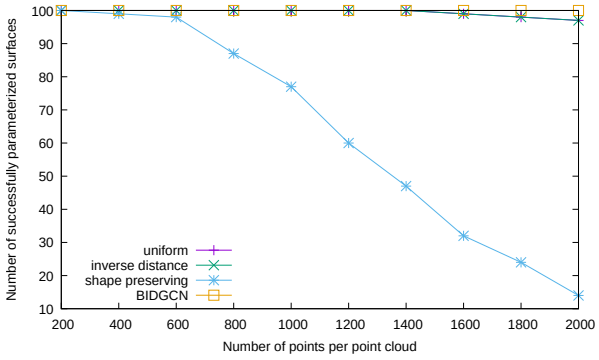

Fig. 6: Number of successfully parameterized surfaces out of 100 biquadratic surfaces with added Gaussian noise of size 0.05.

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.

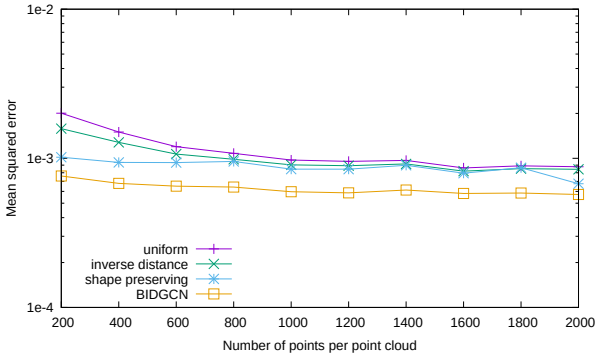

Fig. 7: Mean squared error when applying the four methods to point clouds of different sizes sampled from 100 biquadratic surfaces with added Gaussian noise of standard deviation 0.05. All cases where a method did not succeed were removed from the mean.

Finally, we report the mean squared errors of the surfaces that were successfully parameterized in Fig. 7. Also in this case the neural network results in significantly smaller errors.

6.4 Generalization to higher degrees

In order to test the generalization properties of the neural network in terms of degree, we apply it to point clouds sampled from surfaces of higher bidegree, namely (p, p) with $p = 3, 4, 5$, with and without Gaussian noise. In doing so, we use tensor-product Bézier surfaces of the same degree for fitting the parameterized point clouds. Fig. 8 shows the mean squared errors as well as the timings for point clouds without noise. We observe that BIDGCN results in similar mean squared errors compared to the SP parameterization. However, the computation time for BIDGCN is more than one order of magnitude smaller with respect to SP.

Fig. 9 shows the results on point clouds with added Gaussian noise of standard deviation 0.01. We observe that in this case BIDGCN results in improved mean squared errors compared to the SP parameterization, while the computation time is still over one order of magnitude smaller.

6.5 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 approximation power of the neural network. We applied the BIDGCN as well as SP to the first two point clouds of size 1000 in our test data set, with and without noise, sampled from biquadratic surfaces. For SP we compare two choices of the radius: $r = 0.09 \approx 3/\sqrt{|\mathcal{P}|}$ and $r = 0.2$. Fig. 10 and 11 show the parameters, the evaluation of the resulting surface at the parameters as well as the fitted surface for both methods as well as the input data. While the point cloud in Fig. 10 does not have any noise, we added Gaussian noise of standard deviation 0.01 to the point cloud shown in Fig. 11. We observe that the neural network predicts parameters that are visually very close to the original parameters, both for the noisy and the non-noisy case. On the other hand, the behavior of SP largely depends on the choice of the radius r. For $r = 0.09$, SP parameterization performs 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 SP 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.

6.6 Reconstruction and comparison on real-world data

In this experiment we process real world point clouds of different size, 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 Fig. 10 and 11, we execute an heuristic search for $r = \frac{3}{\sqrt{|\mathcal{P}|}}, 0.05, 0.075, 0.1, \ldots, 0.25, 0.275, 0.3$ on each dataset by computing the polynomial approximation of bidegree $(2, 2)$, and select the radius r giving the best mean squared error. By analyzing the value of the error with respect to r, we decide to fix the radius as defined in (7) since it leads to the best approximation results for almost all the real data point clouds.

As first analysis, we collect 6 data sets of $532, 1043, 2062, 3253, 6024, 8818$ 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 subsequently construct a polynomial biquadratic least squares approximation. For each method we report the mean squared error associated with the reconstructed polynomial surface and the total computational time needed to parameterize and approximate each point cloud in Fig. 12. In line with the trends observed on the synthetic data investigated in the previous sections, the uniform and inverse distance parameterizations lead to mean squared errors that are, in general, much higher than SP and BIDGCN. On the other hand, the error values obtained with BIDGCN and SP are similar but our method scales favourably with the dimension of the point cloud always having the lowest computational time.

As a second analysis on the Nefertiti model, we consider the adaptive THB-spline fitting of

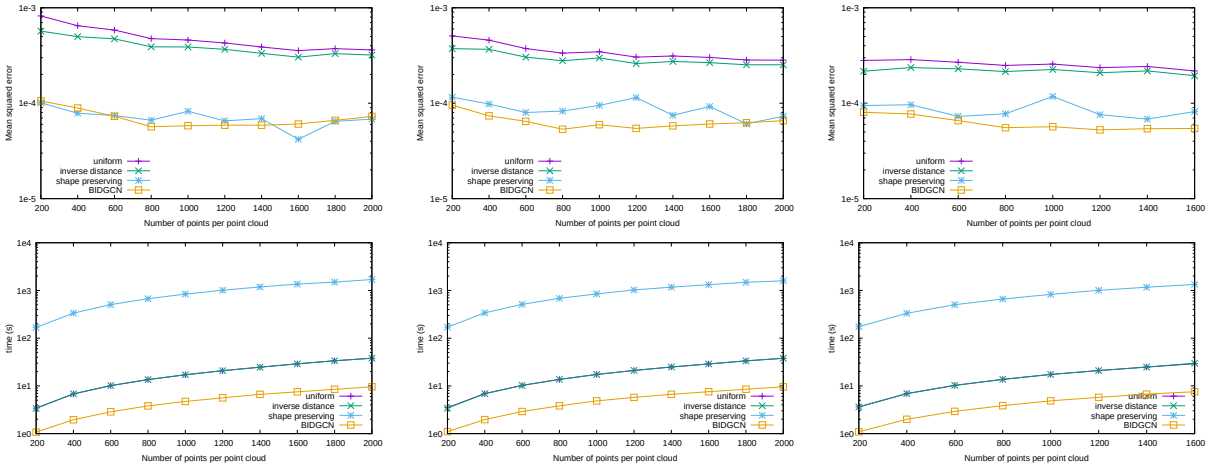

Fig. 8: Mean squared error (top) and computation time (bottom) when applying the four methods to point clouds of different sizes sampled from 100 surfaces of bidegree $(3,3)$ (left), $(4,4)$ (middle) and $(5,5)$ (right).

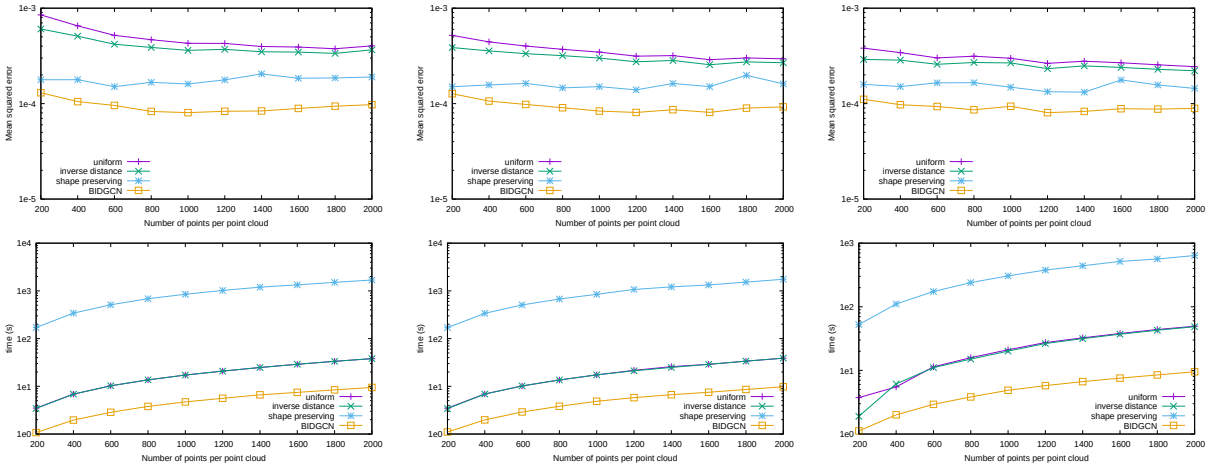

Fig. 9: Mean squared error (top) and computation time (bottom) when applying the four methods to point clouds of different sizes sampled from 100 of bidegree $(3,3)$ (left), $(4,4)$ (middle) and $(5,5)$ (right), with added Gaussian noise of standard deviation 0.01.

the point cloud of size 8818, shown in Fig. 13 (left). In this case, we parameterize the input scattered data with the standard SP method as well as the proposed BIDGCN method and subsequently reconstruct the penalized least square adaptive spline model by performing the algorithm for bidegree $(2,2)$. Note that among all the listed values for the radius of the shape preserving method that we tested on this point cloud, the choice $r = \frac{3}{\sqrt{|\mathcal{P}|}}$ gives the best results in terms of polynomial approximation. To properly compare the results, the refinement tolerances are chosen such that they result in (almost) the same number of degrees of freedom in the final hierarchical spline model. When the SP method is considered, we obtain a THB-spline model with 5492 degrees of freedom, which is 62% of the total number of scattered input items, characterized by a mean squared error of 9.32e−4. If we parameterize the Nefertiti data with the BIDGCN method, the THB-spline approximation has 5228 degrees of freedom, i.e. 59% of the number of points, and

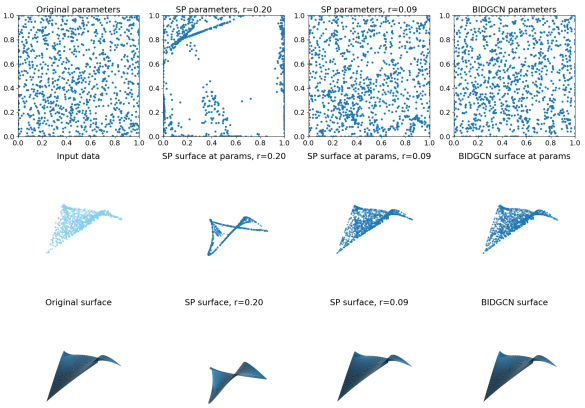

Fig. 10: Visual comparison of the BIDGCN and SP for radius $r = 0.2$ and $r = 0.09$ on a point cloud of size 1000 sampled from a biquadratic surface without noise.

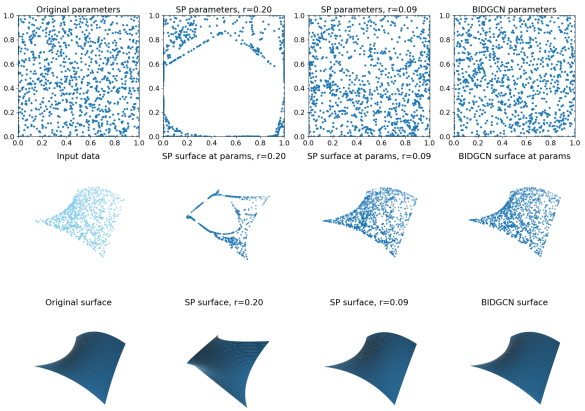

Fig. 11: Visual comparison of the BIDGCN and SP for radius $r = 0.2$ and $r = 0.09$ on a point cloud of size 1000 sampled from a biquadratic surface with Gaussian noise of standard deviation 0.01.

a corresponding mean squared error of 6.32e−4. The THB-spline approximations, scaled error distributions on the point cloud, parameter values, and hierarchical meshes obtained with SP and BIDGCN parameterizations are shown on the top and bottom of Fig. 14, respectively. We observe that BIDGCN leads to better results in terms of final model accuracy, since in average a smaller error is registered and the sharp features are better reproduced. In particular, close to the mouth of the model, the shape preserving parametric values tend to be clustered, a behaviour that clearly

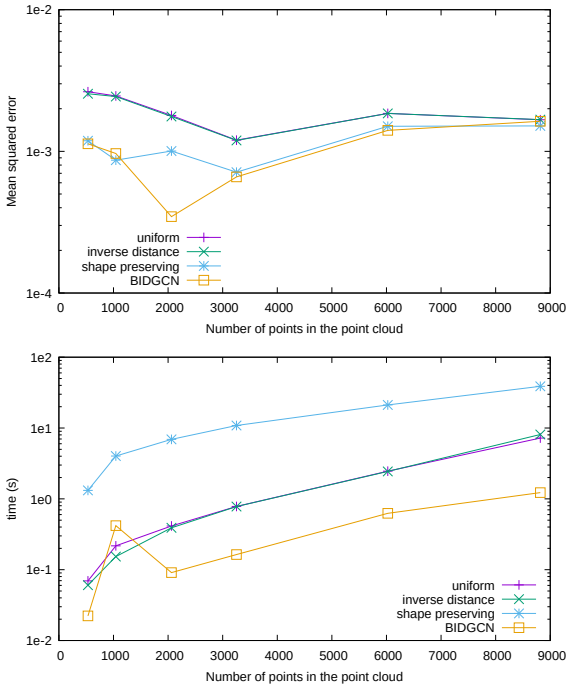

Fig. 12: Quantitative comparison of the computation time and mean squared error of the approximating biquadratic surface when parameterizing the point clouds of different size sampled from the Nefertiti bust model using BIDGCN and the standard methods.

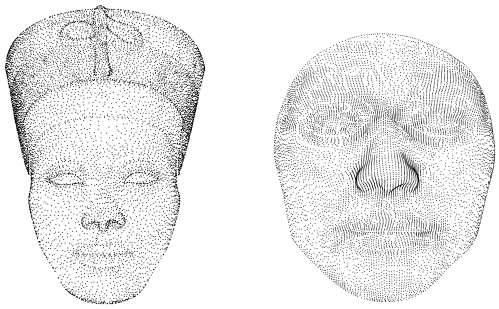

Fig. 13: Nefertiti (left) and face (right) point clouds characterized by 8818 and 9283 points, respectively.

affects the final quality of the adaptive spline approximation.

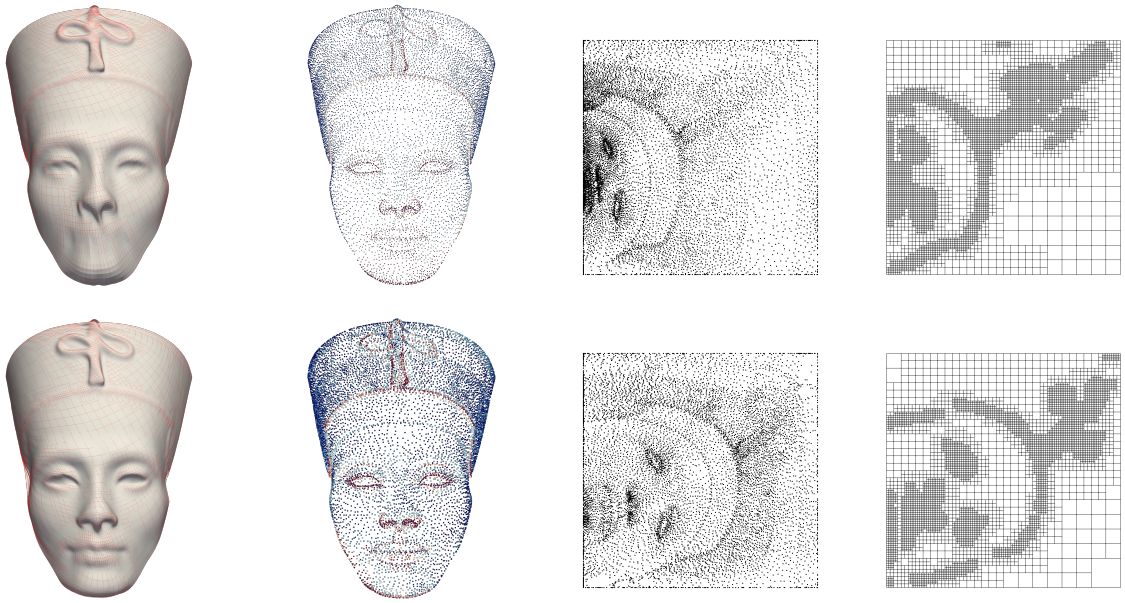

Fig. 14: Hierarchical spline model reconstruction of the Nefertiti point cloud shown on left of Fig. 13 for SP parameterization with 5492 degrees of freedom (top), as well al for the BIDGCN parameterization with 5288 degrees of freedom (bottom) with bidegree $(2, 2)$. The reconstructed THB-spline models are shown together with the scaled error distributions on the point cloud, the computed parametric values, and the corresponding hierarchical meshes (from left to right).

6.7 Hierarchical spline reconstruction of different degrees

In this last example we investigate the generalization capabilities of our BIDGCN parameterization to spline configurations of different bidegrees for the reconstruction of different point clouds. We start by performing an adaptive THB-spline fitting of a human face model of 9283 points, shown on the right of Fig. 13. In particular, we reconstruct the THB-spline models of the input point cloud by performing adaptive penalized least squares approximation for bidegree $(3, 3)$ and $(4, 4)$. In the first case, the final reconstructed geometry has 2687 degrees of freedom (29% of the number of points), that approximate the input point cloud registering a mean square error of 6.86e−5. As concerns the fitting for bidegree $(4, 4)$, the final reconstructed model is a THB-spline geometry with 2793 degrees of freedom (30% of the number of points) that approximates the input point cloud with a mean squared error of 6.08e−5. The reconstructed THB-spline models, their scaled error distributions on the point cloud, the computed parametric values, and the corresponding hierarchical meshes are illustrated in Fig. 14.

We end by considering two additional point clouds of 9636 items collected from a ship hull and a wind turbine. In particular, we approximate the two scattered data sets by performing adaptive THB-spline fitting for bidegree (5,2). The point clouds, the reconstructed THB-spline model, the point wise error distribution on the scattered data and the hierarchical meshes are displayed in Fig. 16 for both the ship hull (top) and the wind turbine (bottom). In addition, the final THB-spline model for the ship hull geometry has 4092 degrees of freedom (i.e. 42% of the number of points), and a mean square error equal to 3.38e−6, whereas the THB-spline model for the wind turbine has 2449 degrees of freedom, equivalent in magnitude to 25% of the number of input data, and a mean squared error of 5.61e−6.

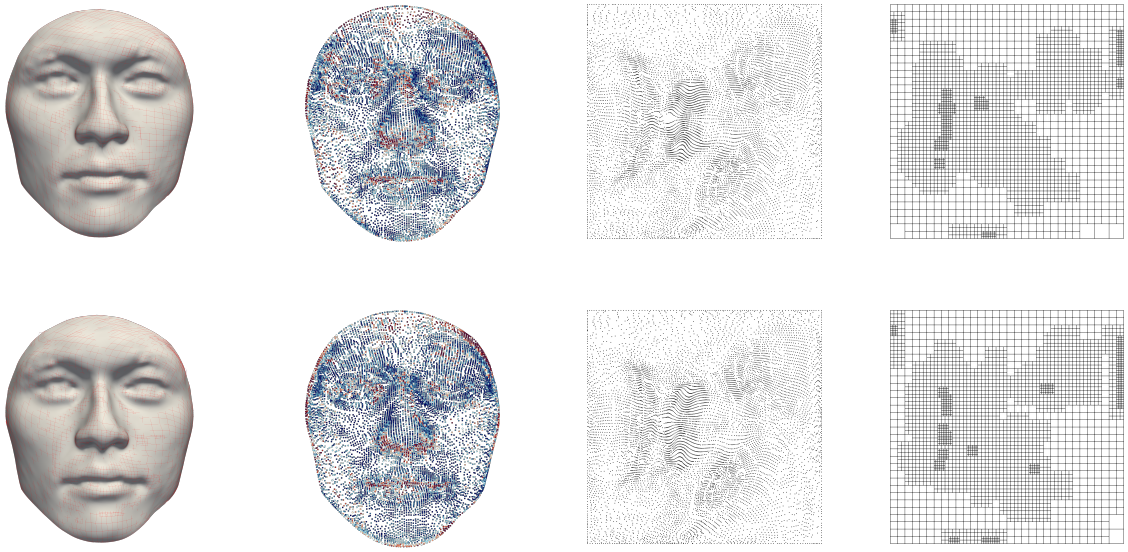

Fig. 15: Hierarchical spline model reconstruction of the face point cloud shown on the right of Fig. 13 for bidegree $(3, 3)$ (top) and $(4, 4)$ (bottom) using the BIDGCN parameterization. The reconstructed THB-spline models are shown together with the scaled error distributions on the point cloud, the computed parametric values, and the corresponding hierarchical meshes (from left to right).

Fig. 16: The point cloud, the hierarchical spline model reconstruction, the scaled error distribution and the hierarchical mesh for a ship hull (top) and a wind turbine (bottom) using the BIDGCN parameterization with bidegree $(5, 2)$.

7 Conclusion

We introduced the novel graph convolutional neural network *BIDGCN* for learning on point clouds.

In particular, we developed a new input layer which takes into account boundary conditions and propagates this information into the interior of the point cloud. Further hidden layers can then incorporate the boundary information into the prediction.

We applied this network to the problem of point cloud parameterization for surface approximation. When compared to standard parameterization methods, the architecture based on BIDGCN parameterization scheme achieves higher accuracy with less computational time, especially in the presence of noise in the data. In addition, BIDGCN is parameter independent and robust, avoiding failure issues in the case of sparse neighborhoods. Finally, we demonstrate the suitability of the network parameterization results for adaptive surface reconstruction with a new THB-spline fitting scheme with moving parameters. While we obtain valuable results on real-world data with BIDGCN trained on synthetic data, the performance of this new architecture can be highly optimized in real application settings by a suitable training on complex data sets in line with the specific test cases. This specialized training will be an important subject of our future work. Moreover, we plan to apply the new boundary informed graph convolutional layer to different computational tasks where boundary conditions play a role. A promising potential application of BIDGCN is the solution of partial differential equations on point cloud data. This could be done directly by adding Dirichlet or Neumann boundary conditions to the boundary vertices features and training a network to predict the solution of a specific PDE. Alternatively, a neural network based on BIDGCN can perform a supporting task for classical numerical methods for solving PDE, such as the mesh generation problem considered in [33], where adaptive mesh refinement for planar geometries was investigated.

Declarations

Funding

CG acknowledges the contribution of the National Recovery and Resilience Plan, Mission 4 Component 2 – Investment 1.4 – CN_00000013 "CENTRO NAZIONALE HPC, BIG DATA E QUANTUM COMPUTING", spoke 6. CG and SI are members of the INdAM group GNCS. The INdAM-GNCS support is gratefully acknowledged. CG, SI, AM also acknowledge the PHC GALILEE project 47786N and AM acknowledges H2020 Marie Sklodowska-Curie grant GRAPES No. 860843.

Competing interests

The authors have no relevant financial or non-financial interests to disclose.

Data availability

The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

References

[1] Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: Going beyond Euclidean data. IEEE Signal Processing Magazine **34**(4), 18–42 (2017)

[2] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y.: A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems **32**(1), 4–24 (2020)

[3] Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: International Conference on Machine Learning, pp. 1263–1272 (2017). PMLR

[4] Welling, M., Kipf, T.N.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR) (2017)

[5] Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 652–660 (2017)

[6] Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., Cohen-Or, D.: Meshcnn: a network with an edge. ACM Transactions on Graphics (TOG) **38**(4), 1–12 (2019)

[7] Sharp, N., Attaiki, S., Crane, K., Ovsjanikov, M.: DiffusionNet: Discretization agnostic learning on surfaces. ACM Transactions on Graphics (TOG) **41**(3), 1–16 (2022)

[8] Berrone, S., Della Santa, F., Mastropietro, A., Pieraccini, S., Vaccarino, F.: Graph-informed neural networks for regressions on graph-structured data. Mathematics **10**(5), 786 (2022)

[9] Fan, L., Ji, D., Lin, P.: Arbitrary surface data patching method based on geometric convolutional neural network. Neural Computing and Applications **35**(12), 8763–8774 (2023)

[10] Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph CNN for learning on point clouds. Acm Transactions On Graphics **38**(5), 1–12 (2019)

[11] Kiss, G., Giannelli, C., Zore, U., Jüttler, B., Großmann, D., Barner, J.: Adaptive CAD model (re-)construction with THB-splines. Graphical Models **76**(5), 273–288 (2014). Geometric Modeling and Processing 2014

[12] Bracco, C., Giannelli, C., Großmann, D., Sestini, A.: Adaptive fitting with THB-splines: Error analysis and industrial applications. Computer Aided Geometric Design **62**, 239–252 (2018)

[13] Bracco, C., Giannelli, C., Großmann, D., Imperatore, S., Mokriš, D., Sestini, A.: THB-Spline Approximations for Turbine Blade Design with Local B-Spline Approximations. In: Barrera, D., Remogna, S., Sbibih, D. (eds.) Mathematical and Computational Methods for Modelling, Approximation and Simulation, pp. 63–82. Springer, Cham (2022)

[14] Floater, M.S., Reimers, M.: Meshless parameterization and surface reconstruction. Computer Aided Geometric Design **18**(2), 77–92 (2001)

[15] Floater, M.S.: Parametrization and smooth approximation of surface triangulations. Computer aided geometric design **14**(3), 231–250 (1997)

[16] Yavuz, E., Yazici, R.: A dynamic neural network model for accelerating preliminary parameterization of 3D triangular mesh surfaces. Neural Computing and Applications **31**(8), 3691–3701 (2019)

[17] Giannelli, C., Imperatore, S., Mantzaflaris, A., Scholz, F.: Learning meshless parameterization with graph convolutional neural networks. In: Lecture Notes in Networks and Systems, World Conference on Smart Trends in Systems, Security and Sustainability. Springer, London (2023). to appear. https://inria.hal.science/hal-04142674

[18] Giannelli, C., Jüttler, B., Speleers, H.: THB-splines: The truncated basis for hierarchical splines. Computer Aided Geometric Design **29**(7), 485–498 (2012)

[19] Hoschek, J.: Intrinsic parametrization for approximation. Computer Aided Geometric Design **5**(1), 27–31 (1988)

[20] Wang, Y., Zheng, J.: Curvature-guided adaptive T-spline surface fitting. Computer-Aided Design **45**(8), 1095–1107 (2013)

[21] Shang, C., Fu, J., Feng, J., Lin, Z., Li, B.: Effective re-parameterization and GA based knot structure optimization for high quality t-spline surface fitting. Computer Methods in Applied Mechanics and Engineering **351**, 836–859 (2019)

[22] Sajavičius, S., Jüttler, B., Špeh, J.: In: Giannelli, C., Speleers, H. (eds.) Template Mapping Using Adaptive Splines and Optimization of the Parameterization. Springer INdAM Series, vol. 35, pp. 217–238. Springer, Cham (2019)

[23] Piegl, L., Tiller, W.: The NURBS Book (2nd Ed.). Springer, Berlin, Heidelberg (1997)

[24] Fang, J.-J., Hung, C.-L.: An improved parameterization method for B-spline curve and surface interpolation. Computer-Aided Design **45**(6), 1005–1028 (2013)

[25] Balta, C., Öztürk, S., Kuncan, M., Kandilli, I.: Dynamic centripetal parameterization method for b-spline curve interpolation. IEEE Access **8**, 589–598 (2020)

[26] Hoschek, J.: Intrinsic parametrization for approximation. Computer Aided Geometric Design **5**(1), 27–31 (1988)

[27] Saux, E., Daniel, M.: An improved Hoschek intrinsic parameterization. Computer Aided Geometric Design **20**, 513–521 (2003)

[28] Laube, P., Franz, M.O., Umlauf, G.: Deep Learning Parametrization for B-Spline Curve Approximation. In: 2018 International Conference on 3D Vision (3DV), pp. 691–699. IEEE Computer Society, Los Alamitos, CA, USA (2018)

[29] Scholz, F., Jüttler, B.: Parameterization for polynomial curve approximation via residual deep neural networks. Computer Aided Geometric Design **85**, 101977 (2021)

[30] Giannelli, C., Jüttler, B., Kleiss, S.K., Mantzaflaris, A., Simeon, B., Špeh, J.: THB-splines: An effective mathematical technology for adaptive refinement in geometric design and isogeometric analysis. Computer Methods in Applied Mechanics and Engineering **299**, 337–365 (2016)

[31] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. Curran Associates Inc., Red Hook, NY, USA (2019)

[32] Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)

[33] Chan, C.L., Scholz, F., Takacs, T.: Locally refined quad meshing for linear elasticity problems based on convolutional neural networks. Engineering with Computers **38**(5), 4631–4652 (2022)