



**HAL**  
open science

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

Carlotta Giannelli, Sofia Imperatore, Angelos Mantzaflaris, Felix Scholz

► **To cite this version:**

Carlotta Giannelli, Sofia Imperatore, Angelos Mantzaflaris, Felix Scholz. BIDGCN: Boundary informed dynamic graph convolutional network for adaptive spline fitting of scattered data. *Neural Computing and Applications*, In press, 10.1007/s00521-024-09997-0 . hal-04313629v2

**HAL Id: hal-04313629**

**<https://hal.science/hal-04313629v2>**

Submitted on 13 May 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

Carlotta Giannelli<sup>1</sup>, Sofia Imperatore<sup>1</sup>, Angelos Mantzaflaris<sup>2</sup>, Felix Scholz<sup>3\*</sup>

<sup>1</sup>Dipartimento di Matematica e Informatica, Università degli Studi di Firenze, Italy.

<sup>2</sup>Centre Inria at Université Côte d’Azur, Sophia Antipolis, France.

<sup>3\*</sup>Institute of Applied Geometry, Johannes Kepler University Linz, Austria.

\*Corresponding author(s). E-mail(s): [felix.scholz@jku.at](mailto:felix.scholz@jku.at);

Contributing authors: [carlotta.giannelli@unifi.it](mailto:carlotta.giannelli@unifi.it); [sofia.imperatore@unifi.it](mailto:sofia.imperatore@unifi.it);  
[angelos.mantzaflaris@inria.fr](mailto:angelos.mantzaflaris@inria.fr);

## Abstract

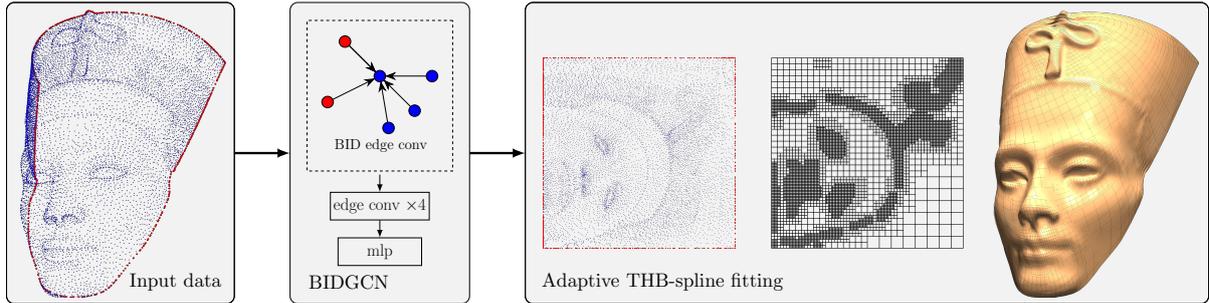
Surface reconstruction from scattered point clouds is the process of generating surfaces from unstructured data configurations retrieved using an acquisition device such as a laser scanner. Smooth surfaces are possible with the use of spline representations, an established mathematical tool in Computer-Aided Design and related application areas. One key step in the surface reconstruction process is the parameterization of the points, that is, the construction of a proper mapping of the 3D point cloud to a planar domain that preserves surface boundary and interior points. Despite achieving a remarkable progress, existing heuristics for generating a suitable parameterization face challenges related to the accuracy, the robustness with respect to noise, and the computational efficiency of the results. 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. A selection of numerical examples shows the effectiveness of the proposed approach for adaptive spline fitting with (truncated) hierarchical B-spline constructions. In our experiments, improved accuracy is obtained, e.g., from 60% up to 80% for noisy data, while speedups ranging from 4 up to 180 times are observed with respect to classical algorithms. Moreover, our method automatically predicts the local neighborhood graph, leading to much more robust results without the need for delicate free parameter selection.

**Keywords:** geometric deep learning, parameterization, surface fitting, adaptive spline approximation, THB-splines

## 1 Introduction

In this paper we use geometric deep learning techniques and spline fitting methods to obtain a

new *Boundary Informed Dynamic Graph Convolutional Network (BIDGCN)* for parameterization and spline surface approximation of scattered point clouds. More precisely, we consider the problem of generating surfaces from unstructured point



**Fig. 1:** BIDGCN point cloud parameterization and adaptive THB-spline fitting. From left to right: an input scattered point cloud with interior (blue) and boundary (red) points is processed by the proposed BIDGCN network to predict the input point parameterization. The predicted parameters are subsequently employed in an adaptive surface reconstruction scheme to get the final hierarchical geometric model.

data retrieved using acquisition devices such as laser scanners. First, we train and use BIDGCN to obtain a parameterization of the points, that is, the definition of a proper mapping of the 3D point cloud to a planar domain, which preserves surface boundary and interior points. Second, we compute smooth surface representations of the data using adaptive spline models that generalize the established tensor-product splines used in Computer-Aided Design. The workflow of the proposed method is summarized in Fig. 1.

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 [1–4]. A class of methods for scattered point cloud parameterization was proposed in [5] building upon the methods from [6] for the related problem of parameterizing triangulated surface data. In these methods, the one-dimensional boundary of the point cloud is parameterized separately to guarantee that it is properly mapped to the boundary of the planar domain. 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 [5]. Recently, machine learning tools have been applied to the problem of determining these coefficients [7, 8]. In particular, [7] builds upon [6] and trains a network for a fixed triangular mesh, whereas [8] provides a more general meshless data-driven approach based on [5]. 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.

If data are arranged on a regular grid in a Euclidean space, they can be processed by feed-forward convolutional neural networks. In these models the information flows from the input, through the intermediate convolutions used to define the architecture, and finally to the output. There are no feed-back connections in which the outputs of the model are fed back into itself. Geometric deep learning refers to the generalization of convolutional neural networks to non-Euclidean data [9]. These include discrete manifolds, i.e., a topological spaces that are locally Euclidean [10], graphs, i.e., finite sets of points together with a prescribed set of connections (edges) [11], and general point clouds, i.e. discrete sets of points in space. An example of a scattered point cloud is shown in Fig. 1 (left). Its boundary points are highlighted in red and its interior points in blue.

In the case of unstructured non-Euclidean data several different operators that mimic the behavior of standard convolutional neural networks are possible [12]. Most current graph neural

network architectures are classified as *message passing neural networks* [13], 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 (edge-connected vertices). In [14] an edge convolution operator is presented, where 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.

While graph neural networks have been successfully applied to classification tasks such as shape recognition, segmentation, and registration [15–18], regression tasks where the network should predict a (potentially vector-valued) function on the input geometry are much less studied [19, 20]. 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 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.

The key idea of the Boundary Informed Dynamic Graph Convolutional Network proposed in this paper 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) and, (ii) the dynamic prediction of the graph used for message passing (due to the dynamic edge convolution approach as in [14]).

We use BIDGCN to obtain boundary-fitted parameterizations of point clouds, and we consequently apply adaptive fitting with truncated hierarchical B-splines [21] to reconstruct smooth surfaces. The performance of the adaptive approximation scheme is enhanced by introducing a parameter correction step [22] within each iteration of the adaptive strategy, see also [4, 23]. This extends the case of *structured* data fitting with T-splines studied in [24, 25]. Our main contributions to the problem of parameterizing and fitting scattered point clouds are summarized as follows:

1. *Computational efficiency:* After training the network its evaluation is computationally much more efficient than applying the classical methods in [5] and the learning-based method in [8]. 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., [5, 8], 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. 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 neighborhoods.

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 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 [14]. We first briefly describe the original dynamic edge convolution operator in Section 2.1. Then, we present our new boundary informed dynamic convolutional layer and the additional layers of the network architecture in Section 2.2.

### 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 [14] 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

$$\|x_i - x_j\| \quad (1)$$

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

$$e_{ji} = h_{\Theta}(x_i, x_j - x_i), \quad (2)$$

is evaluated, where  $h_{\Theta}$  is a feed-forward neural network

$$h_{\Theta} : \mathbb{R}^{2n} \rightarrow \mathbb{R}^m \quad (3)$$

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

$$x'_i = \square_{(j,i) \in \mathcal{G}_k} e_{ji}, \quad (4)$$

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. This implies that information can travel arbitrarily far in each layer, as it is determined by the training process. In [14] 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 [14] the  $k$  nearest-neighbor graph was used, we propose to use the radius graph also in the hidden layers. In particular, 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

$$\|x_i - x_j\| \leq r. \quad (5)$$

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. 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. We refer to the graph neural networks characterized by dynamic edge convolution operators as DGCNN (Dynamic Graph Convolutional Neural Networks).

## 2.2 Boundary informed dynamic edge convolution

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  $\mathcal{P} = \mathcal{P}_B \cup \mathcal{P}_I$  into boundary and interior points. Moreover, we assume that each vertex  $i \in \mathcal{P}_I$  comes with features  $x_i \in \mathbb{R}^n$  and every vertex  $j \in \mathcal{P}_B$  comes with features  $(x_j, u_j) \in \mathbb{R}^n \times \mathbb{R}^d$ , where we regard  $u_j \in \mathbb{R}^d$  as boundary conditions.

We first compute two different radius graphs

$$\mathcal{G}_{I \rightarrow I} \quad \text{and} \quad \mathcal{G}_{B \rightarrow I}, \quad (6)$$

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

$$(j, i) \quad \text{with} \quad i, j \in \mathcal{P}_I : \|x_i - x_j\| \leq r, \quad (7)$$

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

$$(k, i) \quad \text{with} \quad i \in \mathcal{P}_I, k \in \mathcal{P}_B : \|x_i - x_k\| \leq r. \quad (8)$$

This means that even in  $\mathcal{G}_{B \rightarrow I}$ , the edges do not depend on the boundary conditions  $u_j$ . 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  $(j, i) \in \mathcal{G}_{I \rightarrow I}$ , we compute

$$e_{ji} = h_{\Theta}(x_i, x_j - x_i), \quad (9)$$

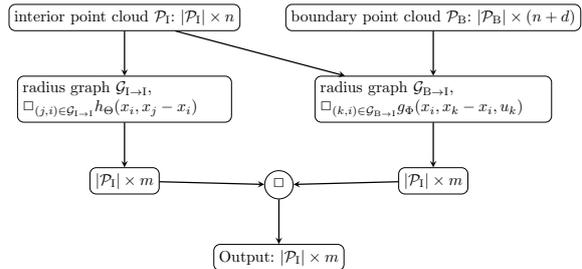


Fig. 2: Boundary informed input layer.

where  $h$  is a feed-forward neural network

$$h_{\Theta} : \mathbb{R}^{2n} \rightarrow \mathbb{R}^m \quad (10)$$

with output feature size  $m$  and learnable weights  $\Theta$ .

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

$$e_{ik} = g_{\Phi}(x_i, x_k - x_i, u_k), \quad (11)$$

where  $g$  is another feed-forward neural network

$$g_{\Phi} : \mathbb{R}^{2n+d} \rightarrow \mathbb{R}^m \quad (12)$$

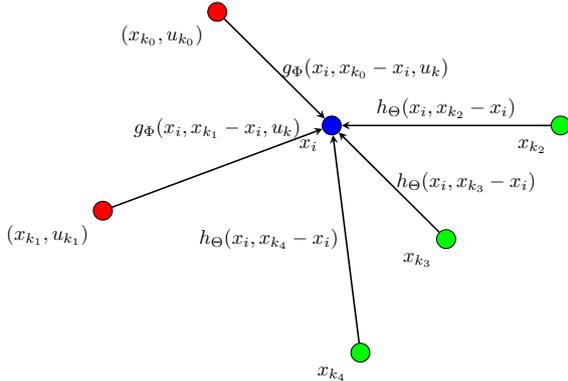
with the same output feature size and independent learnable weights  $\Phi$ .

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  $i \in \mathcal{P}_I$ , we have

$$x'_i = (\square_{j:(j,i) \in \mathcal{G}_{I \rightarrow I}} e_{ji}) \square (\square_{k:(k,i) \in \mathcal{G}_{B \rightarrow I}} e_{ki}), \quad (13)$$

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  $\mathcal{P}_I$ . During training  $\Theta$  and  $\Phi$  are optimized simultaneously. The flow of the input layer is depicted in Fig. 2 and its application to an interior vertex is illustrated in Fig. 3.

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 neighbors in  $\mathcal{G}_{B \rightarrow I}$



**Fig. 3:** Edge contributions of the input layer for  $x_i$  (blue) computed from the features at the neighboring boundary vertices (red) and interior vertices (green).

is propagated further into the interior of the point cloud.

### 3 Scattered point cloud parameterization for (THB-spline) surface fitting

In this section, we introduce the problem of scattered point cloud parameterization for approximation with smooth surfaces. We summarize standard parameterization methods of scattered point clouds and we introduce tensor-product B-splines and their generalization to truncated hierarchical (TH) B-splines.

#### 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

$$u : \mathcal{P} \rightarrow \Omega, \quad (14)$$

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 frameworks. In particular, we consider the task of approximating a point cloud sampled from a surface with a smooth parametric surface defined on  $\Omega = [0, 1]^2$ :

$$S : [0, 1]^2 \rightarrow \mathbb{R}^3 \quad (15)$$

with

$$S(u) := \sum_{j=0}^N c_j B_j(u), \text{ for } u \in [0, 1]^2 \quad (16)$$

defined in the spline space spanned by the basis  $\{B_0, \dots, B_N\}$ , with  $B_j : [0, 1]^2 \rightarrow \mathbb{R}$  for  $j = 0, \dots, N$ . In this paper, we consider either the tensor-product Bernstein polynomials [26, 27] or truncated hierarchical B-splines (THB-splines) [21] as basis functions  $B_j$ .

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

$$\min_{\substack{c_0, \dots, c_N \in \mathbb{R}^3, \\ u_1, \dots, u_{|\mathcal{P}|} \in [0, 1]^2}} \sum_{i=1}^{|\mathcal{P}|} \|S(u_i) - x_i\|^2. \quad (17)$$

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

$$\min_{c_0, \dots, c_N \in \mathbb{R}^3} \sum_{i=1}^{|\mathcal{P}|} \left\| \sum_{j=0}^N c_j B_j(u_i) - x_i \right\|^2 \quad (18)$$

using standard techniques from computational linear algebra. Our goal is find the parameterization  $u$  that minimizes the residual of (18).

#### 3.2 Standard and learning-based parameterization methods for scattered data

In order to demonstrate the efficiency and accuracy of the proposed method, we will compare its performance with the three parameterization methods presented in [5], i.e., the *uniform*, the *inverse distance*, and the *shape preserving* (SP) parameterizations, as well as the learning-based parameterization method proposed in [8], denoted as PARGCN. In all these methods the parameterization of the boundary of the point cloud is first determined separately. This can be done using

standard techniques such as uniform, chord length and centripetal parameterization [28] and their improvements [29, 30], using iterative schemes [31, 32] as well as using feed-forward neural networks [33, 34].

Then, a graph of the input point cloud  $\mathcal{P} = \mathcal{P}_I \cup \mathcal{P}_B$  is constructed by choosing neighborhoods  $N_i \subset \mathcal{P}$  for all  $i \in \mathcal{P}_I$ . 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

$$u_i = \sum_{k \in N_i} \lambda_{ik} u_k, \quad (19)$$

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

For the uniform and the inverse distance parameterizations, the neighborhoods  $N_i$  are chosen as

$$N_i = \{k \in \mathcal{P} : \|x_k - x_i\| \leq r\} \quad (20)$$

for some radius  $r$  and the weights are set to

$$\lambda_{ik} = \frac{1}{|N_i|} \quad (\text{uniform}) \quad (21)$$

or to

$$\lambda_{ik} = \frac{\frac{1}{\|x_k - x_i\|}}{\sum_{\ell \in N_i} \frac{1}{\|x_\ell - x_i\|}} \quad (\text{inverse distance}). \quad (22)$$

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  $i \in \mathcal{P}$  in the Delaunay triangulation are its neighbors in the *local projection* graph. The corresponding weights are found based on the shape preserving weights for triangulated surfaces presented in [6]. For each interior point  $i$ , the neighbors are ordered counter-clockwise and intermediate local parameters  $\tilde{u}$  are determined such that for all  $j \in N_i$

$$\|\tilde{u}_i - \tilde{u}_j\| = \|x_i - x_j\| \quad (23)$$

and

$$\angle(\tilde{u}_j, \tilde{u}_i, \tilde{u}_{j+1}) = \frac{2\pi \angle(x_j, x_i, x_{j+1})}{\sum_{k, (k+1) \in N_i} \angle(x_k, x_i, x_{k+1})}, \quad (24)$$

i.e. the lengths and the angle ratios around  $x_i$  are preserved. Finally, corresponding weights  $\lambda_{ij}$  are computed from the parameters  $\tilde{u}_j$ .

For the PARGCN parameterization, the neighborhoods  $N_i$  are radius-neighborhoods as in (20). Their collection yields a radius graph embedding the proximity information to each 3D point in the point cloud. This graph is subsequently converted to its line graph [35], which encodes point distances and attaches corresponding parameterization weights to each edge. The line graph is used as input to a geometry-informed graph convolutional neural network trained to predict optimal parameterization weights. By solving the linear system in (19) with the weights predicted by the network, the parameter values are obtained. As a further processing step, their Delaunay triangulation [36] is computed and for each interior point  $i \in \mathcal{P}_I$ , the shape preserving weights  $\lambda_{ij}$  with respect to the corresponding neighbors  $j \in \mathcal{P}$  in the planar triangulation are determined. The PARGCN parameterization is obtained by solving the linear system (19) once more based on the neighborhoods defined by the Delaunay triangulation and the shape preserving weights.

### 3.3 (TH)B-spline constructions

The basis  $\{B_0, \dots, B_N\}$  used to define the parametric surface  $S : [0, 1]^2 \rightarrow \mathbb{R}^3$  as in (16), is fundamental, since it significantly influences the geometrical and numerical features of the approximant  $S$ . Because of their beneficial properties and the simplicity of their construction, spline have been largely adopted for geometric modelling applications, see e.g., [27, 28].

#### 3.3.1 B-splines

The standard Computer-Aided Design representation for univariate splines are B-splines. More precisely, let  $[0, 1] \subset \mathbb{R}$  be the considered parameter domain, let  $d \in \mathbb{N}$  be a polynomial degree,  $k := d + 1$  the corresponding order, and let  $\mathbf{t} := [t_0, \dots, t_{N+k}]$  be a vector of non-decreasing real values, with  $t_{k-1} \equiv 0$  and  $t_{N+1} \equiv 1$ , called *knot-vector*. Moreover, indicate with  $1 \leq \mu_j \leq d$  the

number of times the knot value  $t_j$  appears in  $\mathbf{t}$ , for  $j = 0, \dots, N + k$ . The  $N + 1$  univariate B-splines of degree  $d$  over  $[0, 1]$  can be recursively generated [37]. To this purpose, let  $\beta_{j,1} : \mathbb{R} \rightarrow \mathbb{R}$  indicate the  $j$ -th B-spline of order 1, for each  $j = 0, \dots, N$ . More specifically, for  $u \in \mathbb{R}$ ,

$$\beta_{j,1}(u) := \begin{cases} 1 & \text{if } u \in [t_j, t_{j+1}) \\ 0 & \text{otherwise} \end{cases} \quad (25)$$

and  $\beta_{j,1}(t_{N+1}) = 1$ . Subsequently, let  $\beta_{j,r} : \mathbb{R} \rightarrow \mathbb{R}$  indicate the  $j$ -th B-spline of order  $2 \leq r \leq k$ , then

$$\begin{aligned} \beta_{j,r}(u) &= \omega_{j,r}(u)\beta_{j,r-1}(u) \\ &+ (1 - \omega_{j+1,r}(u))\beta_{j+1,r-1}(u), \end{aligned} \quad (26)$$

where  $\omega_{j,r} : \mathbb{R} \rightarrow \mathbb{R}$  is a piecewise *linear* polynomial of the form

$$\omega_{j,r}(u) = \begin{cases} \frac{u-t_j}{t_{j+r-1}-t_j}, & \text{if } u < t_{j+r-1} \\ 0 & \text{otherwise.} \end{cases} \quad (27)$$

For  $r = k$ , the maximum desired polynomial order, it follows that  $B_j = \beta_{j,k}$ . In particular, each B-spline  $B_j$  for  $j = 0, \dots, N$  is a piecewise polynomial function of degree  $d$  that has maximum local smoothness on each subinterval of the partition  $\mathbf{t}$ , whereas the regularity at each unique knot  $t_j \in \mathbf{t}$  is  $d - \mu_j$ . Univariate B-splines are characterized by three key properties: (1) non-negativity,  $B_j \geq 0$  for all  $u \in \mathbb{R}$ , (2) local support,  $B_j(u) = 0$  if  $u \notin [t_j, t_{j+k})$ , and (3) partition of unity,  $\sum_{j=0}^N B_j(u) = 1$  if  $u \in [t_{k-1}, t_{n+1}] \equiv [0, 1]$ . The *spline space* of order  $k$  and knots  $\mathbf{t}$  is generated by  $\{B_0, \dots, B_N\}$ , namely

$$V = \text{span}\{B_0, \dots, B_N\}. \quad (28)$$

Finally, note that with an appropriate choice of the knot-vector  $\mathbf{t}$ , as

$$\mathbf{t} = \underbrace{[0, \dots, 0]}_k, \underbrace{[1, \dots, 1]}_k, \quad (29)$$

Bernstein polynomials [26, 27] can be recovered from B-spline constructions.

### 3.3.2 Tensor-product B-splines

A straightforward extension of univariate B-splines to higher dimensions can be achieved by

considering a tensor-product construction. Within the bivariate framework, let  $\Omega = [0, 1]^2$  be the unit-square domain of  $\mathbb{R}^2$ ,  $\mathbf{d} = (d_1, d_2)$  the polynomial bidegree,  $\mathbf{k} = (k_1, k_2)$  the corresponding polynomial biorder and  $G$  the tensor-product mesh, defined by suitable knot vectors in each parametric direction  $\mathbf{t}_1, \mathbf{t}_2$ . A bivariate tensor-product B-spline  $B_j : \mathbb{R}^2 \rightarrow \mathbb{R}$  is then defined as the product of 2 univariate B-splines, for each  $j \in \Gamma_{\mathbf{k}} := \{j = (j_1, j_2) \mid j_h = 1, \dots, n_h, h = 1, 2\}$ . An example of a tensor product mesh, a tensor-product basis with bidegree  $(3, 2)$  and two tensor-product B-spline basis functions is depicted in Fig. 4. Similarly to the univariate case, note that by choosing  $\mathbf{t}_1$  and  $\mathbf{t}_2$  as in (29), the tensor-product Bernstein polynomials can be recovered from tensor-product B-spline constructions.

Because of the tensor construction, many of the simple algebraic properties of the univariate B-splines hold. In particular, non-negativity, locality, and partition of unity are preserved. The *tensor-product spline space* of biorder  $\mathbf{k}$  with respect to the tensor-product grid  $G$  is the space

$$V := \text{span}\{B_j \mid j \in \Gamma_{\mathbf{k}}\}. \quad (30)$$

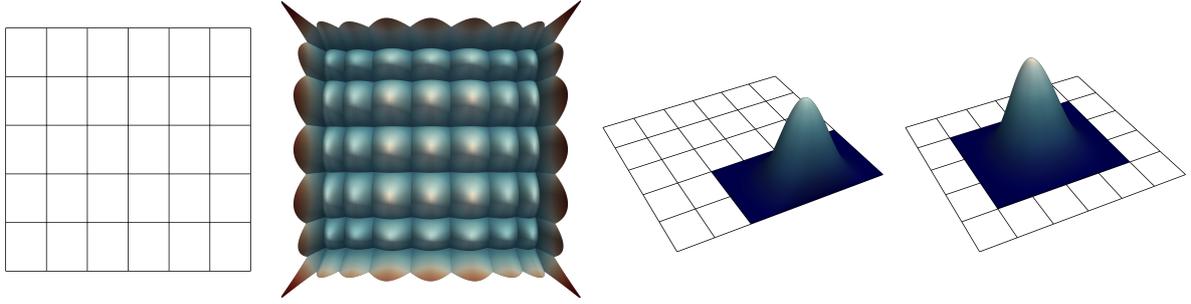
### 3.3.3 (Truncated) Hierarchical B-splines

Hierarchical B-splines provide a natural strategy to overcome the limitation imposed by standard tensor-product B-spline constructions. In particular, hierarchical B-splines [38] allow to develop adaptive and locally refined spline models.

Let  $\Omega = [0, 1]^2$  and let  $\mathbf{d} = (d_1, d_2)$  be a polynomial bidegree, while  $\mathbf{k} = (d_1 + 1, d_2 + 1)$  is the corresponding biorder. Consider a sequence of *nested* linear vector *spline* spaces defined by  $V^0 \subset \dots \subset V^\ell \subset V^{\ell+1} \subset V^{L-1}$ , where the index  $\ell$  of  $V^\ell$  is called its level. We assume that

$$V^\ell = \text{span}\{B_0^\ell, \dots, B_{N_\ell}^\ell\} = \text{span}\{B_j^\ell \mid j \in \Gamma_{\mathbf{k}}^\ell\} \quad (31)$$

has finite dimension  $N_\ell = \dim V^\ell$  where the B-spline basis  $\{B_0^\ell, \dots, B_{N_\ell}^\ell\}$  is defined by the polynomial biorder  $\mathbf{k}$  and spline basis functions  $B_j^\ell : \Omega \rightarrow \mathbb{R}$ . In addition, for each level  $\ell$ , we assume that the boundaries of the supports of all basis functions  $B_j^\ell$ , for  $j = 0, \dots, N_\ell$ , partition the



**Fig. 4:** From left to right: bivariate tensor-product mesh; the corresponding bivariate tensor-product basis with bidegree  $(3, 2)$ ; two examples of basis functions.

domain  $\Omega$  into a certain number of connected *cells* of level  $\ell$ . Furthermore, let  $\Omega \equiv \Omega^0 \supset \dots \supset \Omega^\ell \supset \Omega^{\ell+1} \supset \dots \supset \Omega^L \equiv \emptyset$  be a sequence of nested open subdomains, where each  $\Omega^\ell$  represents a local region of  $\Omega = [0, 1]^2$ . In particular, we assume that the closure of each subdomain  $\Omega^\ell$  coincides with the closure of a collection of cells of level  $\ell$ . The hierarchical spline basis is defined as

$$\mathcal{H}_{\mathbf{k}} := \{B_j^\ell \mid j \in A_{\mathbf{k}}^\ell, \ell = 0, \dots, L-1\} \quad (32)$$

with the indices

$$A_{\mathbf{k}}^\ell := \{j \in \Gamma_{\mathbf{k}}^\ell \mid \text{supp}(B_j^\ell) \subseteq \Omega^\ell \wedge \text{supp}(B_j^\ell) \not\subseteq \Omega^{\ell+1}\} \quad (33)$$

where  $\text{supp}(B_j^\ell)$  denotes the intersection of the support of  $B_j^\ell$  with  $\Omega^0$ . The corresponding hierarchical space is defined as  $V = \text{span}\{\mathcal{H}_{\mathbf{k}}\}$ . An example of univariate hierarchical B-spline basis with degree  $d = 2$ , constructed over 3 levels is illustrated in Fig. 5 (left). The coarser to finer level hierarchy is represented from top to bottom and the subdomain  $\Omega^1 \supset \Omega^2$  are highlighted in red. More precisely, for  $\ell = 0, 1, 2$  any spline basis function of level  $\ell$  whose support is completely contained in  $\Omega^{\ell+1}$  is replaced by splines at successively hierarchical levels.

This simple definition, however, leads to different overlaps of coarse and fine B-spline supports, and partition of unity of the basis is lost. Truncated hierarchical B-splines (THB-splines) were introduced in [21] to reduce the interaction between hierarchical functions at different levels and recover the partition of unity property. For any  $\ell = 0, \dots, L-2$ , a spline  $s \in V^\ell \subset V^{\ell+1}$  can

be represented in terms of the basis of the refined space  $V^{\ell+1}$  as

$$s(u) = \sum_{j \in \Gamma_{\mathbf{k}}^{\ell+1}} c_j^{\ell+1}(s) B_j^{\ell+1}(u), \quad \text{for } u \in \Omega, \quad (34)$$

for suitable coefficients  $c_j^{\ell+1}(s)$ , for each  $j \in \Gamma_{\mathbf{k}}^{\ell+1}$ . The truncation of  $s \in V^\ell$  at level  $\ell+1$  is defined as

$$\text{trunc}^{\ell+1}(s) := \sum_{\substack{j \in \Gamma_{\mathbf{k}}^{\ell+1} \\ \text{supp}(B_j^\ell) \not\subseteq \Omega^{\ell+1}}} c_j^{\ell+1}(s) B_j^{\ell+1} \quad (35)$$

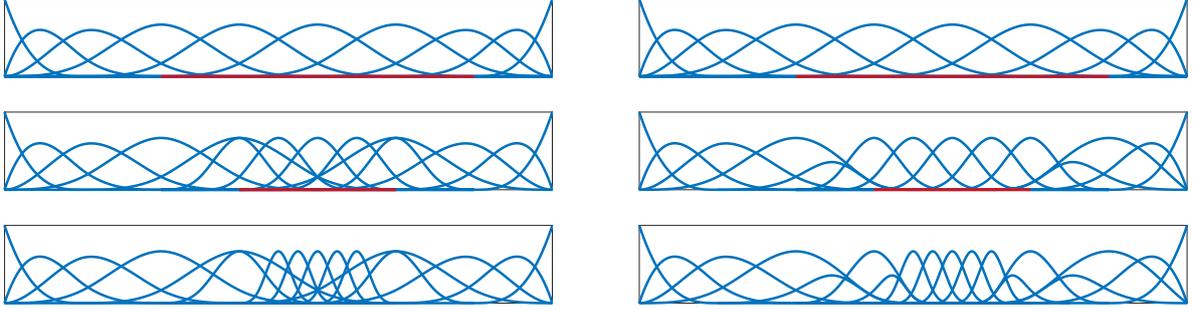
and the cumulative truncation with respect to all finer levels is

$$\text{Trunc}^{\ell+1}(s) := \text{trunc}^{L-1}(\text{trunc}^{L-2}(\dots (\text{trunc}^{\ell+1}(s)) \dots)), \quad (36)$$

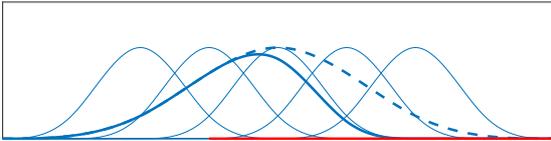
with  $\text{Trunc}^L(s) \equiv s$ , for  $s \in V^{L-1}$ . An example of the truncation mechanism over one level of refinement is represented in Fig. 6. Finally, the THB-spline basis for the hierarchical space can be defined as

$$\mathcal{T}_{\mathbf{k}} := \{T_j^\ell = \text{Trunc}^{\ell+1}(B_j^\ell) \mid j \in A_{\mathbf{k}}^\ell, \ell = 0, \dots, L-1\}, \quad (37)$$

where the B-spline  $B_j^\ell$  is the *mother* B-spline of the truncated  $T_j^\ell$ . THB-splines (1) are non-negative, (2) have local support, (3) form a partition of unity, and span the same space of the hierarchical B-spline basis,  $V = \text{span}\{\mathcal{H}_{\mathbf{k}}\} \equiv \text{span}\{\mathcal{T}_{\mathbf{k}}\}$  [21]. Fig. 5 (right) shows an example of univariate truncated hierarchical B-spline basis



**Fig. 5:** Univariate HB- (left) and THB-splines of degree 2 over three hierarchical levels (from top to bottom). The refinement region of interest at each level is highlighted in red.



**Fig. 6:** Example of truncation between two consecutive hierarchical levels for univariate THB-spline. The domain of interest  $\Omega^{\ell+1}$  is highlighted in red; the B-splines of level  $\ell + 1$  are depicted as thin blue lines; the THB-spline is represented with a thick blue line and its mother B-spline of level  $\ell$  is illustrated with a blue dashed line.

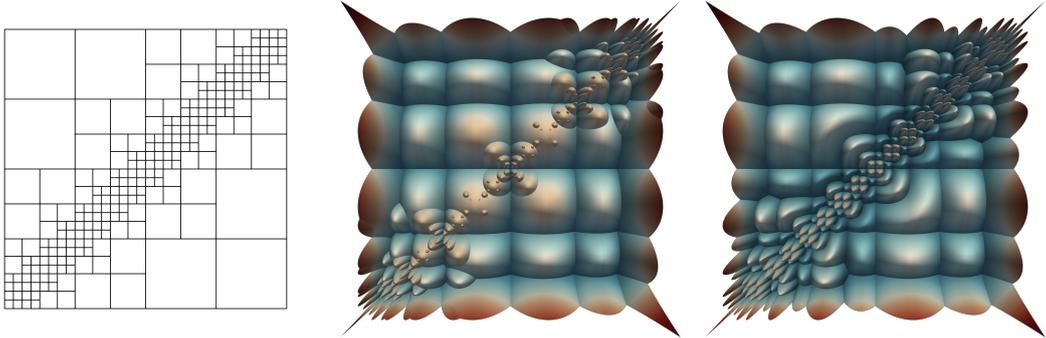
with degree  $d = 2$ , constructed over 3 levels. In particular, it consists of the truncated counterpart of the hierarchical basis shown on the left part of the same figure. As concerns the bivariate case, Fig. 7 shows a hierarchical mesh characterized by local refinement along its diagonal and the corresponding HB- and THB-spline bivariate basis of bidegree  $(2, 2)$ . Because of their properties, THB-splines are a desirable tool for building flexible geometric models. The effectiveness of employing THB-splines both for geometric design and isogeometric analysis has been shown in [39], among others.

THB-splines have been here presented in their bivariate form on  $[0, 1]^2$ , yet their construction can be developed in any parametric and physical dimension, namely for any  $\Omega \subseteq \mathbb{R}^D$ , and  $D \geq 2$ .

## 4 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  $u_i \in [0, 1]^2 \subset \mathbb{R}^2$  for the given point cloud such that the resulting solution  $c_0, \dots, c_N \in \mathbb{R}^3$  of (18) 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 (18). Without fixing some of the parameter values a priori, this operator is not uniquely defined, making it difficult to directly train neural networks for this task. Inspired by the standard methods summarized in Section 3.2, we fix the parameters  $u_i$  for the boundary points using 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 this setting.



**Fig. 7:** Hierarchical mesh (left) and the corresponding bivariate HB- (center) and THB-spline (right) basis of bidegree  $(2, 2)$ .

#### 4.1 Network architecture

To tackle the point cloud parameterization problem, we design a neural network based on the new boundary informed input layer described in Section 2.

The architecture of our proposed network is shown in Fig. 8. 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  $\mathcal{G}_{I \rightarrow I}$  and one for the edges  $\mathcal{G}_{B \rightarrow I}$ . For  $\mathcal{G}_{I \rightarrow I}$  we train an MLP with layer sizes  $\{6, 64, 64\}$ , while for  $\mathcal{G}_{B \rightarrow I}$  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. 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.

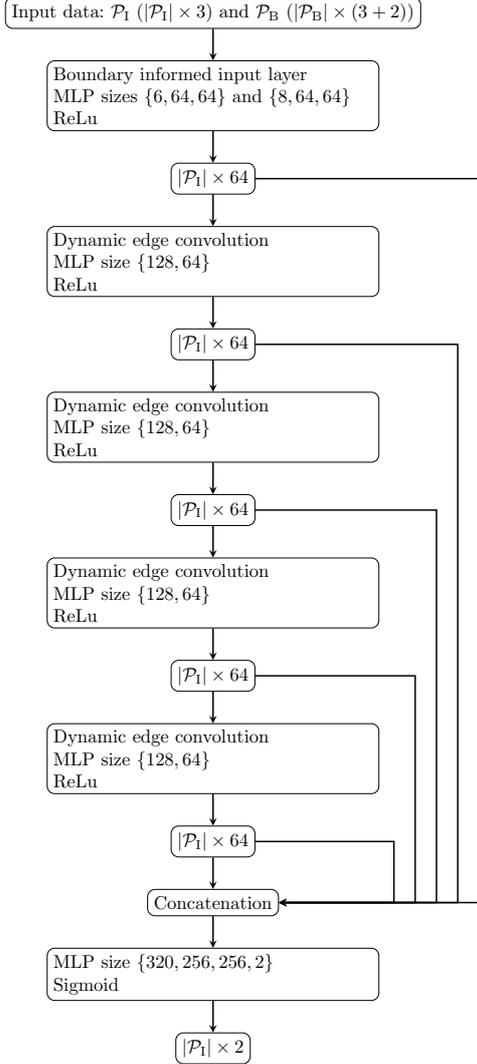
#### 4.2 Data generation

In our training, we employ tensor-product Bézier surfaces, meaning that the basis  $B_j(u) = B_{ab}(u_1, u_2)$ , for  $j(ab) = 0, \dots, N$ , that we use for fitting a surface is the tensor-product B-spline basis with knot vectors chosen as (29) for both directions. 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  $\mathcal{P} = \mathcal{P}_I \cup \mathcal{P}_B$  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. The point clouds are sampled from biquadratic tensor-product Bézier surfaces whose control points  $c_{j(a,b)}$  were randomly sampled from

$$\mathcal{C}_{a,b} = \left[ \frac{a}{2} - \frac{1}{4}, \frac{a}{2} + \frac{1}{4} \right] \times \left[ \frac{b}{2} - \frac{1}{4}, \frac{b}{2} + \frac{1}{4} \right] \times [-1, 1] \quad (38)$$

for  $a, b = 0 \dots 2$  according to the uniform distribution. This choice of sampling space ensures a



**Fig. 8:** Proposed network architecture of the point cloud parameterizing network based on BIDGCN.

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 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

---

**Algorithm 1:** Generation of a Bézier scattered point cloud  $\mathcal{P}$ , with interior points  $P_I$ , boundary points  $\mathcal{P}_B$ , and boundary parameters  $U_B$ .

---

**Input:** Polynomial bidegree  $(d, d)$ ,  
number of interior points  $n$ ,  
number of boundary points  $b$ .

1 Initialize  $U_B \leftarrow$  array of dimension  $b \times 2$ ,  
 $P \leftarrow$  array of dimension  $(n + b) \times 3$ ,  $C \leftarrow$   
array of dimension  $(d + 1) \times (d + 1) \times 3$ .

2 Sample  $\alpha, \beta, \gamma \leftarrow \text{UNIFORM}(0, 2\pi)$

**for**  $i = 0, \dots, d$  **do**

3 **for**  $j = 0, \dots, d$  **do**

4  $x \leftarrow \text{UNIFORM}(\frac{j}{d} - \frac{1}{2d}, \frac{j}{d} + \frac{1}{2d})$

$y \leftarrow \text{UNIFORM}(\frac{j}{d} - \frac{1}{2d}, \frac{j}{d} + \frac{1}{2d})$

$z \leftarrow \text{UNIFORM}(-1, 1)$

$C[i, j] \leftarrow \text{ROTATE}(x, y, z, \alpha, \beta, \gamma)$

5 **end**

6 **end**

7 **for**  $k = 1, \dots, n$  **do**

8  $u^1 \leftarrow \text{UNIFORM}(0, 1)$

$u^2 \leftarrow \text{UNIFORM}(0, 1)$

$P[k] \leftarrow \text{EVALBEZIER}(u^1, u^2, C)$

9 **end**

10 **for**  $l = 1, \dots, b$  **do**

11  $u^1 \leftarrow \text{UNIFORM}(0, 1)$

$u^2 \leftarrow \text{UNIFORM}(0, 1)$

$U_B[l] \leftarrow (u^1, u^2)$

$P[n + l] \leftarrow \text{EVALBEZIER}(u^1, u^2, C)$

12 **end**

**Output:** Scattered point cloud

$\mathcal{P} = \mathcal{P}_I \cup \mathcal{P}_B$ ; boundary  
parameters  $U_B$ .

---

the numerical experiments that it performs well for a large range of point cloud sizes.

For our experiments, we also need to generate data from surfaces of higher degree. The procedure is analogous to the biquadratic case and it is summarized in Algorithm 1. The code for data generation is available at <https://github.com/felixfeliz/BIDGCN>.

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  $u_i \in \mathbb{R}^2$ .

### 4.3 Loss function

As we want to train the neural network to predict parameters that minimize (18), 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

$$A_{ij} = B_j(u_i), \quad (39)$$

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  $i \in \mathcal{P}_B$  and the parameters predicted by the neural network if  $i \in \mathcal{P}_I$ .

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

$$b_i = x_i. \quad (40)$$

We then have reformulated (18) into the linear least squares problem

$$\min_{c \in \mathbb{R}^{9 \times 3}} \|Ac - b\|^2 \quad (41)$$

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 (41).

## 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 development of (truncated) hierarchical B-spline (THB-spline) adaptive approximation schemes, see e.g., [39] among others.

We consider a hierarchical spline model  $S : [0, 1]^2 \rightarrow \mathbb{R}^3$  of the form (16) that approximates the input point cloud  $\mathcal{P}$ , where  $B_j :$

$[0, 1]^2 \rightarrow \mathbb{R}$  are bivariate THB-splines [21], whose definition and key properties are briefly recalled in Section 3.3. The state of the art of scattered data fitting schemes address the data parameterization and the design of the hierarchical spline model separately, see [1–3]. 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 the adaptive THB-spline least squares fitting scheme with *moving* parameterization, recently proposed in [4]. 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 (16) are computed by solving the penalized least squares problem

$$\min_{c_0, \dots, c_N} \frac{1}{2} \sum_{i=1}^{|\mathcal{P}|} \|S(u_i) - x_i\|_2^2 + \lambda J(c_0, \dots, c_N), \quad (42)$$

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

$$J(c_0, \dots, c_N) = \int_{[0, 1]^2} \left( \left\| \frac{\partial^2 S}{\partial u_1^2} \right\|_2^2 + \left\| \frac{\partial^2 S}{\partial u_1 \partial u_2} \right\|_2^2 + \left\| \frac{\partial^2 S}{\partial u_2^2} \right\|_2^2 \right) du_1 du_2, \quad (43)$$

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 [1] 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. The authors in [4] then introduced a parameter correction step [22] at every iteration of the fitting scheme. 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 foot-point  $S(\hat{u}_i)$  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

$$\hat{u}_i = \min_{u \in [0,1]^2} \|S(u) - x_i\|_2^2, \quad (44)$$

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 Section 6.3, the initial parameterization plays a fundamental role to obtain high quality results. The THB-spline

---

**Algorithm 2:** Adaptive THB-spline fitting scheme with moving parameters.

---

**Input:** Point cloud  $\{x_i\}_{i=1}^m$  and initial parameterization  $\{u_i\}_{i=1}^m$ , an initial tensor-product B-spline space  $V^0$ , the error tolerance  $\epsilon > 0$ , a maximum number of hierarchical levels  $L$ , and a maximum number of parameter correction steps  $c_{\max}$ .

- 1 Compute the initial tensor-product B-spline approximation  $S \in V^0$  and the point-wise errors  $e_i = \|S(u_i) - x_i\|_2$ , for each  $i = 1, \dots, m$  and set  $\ell = 0$
- 2 **while**  $\max_i e_i > \epsilon$  *and*  $\ell < L$  **do**
  - 2 Solve the least squares problem
$$\min_{c_0, \dots, c_N} \frac{1}{2} \sum_{i=1}^{|\mathcal{P}|} \|S(u_i) - x_i\|_2^2 + \lambda J(c_0, \dots, c_N),$$
  - 3 **for**  $\text{step} = 1, \dots, c_{\max}$  **do**
    - 3 Compute the foot-point projections
$$\min_{x_i \in [0,1]^2} \frac{1}{2} \|S(u_i) - x_i\|_2^2,$$
    - for each  $i = 1, \dots, m$ , and update the parameters and control points of the spline geometry.
  - 4 **end**
  - 5 Compute the errors  $e_i = \|S(u_i) - x_i\|_2$ , for  $i = 1, \dots, m$ . Mark the domain elements where  $e_i > \epsilon$ . Refine the marked cells and the two surrounding rings of cells. Update the hierarchical mesh and hierarchical space  $V^\ell$ . Set  $\ell = \ell + 1$ .
- 6 **end**

**Output:**  $S : [0, 1]^2 \rightarrow \mathbb{R}^3$ , the THB-spline approximant.

---

fitting scheme with moving parameterization is summarized in Algorithm 2.

## 6 Numerical results

We implemented our method using the PyTorch [40] and PyG [41] libraries. The code for training and testing is available at <https://github.com/felixfeliz/BIDGCN>.

We trained the network architecture described in Section 4.1 and visualized in Fig. 8 on our data set consisting of 100,000 point clouds sampled from biquadratic surfaces according to Algorithm 1. Referring to Section 4.3, the loss function during training is the fitting error. We train the network using stochastic gradient descent. In the beginning of the training we set the learning rate to 0.1 and decrease it whenever the loss plateaus.

After the training, 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 6.2.1, 6.2.2, 6.2.3, and 6.2.4). In order to further test the generalization capabilities of BIDGCN on complex data sets and adaptive spline constructions, we consider in Sections 6.3 and 6.4 the adaptive THB-spline approximation scheme with moving parameters introduced in the previous section. To evaluate the accuracy of the hierarchical approximation model, in addition to the mean square error, we report also the maximum approximation error, and the directed Hausdorff distance from the point cloud to the final geometry.

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 1.

In order to test the performance of our method, we generate several test data sets using Algorithm 1. 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

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, 0.005, 0.01, 0.05

Table 1: Summary of hyperparameters.

number of sample points. In particular, we used polynomial bidegrees 2, 3, 4 and 5, Gaussian noise with standard deviation 0.005, 0.01 and 0.05, 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.

### 6.1 Ablation study and comparison with learning-based methods

In our first experiments, we compare the performance of our network architecture with two other methods based on deep neural networks. The first method we compare with is the standard dynamic graph convolutional neural network proposed in [14]. In order to ensure a fair comparison, we trained a network with the same architecture as our BIDGCN shown in Fig 8, 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 4.2. In our plots, we will denote this method by DGCNN.

The second learning-based method that we compare with is the PARGCN method recently proposed in [8]. As described in Section 3.2, 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 3.2. PARGCN was trained on a dataset consisting of point clouds of size 200, sampled from randomly generated biquadratic surfaces.

### 6.1.1 Point clouds sampled from biquadratic surfaces

We first apply the three methods to a test data set consisting of point clouds sampled from biquadratic surfaces with sample size varying between 200 and 2000. 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.

We use each predicted parameterization to fit the input point cloud with a biquadratic surface and compute the mean squared error of the approximation. The computation time as well as the mean squared fitting error is shown in the left column of Fig. 9. All the plots are semi-log plots with a logarithmic scale used for the time and error axes. Our first observation is that the mean squared error 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 Fig. 10. 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.

A second observation from Fig. 9 is that the performance of our network 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 mean squared error 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. Our 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.

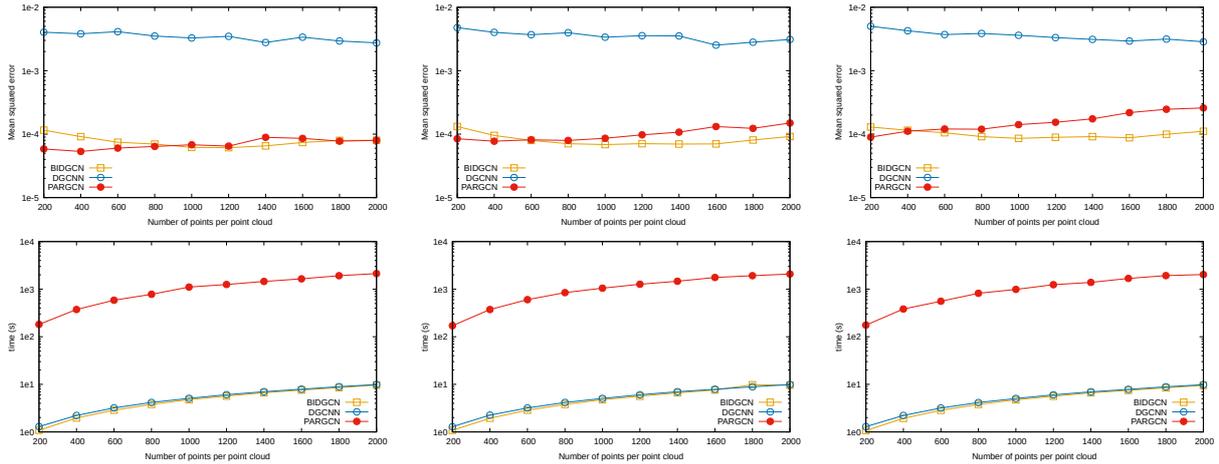
### 6.1.2 Comparison on noisy data

Measured real-world data is always subjected to noise. For this reason, we study the behavior of BIDGCN, DGCNN, and PARGCN when applied to noisy data. In particular, we evaluate all methods on 100 point clouds sampled from biquadratic surfaces that were generated as described in Section 4.2 with added Gaussian noise of standard deviation 0.005 and 0.01.

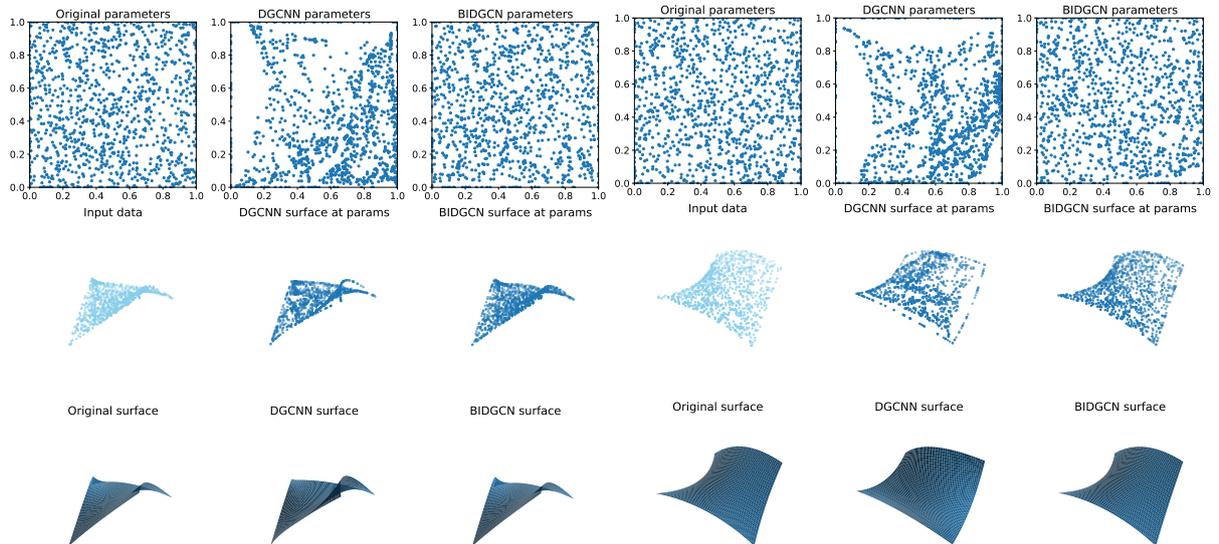
In the middle and right columns of Fig. 9 we show the MSE and computation time when applying the methods to noisy data with Gaussian noise of standard deviation 0.005 and 0.01, respectively. As in the non-noisy case, the plain DGCNN does not result in acceptable fitting errors in the presence of noise. We observe that BIDGCN is more robust than PARGCN with respect to noise and results in better approximations for larger point clouds while needing much smaller amount of computation time. In particular, this means that our proposed network 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.

## 6.2 Comparison with standard methods

In this section, we compare our method with the standard meshless parameterization methods described in Section 3.2.



**Fig. 9:** Mean squared error (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 biquadratic surfaces without noise (left), with Gaussian noise of standard deviation 0.005 (middle) and with Gaussian noise of standard deviation 0.01 (right).



**Fig. 10:** Comparison of our BIDGCN network with a plain DGCNN 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.

### 6.2.1 Point clouds sampled from biquadratic surfaces

First, we study the performance on test data from the same class as the training data, i.e., data sampled from biquadratic surfaces without noise.

We compare our method with the three parameterization methods presented in [5], namely the parameterizations 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

$$r = \frac{3}{\sqrt{|\mathcal{P}|}}. \quad (45)$$

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.

As in the previous section, we evaluate the four methods on a sequence of test data sets, each consisting of 100 point clouds of a fixed size, all sampled from biquadratic surfaces. For each test data set, we report the mean squared error of the fitted surface as well as the total computation time needed to parameterize and approximate all point clouds. As in the previous section, we study the dependence of the accuracy and the computation time on the number of points in each point cloud. The left column of Fig. 11 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. The plots are semi-log plots. We observe that the mean squared errors obtained with 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 the three other methods. In particular, SP is very costly, with a computation time that is about two orders of magnitude higher than the one of BIDGCN.

### 6.2.2 Evaluations on noisy data

We perform the comparison of our proposed method with the previously considered standard methods on noisy data. The middle column of Fig. 11 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. 11 shows their behavior on surfaces with Gaussian noise of standard deviation 0.01.

We observe that when applied to noisy data, BIDGCN performs much better than the three other methods.

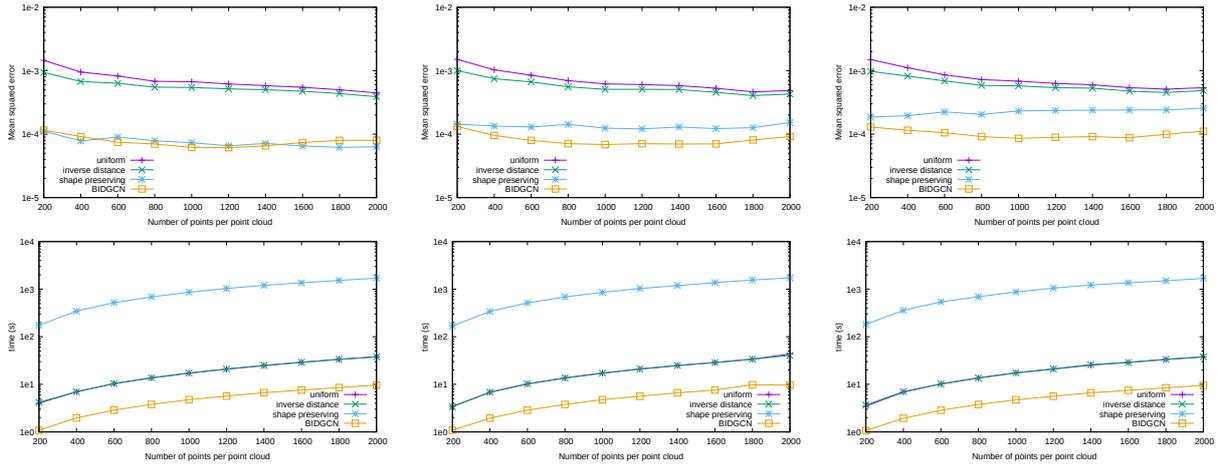
As in the non-noisy data case, we observe that the evaluation of our method is much faster than the evaluations of the other methods. In particular, the speed-up with respect to SP is significant.

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 (45), 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. 12 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 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.

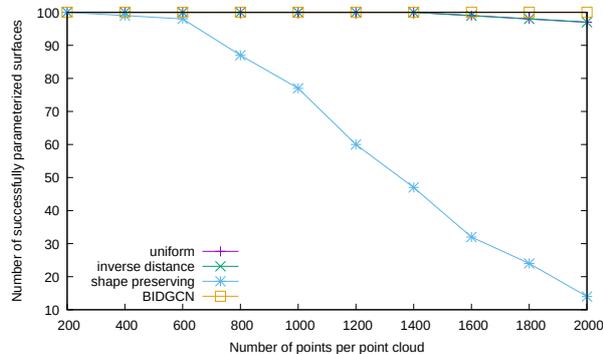
Finally, we report the mean squared errors of the surfaces that were successfully parameterized in Fig. 13. Also in this case the neural network results in significantly smaller errors.

### 6.2.3 Generalization to higher degrees

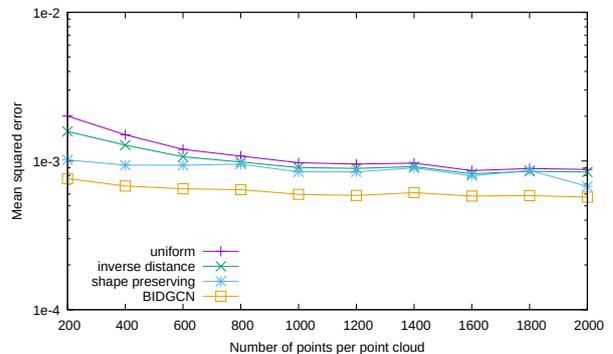
In order to test the generalization properties of the neural network in terms of degree, we apply it to point clouds sampled from surfaces of higher bidegree, namely  $(p, p)$  with  $p = 3, 4, 5$ , with and without Gaussian noise. In doing so, we use tensor-product Bézier surfaces of the same degree for fitting the parameterized point clouds. Fig. 14 shows the mean squared errors as well as the timings for point clouds without noise. We observe that BIDGCN results in similar mean squared



**Fig. 11:** Mean squared error (top) and computation time (bottom) when applying BIDGCN and the three standard methods to point clouds of different sizes sampled from 100 biquadratic surfaces from a test data set without noise (left), with Gaussian noise of standard deviation 0.005 (middle) and with Gaussian noise of standard deviation 0.01 (right).



**Fig. 12:** Number of successfully parameterized surfaces out of 100 biquadratic surfaces with added Gaussian noise of size 0.05.



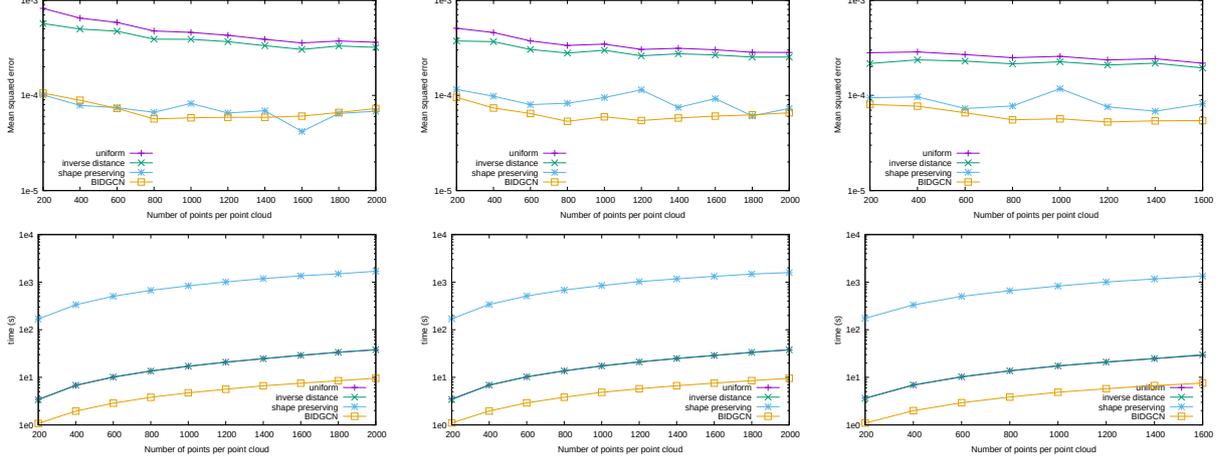
**Fig. 13:** Mean squared error when applying BIDGCN and the three standard methods to point clouds of different sizes sampled from 100 biquadratic surfaces with added Gaussian noise of standard deviation 0.05. All cases where a method did not succeed were removed from the computation of the mean squared error.

errors compared to the SP parameterization. However, the computation time for BIDGCN is more than one order of magnitude smaller with respect to SP.

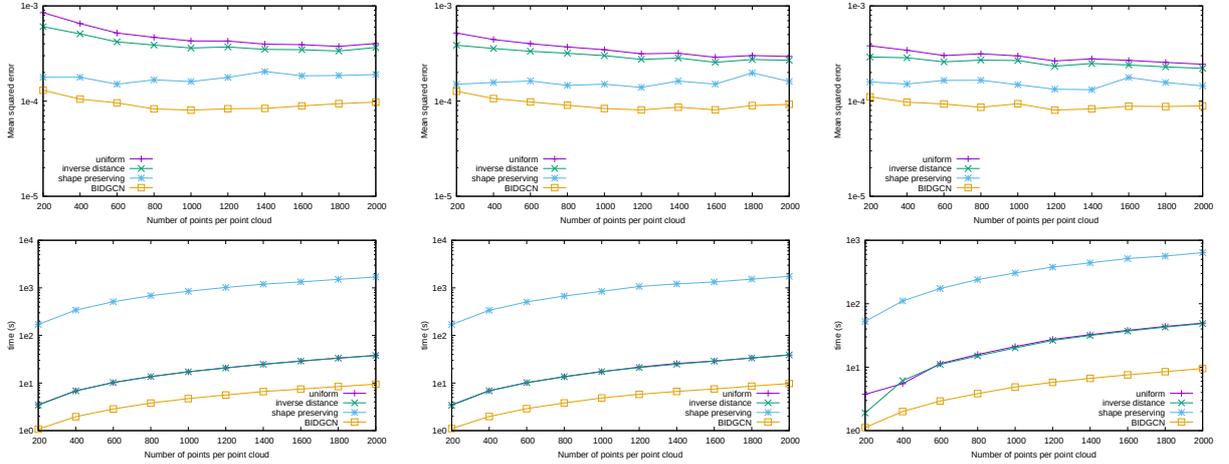
Fig. 15 shows the results on point clouds with added Gaussian noise of standard deviation 0.01. We observe that in this case BIDGCN results in improved mean squared errors compared to the SP parameterization, while the computation time is still over one order of magnitude smaller.

## 6.2.4 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 our neural network. We applied the BIDGCN as well as SP to two point clouds of size 1000 sampled from biquadratic surfaces, one of them without



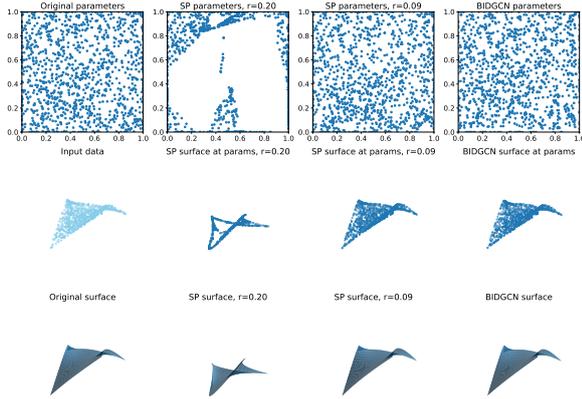
**Fig. 14:** Mean squared error (top) and computation time (bottom) when applying BIDGCN and the three standard methods to point clouds of different sizes sampled from 100 surfaces of bidegree (3, 3) (left), (4, 4) (middle) and (5, 5) (right).



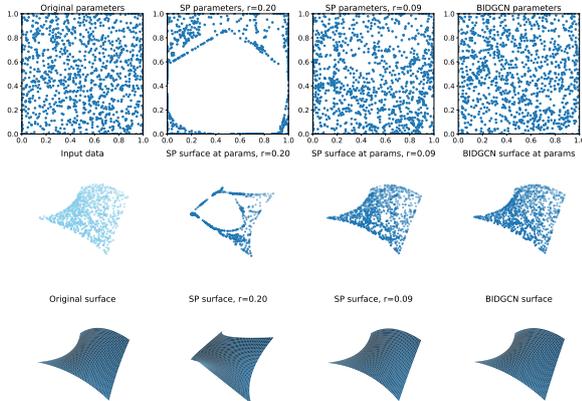
**Fig. 15:** Mean squared error (top) and computation time (bottom) when applying BIDGCN and the three standard methods to point clouds of different sizes sampled from 100 of bidegree (3, 3) (left), (4, 4) (middle) and (5, 5) (right), with added Gaussian noise of standard deviation 0.01.

noise and the other one with added Gaussian noise. For SP we present two choices of the radius:  $r = 0.09 \approx 3/\sqrt{|\mathcal{P}|}$  and  $r = 0.2$ . Fig. 16 and 17 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. 16 does not have any noise, we added Gaussian noise of standard deviation 0.01 to the point cloud shown in Fig. 17. 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



**Fig. 16:** 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. 17:** 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.

able to predict the optimal graph in the first layers of the network architecture and therefore does not require any fine-tuning.

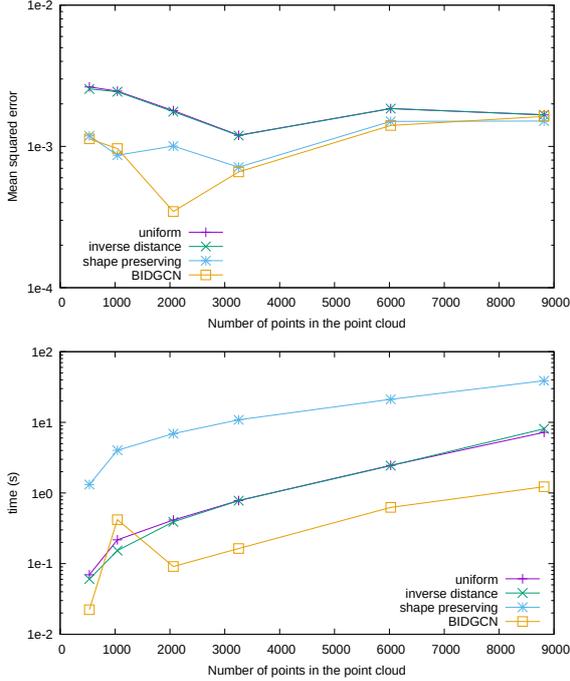
### 6.3 Reconstruction and comparison on real-world data

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. 16 and 17, we execute an heuristic search for  $r = \frac{3}{\sqrt{|\mathcal{P}|}}, 0.05, 0.075, 0.1, \dots, 0.25, 0.275, 0.3$  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 (45) since it leads to the best approximation results for almost all the real data point clouds.

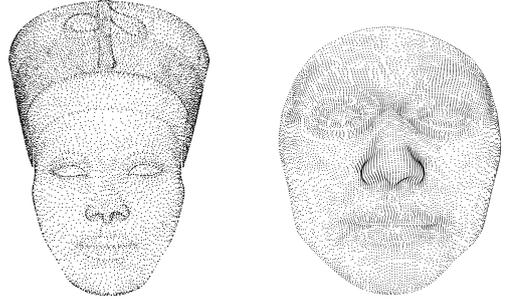
As first analysis, we collect 6 data sets of 532, 1043, 2062, 3253, 6024, 8818 points, acquired from the same Nefertiti face model. We then compute the parametric values for each Nefertiti point cloud with the standard and the proposed BIDGCN methods to subsequently construct a polynomial biquadratic 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. 18. 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 the point cloud of size 8818, shown in Fig. 19 (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 THB-spline model by performing the adaptive fitting scheme with moving parameters described in Section 5 and Algorithm 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. We start from a tensor-product space  $V^0$  with bidegree  $(2, 2)$  and perform 8 adaptive iterations, each of them characterized by 1 step of parameter correction. To properly compare the results,



**Fig. 18:** 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.

the refinement tolerances are chosen such that the final THB-models have (almost) the same number of degrees of freedom. Both BIDGCN and SP are characterized by 4 uniformly refined and 4 locally refined hierarchical levels. Moreover, when the SP method is considered, we obtain a THB-spline model with 5672 degrees of freedom, which is 64% of the total number of scattered input items, characterized by a mean squared error of  $5.50 \cdot 10^{-6}$ , a maximum error of  $3.34 \cdot 10^{-2}$ , and a directed Hausdorff distance from the points to the surface of 1.22. If we parameterize the Nefertiti data with the BIDGCN method, the THB-spline approximation has 5404 degrees of freedom, i.e. 61% of the number of points, and a corresponding mean squared error of  $1.75 \cdot 10^{-6}$ , maximum error of  $1.41 \cdot 10^{-2}$ , and directed Hausdorff distance from the data points to the surface of  $3.46 \cdot 10^{-1}$ . The THB-spline approximations, scaled error distributions on the point cloud, parameter values, and hierarchical



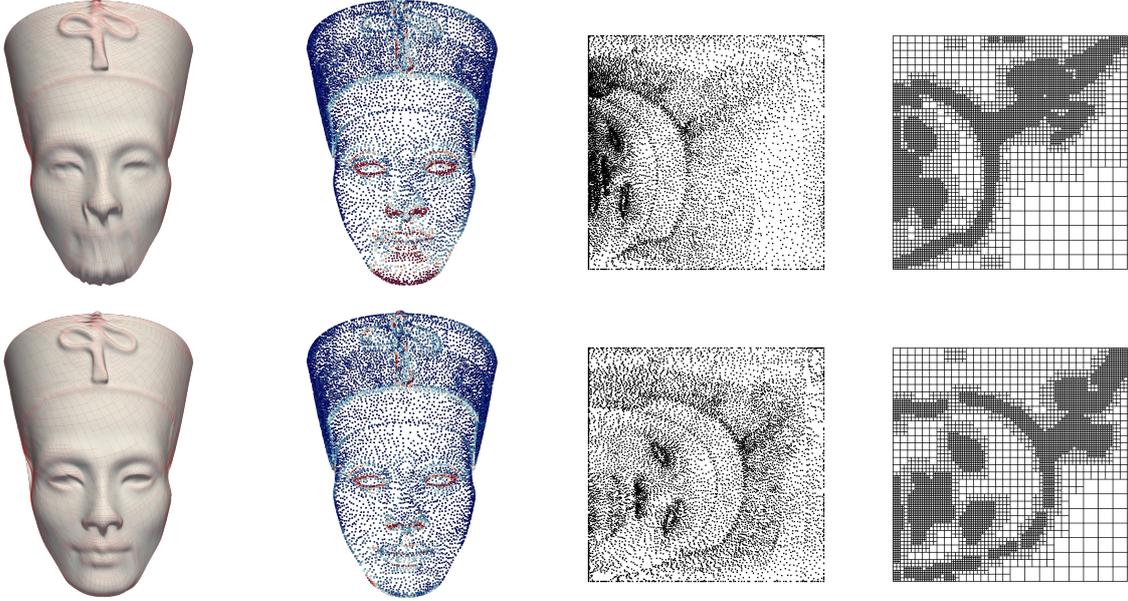
**Fig. 19:** Nefertiti (left) and face (right) point clouds characterized by 8818 and 9283 points, respectively.

meshes obtained with SP and BIDGCN parameterizations are shown on the top and bottom of Fig. 20, 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 affects the final quality of the adaptive spline approximation.

#### 6.4 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. 19. In particular, we reconstruct the THB-spline models of the input point cloud by performing the adaptive fitting scheme with moving parameters described in Section 5 and Algorithm 2, both for bidegree (3,3) and (4,4).

In the first case, we start from a tensor-product space  $V^0$  with bidegree (3,3) and perform 8 adaptive iterations, each of them characterized by 1 step of parameter correction. The smoothing weight  $\lambda$  is set to  $10^{-6}$  and the refinement threshold is  $\epsilon = 8 \cdot 10^{-4}$ . The final reconstructed geometry has 4 uniformly refined and 4 locally refined hierarchical levels. Moreover, it has 2687 degrees of freedom (29% of the number of points) that



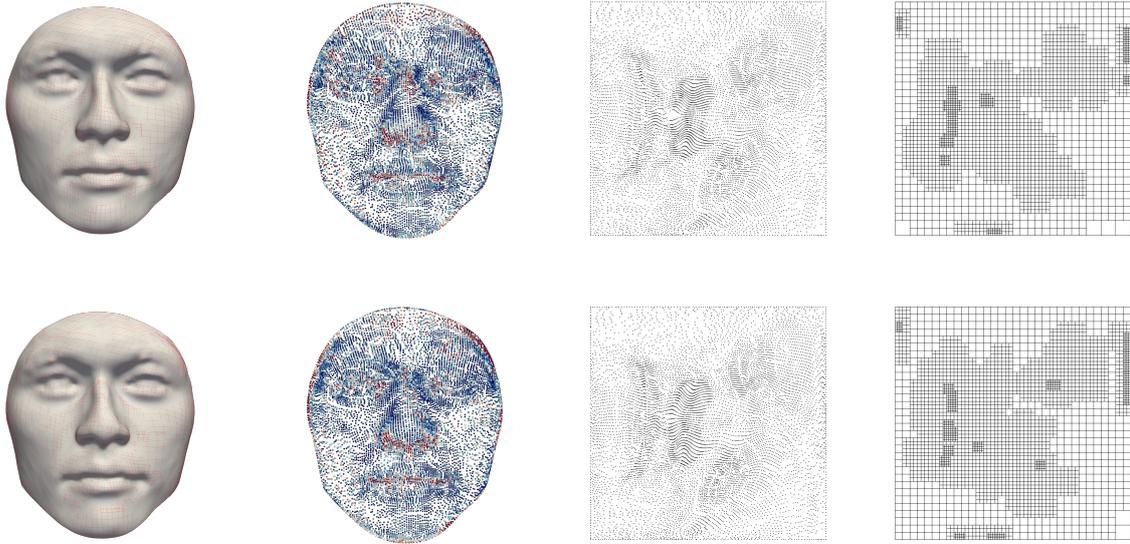
**Fig. 20:** Hierarchical spline model reconstruction of the Nefertiti point cloud shown on left of Fig. 19 for SP parameterization with 5492 degrees of freedom (top), as well as for the BIDGCN parameterization with 5288 degrees of freedom (bottom) with bidegree (2, 2). The reconstructed THB-spline models 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).

approximate the input point cloud registering a mean square error of  $1.45 \cdot 10^{-8}$ , a maximum error of  $1.03 \cdot 10^{-3}$ , and the directed Hausdorff distance from the point cloud to the surface is  $3.43 \cdot 10^{-2}$ . In the second case, the algorithm settings are the same of the previous configuration, except for the bidegree, which is not set to (4, 4), and the refinement threshold, which is lowered to  $6.5 \cdot 10^{-4}$ . The final reconstructed model is a THB-spline geometry with 4 uniformly refined and 4 locally refined hierarchical levels, and 2793 degrees of freedom (30% of the number of points) that approximates the input point cloud with a mean squared error of  $1.44 \cdot 10^{-8}$ , a maximum error of  $1.07 \cdot 10^{-3}$ , and the directed Hausdorff distance from the point cloud to the surface is  $3.23 \cdot 10^{-2}$ . 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. 21.

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 8 iterations of the adaptive THB-spline fitting algorithm with moving parameters, starting from a tensor-product polynomial space  $V^0$  with 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. 22 for both the ship hull (top) and the wind turbine (bottom). As concerns the ship hull data, we set smoothing weight  $\lambda = 10^{-6}$  and the refinement threshold is  $\epsilon = 1 \cdot 10^{-5}$ . The final THB-spline model has 5 uniformly refined and 3 locally refined hierarchical levels. More specifically, it has 4092 degrees of freedom (i.e. 42% of the number of points), a mean square error equal to  $1.30 \cdot 10^{-10}$ , a maximum error equal to  $2.27 \cdot 10^{-4}$ , and registers a directed Hausdorff distance from the points to the surface equals to  $8.45 \cdot 10^{-2}$ .

Finally, for the wind turbine point cloud we set the smoothing weight as  $\lambda = 10^{-6}$  and the refinement threshold as  $\epsilon = 5 \cdot 10^{-5}$ . The final THB-spline model has 5 uniformly refined and 3 locally refined hierarchical levels, 2449 degrees of



**Fig. 21:** Hierarchical spline model reconstruction of the face point cloud shown on the right of Fig. 19 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).

freedom, equivalent in magnitude to 25% of the number of input data, a mean squared error of  $2.13 \cdot 10^{-10}$ , a maximum error of  $4.57 \cdot 10^{-4}$ , and directed Hausdorff distance from the points to the surface of  $1.66 \cdot 10^{-1}$ .

## 7 Discussion

In this final section we give an overview of the presented results and elaborate on possible future research directions.

### 7.1 Concluding remarks

We introduced the novel graph convolutional neural network *BIDGCN* for learning on point clouds. In particular, we developed a new input layer that 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 THB-spline fitting scheme with moving parameters.

One key advantage of our approach is that no manual tuning is needed, and the results outperform classical methods, even when they are fine-tuned for the input instance. For example, 63% better accuracy and a speedup of more than 180 is observed with respect to the shape preserving method, as well as 85% better accuracy and a speedup of around 4 with respect to inverse distance (Fig. 5). BIDGCN behaves favorably also when compared to other parameterization learning methods, see e.g., the comparison with DGCNN and PARGCN in Section 6.1, where the improved scalability of the present approach is also highlighted.

As observed in [14], the dynamic graph approach offers better accuracy results. Computationally, this adds the cost of computing one  $k$ -th



**Fig. 22:** 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  $(5, 2)$ .

nearest neighbor graph in each GCN layer. We did not test the use of a static graph, which might have been faster, yet less accurate.

## 7.2 Future work

In this work, we obtained valuable results on real-world data with BIDGCN trained on synthetic data. However, 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.

Furthermore, we have focused in this work on minimizing the fitting error that is introduced when approximating a point cloud with a smooth spline surface. In future work we will explore the possibilities of using BIDGCN-based networks to optimize point cloud parameterization with respect to more general metrics.

While we successfully treat the case of a point cloud that can be approximated by a single quadrilateral surface patch, surfaces in real world applications can have a more complicated topology. In order to generalize our method to these

cases, we will add a neural network-based segmentation step that makes it possible to approximate such a point cloud with multiple patches, all parameterized using our method.

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 [42], 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 Skłodowska-Curie grant GRAPES No. 860843. The support of the Italian Ministry of University and Research (MUR) through the PRIN project NOTES (No. P2022NC97R), funded by the European Union - Next Generation EU, is also acknowledged.

### *Competing interests*

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

### *Data availability*

The code for generating the synthetic training and test data is archived at the Software Heritage servers: <https://archive.softwareheritage.org/browse/origin/https://github.com/felixfeliz/BIDGCN>.

## References

- [1] 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*
- [2] 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)
- [3] 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)
- [4] Giannelli, C., Imperatore, S., Mantzaflaris, A., Mokriš, D.: Leveraging Moving Parameterization and Adaptive THB-Splines for CAD Surface Reconstruction of Aircraft Engine Components. In: Banterle, F., Caggianese, G., Capece, N., Erra, U., Lupinetti, K., Manfredi, G. (eds.) *Smart Tools and Applications in Graphics – Eurographics Italian Chapter Conference (2023)*. The Eurographics Association
- [5] Floater, M.S., Reimers, M.: Meshless parameterization and surface reconstruction. *Computer Aided Geometric Design* **18**(2), 77–92 (2001)
- [6] Floater, M.S.: Parametrization and smooth approximation of surface triangulations. *Computer aided geometric design* **14**(3), 231–250 (1997)
- [7] 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)
- [8] 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>
- [9] 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)
- [10] Carmo, M.P.d.: *Differential Geometry of Curves and Surfaces: Second Edition*. Dover Publications, Mineola (2017)
- [11] Harary, F.: *Graph Theory*. Addison Wesley

- Longman Publishing Co., Boston (1972)
- [12] 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)
- [13] 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
- [14] 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)
- [15] Welling, M., Kipf, T.N.: Semi-supervised classification with graph convolutional networks. In: *International Conference on Learning Representations (ICLR)* (2017)
- [16] 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)
- [17] Hanocka, R., Hertz, A., Fish, N., Giryas, R., Fleishman, S., Cohen-Or, D.: Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)* **38**(4), 1–12 (2019)
- [18] Sharp, N., Attaiki, S., Crane, K., Ovsjanikov, M.: DiffusionNet: Discretization agnostic learning on surfaces. *ACM Transactions on Graphics (TOG)* **41**(3), 1–16 (2022)
- [19] 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)
- [20] 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)
- [21] Giannelli, C., Jüttler, B., Speleers, H.: THB-splines: The truncated basis for hierarchical splines. *Computer Aided Geometric Design* **29**(7), 485–498 (2012)
- [22] Hoschek, J.: Intrinsic parametrization for approximation. *Computer Aided Geometric Design* **5**(1), 27–31 (1988)
- [23] 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)
- [24] Wang, Y., Zheng, J.: Curvature-guided adaptive T-spline surface fitting. *Computer-Aided Design* **45**(8), 1095–1107 (2013)
- [25] 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)
- [26] Lorentz, G.G.: *Bernstein Polynomials*. Mathematical expositions, no. 8. University of Toronto Press, Toronto (1953)
- [27] Farin, G.: *Curves and Surfaces for CAGD: a Practical Guide*. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, Burlington (2002)
- [28] Piegl, L., Tiller, W.: *The NURBS Book* (2nd Ed.). Springer, Berlin, Heidelberg (1997)
- [29] 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)
- [30] Balta, C., Öztürk, S., Kuncan, M., Kandilli, I.: Dynamic centripetal parameterization method for b-spline curve interpolation. *IEEE Access* **8**, 589–598 (2020)
- [31] Hoschek, J.: Intrinsic parametrization for approximation. *Computer Aided Geometric Design* **5**(1), 27–31 (1988)

- [32] Saux, E., Daniel, M.: An improved Hoschek intrinsic parameterization. *Computer Aided Geometric Design* **20**, 513–521 (2003)
- [33] 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)
- [34] Scholz, F., Jüttler, B.: Parameterization for polynomial curve approximation via residual deep neural networks. *Computer Aided Geometric Design* **85**, 101977 (2021)
- [35] Harary, F., Norman, R.Z.: Some properties of line digraphs. *Rendiconti del Circolo Matematico di Palermo* **9**(2), 161–168 (1960)
- [36] Delaunay, B.: Sur la sphère vide. a la mémoire de georges voronoï. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na* (6), 793–800 (1934)
- [37] De Boor, C.: *A Practical Guide to Splines*. Applied Mathematical Sciences, vol. 27. Springer, New York (2002)
- [38] Forsey, D.R., Bartels, R.H.: Hierarchical B-spline refinement. *SIGGRAPH Comput. Graph.* **22**(4), 205–212 (1988)
- [39] 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)
- [40] 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)
- [41] Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019)
- [42] 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)