Università di Firenze, Università di Perugia, INdAM consorziate nel CIAFM

DOTTORATO DI RICERCA
IN MATEMATICA, INFORMATICA, STATISTICA

CURRICULUM IN MATEMATICA
CICLO XXXVI

Sede amministrativa Università degli Studi di Firenze
Coordinatore Prof. Matteo Focardi

Adaptive spline approximation: data-driven parameterization and CAD model (re-)construction

Settore Scientifico Disciplinare MAT/08

Dottoranda:
Sofia Imperatore

Tutor:
Prof.ssa Carlotta Giannelli
Co-tutor:
Dr. Angelos Mantzaflaris

Coordinatore
Prof. Matteo Focardi

Anni 2020/2023

Sofia Imperatore:
Adaptive spline approximation:
data-driven parameterization and CAD model (re-)construction
Università degli Studi di Firenze

Studiare non serve, studiare comanda.

— Chiara Valerio [173]

ABSTRACT

In this thesis, we combine *Computer Aided Geometric Design* methods with *Deep Learning* technologies. The final objective is to develop and apply advanced geometric reverse engineering methods for the design of complex data-driven free-form spline geometries. In particular, we address the (re-)construction of highly accurate CAD models from point clouds, which is both a fundamental and challenging problem. Depending on the acquisition process, the nature of the data can strongly vary, from uniformly distributed to scattered and affected by noise; yet the reconstructed geometric models are required to be compact, highly accurate, and smooth, while simultaneously capturing key geometric features. An important and crucial step of any parametric model reconstruction scheme consists in solving the parameterization problem, namely to suitably map the input data to a parametric domain. We propose *data-driven parameterization methods* based on (geometric) deep learning to address this problem both in the univariate and multivariate cases, considering either structured or unstructured point cloud configurations. The accuracy of the model obtained with the proposed schemes is higher than the one usually achieved by standard methods. It should be noted that, to the best of our knowledge, data-driven models to address the parameterization problem of scattered data sets in a general multivariate setting were not previously proposed. In addition, we introduce *novel adaptive fitting schemes* with moving parameterization and truncated hierarchical B-splines, based on the optimization of different error metrics. In particular, we propose adaptive alternating and joint optimization methods to optimize the parameter locations and the control points of the (hierarchical) spline geometric model. The alternating methods optimize the parameters separately from the control points computation, whereas the joint approach optimizes the parameters and control points simultaneously. The use of moving parameterization instead of fixed parameter values, when suitably combined with adaptive spline approximation, can significantly improve the resulting geometric model, thus outperforming state-of-the-art hierarchical spline model reconstruction schemes.

PUBLICATIONS

The original results here presented are based on the following publications and manuscripts.

[1] C. Bracco, C. Giannelli, D. Großmann, S. Imperatore, D. Mokriš, and A. Sestini. «THB-Spline Approximations for Turbine Blade Design with Local B-Spline Approximations.» In: *Mathematical and Computational Methods for Modelling, Approximation and Simulation*. Ed. by D. Barrera, S. Remogna, and D. Sbibih. SEMA SIMAI. https://doi.org/10.1007/978-3-030-94339-4_3. Cham: Springer International Publishing, 2022, pp. 63–82.

[2] M. De Vita, C. Giannelli, S. Imperatore, and A. Mantzaflaris. «Parameterization learning with convolutional neural networks for gridded data fitting.» In: *Advances in Information and Communication*. Ed. by K. Arai. Lectures Notes in Networks and Systems. https://doi.org/10.1007/978-981-99-7886-1_32. Cham: Springer Nature Switzerland, 2024, pp. 393–412.

[3] C. Giannelli, S. Imperatore, L. M. Kreusser, E. Loayza-Romero, F. Mohammadi, and N. Villamizar. *A general formulation of reweighted least squares fitting*. Accepted for publication in Mathematics and Computers in Simulation. arXiv preprint arXiv:2404.03742. 2024.

[4] C. Giannelli, S. Imperatore, A. Mantzaflaris, and D. Mokriš. «Leveraging moving parameterization and adaptive THB-splines for CAD surface reconstruction of aircraft engine components.» In: *Smart Tools and Applications in Graphics - Eurographics Italian Chapter Conference*. Ed. by F. Banterle, G. Caggianese, N. Capece, U. Erra, K. Lupinetti, and G. Manfredi. https://doi.org/10.2312/stag.20231301. The Eurographics Association, 2023.

[5] C. Giannelli, S. Imperatore, A. Mantzaflaris, and D. Mokriš. *Efficient alternating and joint distance minimization methods for adaptive surface fitting with THB-splines*. Submitted to Graphical Models. Invited for submission to the "Special Issue STAG 2023: Smart Tool and Applications in Graphics". 2024.

[6] C. Giannelli, S. Imperatore, A. Mantzaflaris, and F. Scholz. *BIDGCN: boundary informed dynamic graph convolutional network for adaptive spline fitting of scattered data*. Under second revision for Neural Computing and Applications. https://inria.hal.science/hal-04313629v1. 2023.

[7] C. Giannelli, S. Imperatore, A. Mantzaflaris, and F. Scholz. «Learning Meshless Parameterization with Graph Convolutional Neural Networks.» In: *Intelligent Sustainable Systems*. Ed. by A. K. Nagar, D. S. Jat, D. K. Mishra, and A. Joshi. https://doi.org/10.1007/978-981-99-7886-1_32. Singapore: Springer Nature Singapore, 2024, pp. 375–387.

ACKNOWLEDGMENTS

I would like to thank all the people who have been next to me during these years and helped me developing not only as a scientist but also as a human.

First of all, I would like to deeply thank my supervisor, Prof.ssa Carlotta Giannelli, for her strong confidence in me, her encouragement of my work, her attention and her constructive supervision. Working together has been a honour and a pleasure. I will always carry with me a great and valuable legacy.

I would also like to deeply thank my co-supervisor, Dr. Angelos Matzanflaris, for his commitment and passion to support each of his students and collaborators. The dedication and patience he invests in teaching coding in G+Smo and C++ are admirable. Thank you very much to take care of our G+Smo community, both scientifically and socially. I am glad to be part of it.

I would also like to thank my reviewers and committee members. Thank you very much to Prof. Matthias Möller not only for carefully reviewing my Thesis, but also for his extreme kindness and for his generosity in hosting me in his research group at TU Delft. Many thanks also to Prof. Bernard Mourrain to accept to review my Thesis, to be a member of my committee, to host me within the AROMATH group at Inria, and finally for the amazing Altitude seminars we shared all together . Thank you very much also to Prof.ssa Alessandra Sestini to accept to be a member of my committee. I am very happy to have someone who has seen me growing since my bachelor studies up to this moment.

Dear professors, you are all a great inspiration for me and helped me a lot to shape the scientist I want to be.

Furthermore, I want to thank Dr. Felix Scholz for our scientific collaborations, fundamental for the success of my thesis. Thank you vert much as well to Dr. Dominik Mokriš, to be a mentor, a collaborator and a friend.

Thanks to Michelangelo to be part of my UNIFamily and my conference partner. Moreovoer, thanks to all the other members of the Numerical Analysis group of Florence for all the valuable and nice time we spent together: Prof. Luigi Brugnano, Prof. Cesare Bracco, Matteo, Lorenzo, Bruno, Gianmarco, Francesco and Mattia.

I would like to extend my sincere thanks to all my new colleagues of the "Dungeons": Anna, Hugo, Peter, Maxence, Miquel, Cosimo and Bernardo. Sharing the office with you is always a great joy and pleasure.

I am deeply grateful to all my friends, the life-long, the old, the new and the alternating ones. Our friendships are inestimable treasures. Thank you very much for the affection you give me to Marta, Chiara, Vanessa, Gabriele, Francesco, Luca, Lorenzo, Leonardo, Maurizio, Claudia, Lorenzo, Alessandro, Jacopo, Alessia,Valentina, Giovanni, Margherita, Cosimo, Emma, Francesco, Veruska, Virginia, Andrea, Folco, Francesco, Alessandra, Giulia, Tiziano and Carys.

It means the a lot to me being still together with my fellow study mates Irene, Vittoria, and Alessia still since the beginning of our journeys as mathematics students. I am so proud of the women you are.

Thank you very much to my sisters, Caterina and Federica, and to Silvia and Gianluca, for the support, the trust and the love we have been exchanging all life long.

I cannot thank enough Miriam to be my person, my sister, and always my mate. It is a privilege of few to develop such a relationship.

My sincere and deep thanks go to my Dutch family, Liduine, Bas, Kamiel, Lucas, and Elise for welcoming me and immediately including me.

I would like to express my gratitude for the unconditional love and support my small, weird, amazing family gives me every single instant of my existence. Thanks to my grandparents Franca and Roberto, to always take so much good care of me. To my uncle Nicola, for his attentions and generosity to me and all the people he meets. Thank you very much to my mother Graziella and to Duccio to become our Accrocchio. Mum, thank you to be my safety net and my anchor, thank you for understanding and encouraging me, but most of all thank you for showing me that it is always worth it to take risks and to put yourself out there in life. A special thank also to Musy and Rajina, to be always literally next to me during all my studies and every single day I worked from home.

With all my heart, thank to Hugo, for believing in me so much, for showing me new perspectives, for cycling with me, for our deep and interesting conversation about life, ethics, math and work, for scaling down my fears and worries every time they arise, for never ordering Hawaii pizza next to me. I am proud of the person you are and I am extremely grateful to share this goal with you. I am looking forward to share all the future ones. I choose you and I love you.

Finally, I would like to dedicate my attention and extreme gratitude to all the women who paved the way for me and all my colleagues. Pursuing and establishing your talent in a world purposely designed to deny it to women is outstanding. I humbly name a few, to whom I have looked at several times in my life. Thanks to be extraordinary ordinary women to Margherita Hack, Chiara

Valerio, Ada Lovelace, Grace Hopper, Samantha Cristoforetti, Suzanne Noël, Rita Levi Montalcini, Maria Skłodowska Curie, Irène Joliot-Curie, Alba Rossi Dell'Acqua, Emma Castelnuovo, Rosalind Franklin, Donna Theo Strickland, Katherine Johnson, Mary Jackson, Dorothy Vaughan, Jocelyn Bell, Elena Long, and again Carlotta Giannelli.

Thank you for everything to my favourite Morgana, Michela Murgia.

CONTENTS

ACRONYMS

A-HDM Alternating Hybrid Distance Minimization

A-PDM Alternating Point Distance Minimization

A-QI Alternating Quasi-Interpolation

A-SDM Alternating Squared Distance Minimization

A-TDM Alternating Tangent Distance Minimization

A-TDMLM Alternating Tangent Distance Minimization Levenberg-Marquardt

AI Artificial Intelligence

BERT Bidirectional Encoder Representations from Transformers

BFGS Broyden-Fletcher-Goldfarb-Shanno

BIDGCN Boundary Informed Dynamic Graph Convolutional neural Network

CAD Computer Aided Design

CAE Computer Aided Engineering

CAGD Computer Aided Geometric Design

CAM Computer Aided Manufacturing

CEN CENtripetal

CHL CHord-Length

CNN Convolutional Neural Network

DAG Directed Acyclic Graph

DGCNN Dynamic Graph Convolutional Neural Network

DHD Direct Hausdorff Distance

DL Deep Learning

DOFs Dregrees Of Freedom

GCN Graph Convolutional neural Network

GD Gradient Descent

HB-spline Hierarchical B-spline

HDM Hybrid Distance Minimization

HLBFGS Hybrid Limited memory Broyden-Fletcher-Goldfarb-Shanno
HSD HauSdorff Distance
IRLS Iterative Reweighted Least Squares
J-PDM Joint Point Distance Minimization
LBFGS Limited memory Broyden-Fletcher-Goldfarb-Shanno
LPSP Local Projection Shape-Preserving parameterization weights
LS Least Squares
MAX MAXimum error
ML Machine Learning
MLP Multi Layer Perceptron
MSE Mean Squared Error
NN Neural Network
NURBS Non Uniform Rational B-Spline
PARCNN PARamerization with Convolutional Neural Network
PARGCN PARameterization with Graph Convolutional neural Network
PC Parameter Correction
PDM Point Distance Minimization
PPN Parameterization for Polynomial curve Network
QI Quasi-Interpolation
RECD RECiprocal Distance parameterization weights
ReLU Rectified Linear Unit activation function
RES RESidual neural network
RMSE Rooted Mean Squared Error
rWLS reWeighted Least Squares
STD STandarD
TDM Tangent Distance Minimization
THB-spline Truncated Hierarchical B-spline
TRA TRAnsformer encoder
UNI UNIform
UNIF UNIForm parameterization weights

1 INTRODUCTION

The digital representation of geometric objects belonging to the real world is of fundamental importance in many scientific disciplines and related application areas. For example, they can be used for simulations, which involve numerical computations directly on the model or its components, as well as for visualization and direct measurements during the design or manufacturing phases.

The input information typically comes in the form of *discrete* data, usually collected as 2D/3D point clouds, hence a *continuous* geometric model needs to be (re-)constructed. Continuous representations facilitate the shaping and manipulation procedures since the dimensionality and complexity of the considered geometric objects are reduced.

The final *goal* of this Thesis consists in developing *automatic*, *robust*, *highly-accurate*, and *efficient* methods for the representations of the input discrete data via a continuous geometric model. To this aim, we properly combine free-form geometric models with data-driven schemes, hence merging *Computer Aided Geometric Design (CAGD)* [55, 91] methods with *Deep Learning (DL)* [21, 76] technologies.

1.1 COMPUTER AIDED GEOMETRIC DESIGN

CAGD provides the mathematical and computational methods as well as suitable modelling and/or approximation schemes for the design, construction, and analysis of geometric objects, i. e. free-form curves, surfaces, and volumes. Examples of its applications are Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM), Geometric Modelling, Robotics, Computer Vision, Computer Graphics, Pattern Recognition, Image Processing, and Scientific Visualization. Key computational tools in this settings are splines, meshes, and subdivision schemes [55, 56, 91], among others.

Splines are piecewise polynomial functions characterized by a certain regularity. The current CAD standard for splines representations relies on B-splines [12] and their non-uniform rational extension, i. e. Non Uniform Rational B-Spline (NURBS) [145, 153]. The multivariate case for standard CAD spline repre-

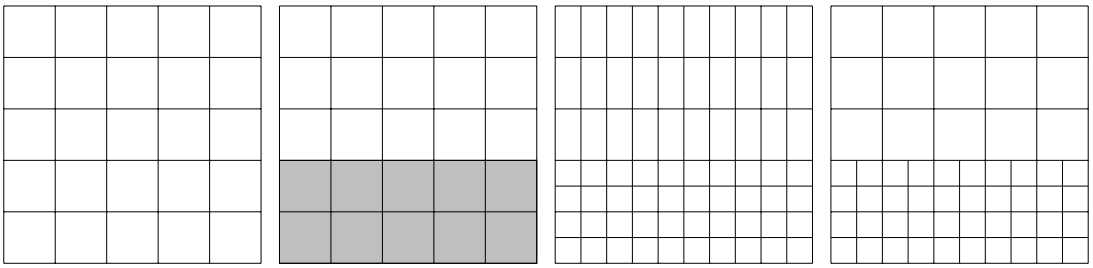


Figure 1.1: Given an initial tensor-product grid (a) and a region of interest for refinement (b), the refined tensor-product grid (c) propagates the refinement beyond the marked areas, while the adaptive model with T-junctions enables local refinement capabilities (d).

sentations relies on the *tensor-product* structure. This construction poses several limitations when we want to properly include *local* properties in the geometric model. More specifically, tensor-product constructions allow only *global* refinement. Hence, unnecessary degrees of freedom are added in regions far away from the one of interest, and this also usually increases the computational costs, see Figure 1.1 (a–b–c). The need to overcome the limits of tensor-product spline constructions, the achieve enhanced flexibility and efficient computation of high-quality approximations, led to the development of new *adaptive* spline spaces defined on extensions of tensor-product constructions on T-meshes, which also include T-junctions and enable local refinement capabilities, see Figure 1.1 (a–b, d).

Spline adaptivity in this setting can be achieved with different approaches, and several proposals can be found in the literature. Among others, we mention: multilevel B-splines [113], defined by a coarse-to-fine hierarchy of control lattices to generate a sequence of B-spline functions, whose sum gives the final model; T-splines [161, 162], defined on meshes where T-junctions between axis-aligned segments are allowed, and related modified versions [95, 117, 119, 181]; S-splines [118], a generalization of T-splines that solves the problem of additional control points propagation in T-spline's local refinement algorithms; PHT-splines [46], namely polynomial splines over hierarchical T-meshes; LR-splines [48], locally refined splines for which the refinement is specified by a sequence of mesh-boxes; U-splines [171], splines defined on unstructured meshes that can accommodate local variation in cell size, polynomial degree, and smoothness. Finally, Hierarchical B-splines (Hierarchical B-splines (HB-splines)) [64, 103] are multilevel B-spline extensions where the tensor-product structure is preserved

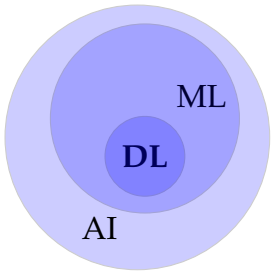


Figure 1.2: AI, ML and DL framework.

at any level of the hierarchy. Their truncated formulation, i. e. Truncated Hierarchical B-splines (*Truncated Hierarchical B-splines (THB-splines)*) [72, 74], is based on HB-splines and relies on the definition of a certain truncation operator. THB-splines are a more flexible tool compared to HB-splines since they have smaller supports and reproduce some characteristic properties of tensor-product B-splines, such as non-negativity and partition of unity. They also guarantee the preservation of coefficients, i. e. they preserve the coefficients of functions expressed in terms of tensor-product B-splines of a certain hierarchical level [73]. For completeness, we recall that a generalization of THB-splines to allow anisotropic mesh refinement can be achieved in the framework of Patchwork B-splines [50].

In this Thesis, we choose to work with THB-splines because of their local tensor-product structure, their key properties, and their ease of implementation.

1.2 DEEP LEARNING

DL is a subset of Machine Learning (ML), the field of Artificial Intelligence (AI) consisting of algorithms, based on numerical and statistical methods, which directly *learn* from data, without being explicitly programmed. The core of ML algorithms consists in their ability to process row data and produce new proper representations, suitable for the considered problem, by automatically setting the value of some internal routine parameters. The novelty of DL, with respect to standard ML, consists in the use of multiple layers of abstraction to progressively extract higher-level features from the raw input data [111]. The relationship between the AI, ML and DL frameworks is summarised in Figure 1.2. It is very well known that DL methods have improved the state-of-the-art in speech recognition, visual object recognition, object detection, and many other domains. In this Thesis, we also show how DL methods can improve the state-of-the-art for (spline) geometric modelling applications.

Neural Networks (NNs) are the computational framework of deep learning. These architectures are usually composed of several processing layers to learn data representations. Specifically, DL routines perform the data feature extraction by employing backpropagation algorithms to identify *intrinsic* relations in big data sets and determine how to adjust the internal parameters of the NN. Among others, some examples of NNs are the following. Multi Layer Perceptrons (Multi Layer Perceptrons (MLPs)) are the core of deep learning models [76], they consist of at least three fully connected layers, intertwined with non-linear (activation) functions. Convolutional Neural Networks (Convolutional Neural Networks (CNNs)) [108] are NNs that use convolutional operators in place of general matrix multiplication in at least one of their layers. They are a specialized kind of neural network for processing data that has a *known grid-like* topology. RESidual neural networks (RESidual neural networks (RESs)) [85] are an extension of CNNs, characterized by shortcut connections, introduced in [166] and also referred to as skip or residual connections, between their input and output, which allow to efficiently train NNs with many ($\gg 100$) hidden layers. Skip connections are also used in the Long Short-Term Memory networks [87] and TRAnsformer encoder (TRA) models [47]. All the above-mentioned NNs are feed-forward networks, i. e. the information flows from the input, through the intermediate computations 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. When feed-forward neural networks are extended to include feed-back connections, they are called Recurrent Neural Networks (RNNs) [155].

In this Thesis, we initially exploit CNNs, which brought several breakthroughs in processing images, video, speech, and audio, and, in general, they thrive when any kind of data involving spatial-correlated patterns has to be processed. On the other hand, CNNs are not suitable to handle data with a non rectilinear-grid like topology, because of the lack of appropriate structures. Consequently, we also employ methods from geometric deep learning [21] to properly process unstructured or unorganized data configurations.

The translation of standard filter operators to graph operators relies on suitable aggregations of vertex and neighbour features. In particular, *Graph Convolutional neural Networks (GCNs)* [186] are a generalization of CNNs to non-Euclidean data, characterized by a *graph structure*, e. g., discrete manifolds, graphs, and general point clouds, 3D shapes, chemical molecules, and social/relational network. Hence, GCNs define the convolution operators on graph domains. Among the various graph convolutional operators available, see again [186] and the

references therein, we exploit the fast localized spectral filters developed in [42] and the dynamic edge convolution operator proposed in [180].

1.3 PROBLEM PRESENTATION

The core problem addressed in this Thesis is the *spline fitting* problem, namely the design of a parametric spline model which approximates a point cloud given as input. The aim is to develop highly-accurate robust and efficient methods for the representation of the discrete input via a continuous spline parametric model.

The data considered in this Thesis are real world data sets or synthetic data that mimic their behaviour. In particular, depending both on the application and the collection method, these data can result in *structured* point grids, meshes, or *scattered* point clouds. We propose new (data-driven) *parameterization* and *fitting* procedures that are able to handle different input data configurations and are suitable for the design of continuous free-from highly accurate and efficient spline models.

The considered data fitting problem can be mathematically described as follows. Given a (noisy) data set of the form,

$$\mathcal{P} = \{\boldsymbol{p}_i \in \mathbb{R}^N \,|\, i = 1, \dots, m\}, \tag{1.1}$$

where $N = 2$ for data points laying a plane and $N = 3$ for data points which belong to the three-dimensional space, find a geometric model $\boldsymbol{s} : \Omega \subseteq \mathbb{R}^D \to \mathbb{R}^N$, which approximates the data $\mathcal{P}$ within a certain tolerance $\epsilon \in \mathbb{R}_{>0}$, in the sense that, for each $i = 1, \dots, m$, $\text{dist}\,(\boldsymbol{s}_i, \boldsymbol{p}_i) \leq \epsilon$, where $\boldsymbol{s}_i$ denotes a point on the geometric model associated with the data point $\boldsymbol{p}_i$, and $\text{dist}\,(\cdot, \cdot)$ is a certain distance metric.

By virtue of their properties, B-splines, and their multivariate hierarchical extension as THB-splines, are a desirable tool for building flexible geometric models. Therefore, the (TH)B-spline fitting problem can be stated in the following way. Given a point cloud as in (1.1) and an error tolerance $\epsilon > 0$, find a (TH)B-spline model $\boldsymbol{s} : \Omega \subseteq \mathbb{R}^D \to \mathbb{R}^N$, so that

$$\text{dist}\,(\boldsymbol{s}(\boldsymbol{u}_i), \boldsymbol{p}_i) \leq \epsilon, \quad \text{with}\, \boldsymbol{u}_i \in \Omega, \quad \text{for each} \quad i = 1, \dots, m, \tag{1.2}$$

, where $\boldsymbol{u}_i$ denotes a point on the parametric domain Ω associated with the data point $\boldsymbol{p}_i$. Solving the problem in (1.2) implies the solution of two essential subproblems: (a) the data parameterization and (b) the definition of the THB-spline approximant $\boldsymbol{s}$. In particular,

(a) Any parametric reconstruction scheme needs to define the data parameterization, namely to define the parametric set

$$\mathcal{U} := \{\boldsymbol{u}_i \in \Omega \subset \mathbb{R}^D \,|\, i = 1, \dots, m\}, \tag{1.3}$$

which assign a parameter value $\boldsymbol{u}_i \in \Omega \subset \mathbb{R}^D$ to each point $\boldsymbol{p}_i \in \mathbb{R}^N$, for $i = 1, \dots, m$. Since the parameters encode intrinsic characteristics of the geometric model representation, estimating a good point cloud parameterization is a fundamental and delicate issue.

(b) The design of a spline model further depends on the characterization of the spline space V, defined by the multivariate degree $\boldsymbol{d} \in \mathbb{N}^D_{>0}$, the *knot* line *placement*, which defines a tessellation of Ω, and, consequently on the computation of the control points.

In this Thesis, we propose solutions to problem (1.2) for gridded and scattered data, by addressing both point (a) and (b) separately. As concerns (a), we propose *data-driven parameterization methods* based on (geometric) deep learning to address this problem both in the univariate and multivariate cases, considering either structured or unstructured point cloud configurations. As concerns (b), we introduce *novel adaptive fitting schemes* with moving parameterization and THB-spline, based on the optimization of different error metrics. In addition, we propose an adaptive approximation paradigm with THB-splines that addresses problems (a) and (b) *simultaneously*. The workflow of the proposed methods is summarized in Figure 1.3.

1.4 CONTRIBUTIONS AND THESIS OUTLINE

Chapter 2 We start in Chapter 2 by providing a selection of the preliminary notions for the contents of this Thesis. In particular, the first part of the Chapter is dedicated to (TH)B-spline constructions: B-splines, tensor-product B-splines, and THB-splines are introduced. The second part of the Chapter focuses on DL and NN architectures. More specifically, we concentrate on CNNs and GCNs, which are the main learning tools employed in the Thesis.

Chapter 3 Subsequently, the main problem addressed in this Thesis – data fitting with THB-splines – is introduced in Chapter 3. Once an initial parameter and mesh configuration are chosen, any adaptive approximation procedure is characterized by four main steps which are successively repeated until a certain stopping criteria is satisfied, i. e.

$$\text{1. SOLVE} \rightarrow \text{2. ESTIMATE} \rightarrow \text{3. MARK} \rightarrow \text{4. REFINE.} \tag{1.4}$$

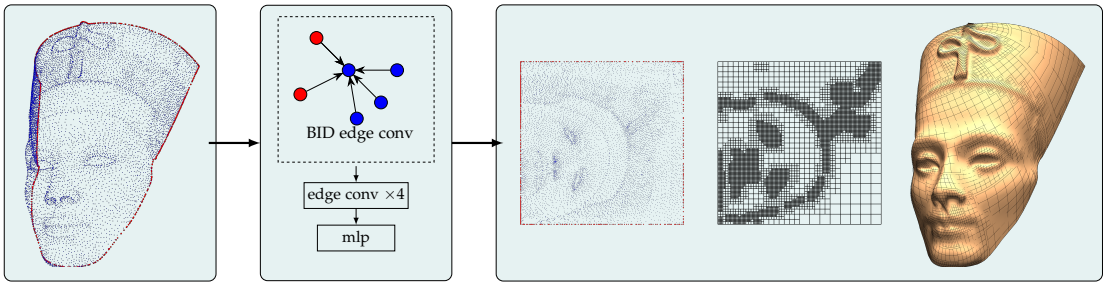


Figure 1.3: Data-driven point cloud parameterization and adaptive spline fitting. From left to right: an input scattered point cloud with interior (blue) and boundary (red) points is processed by the BIDGCN network proposed in [70] to predict the input point parameterization. The predicted parameters are subsequently employed in an adaptive surface reconstruction scheme to get the final hierarchical geometric model.

SOLVE consists in the computation of the approximation on the current mesh; ESTIMATE computes the error estimation according to a suitable error indicator, whose values are then exploited to define a suitable marking strategy in MARK. Finally, REFINE defines the refinement strategies to suitably identify the adaptive mesh to be used in the next iteration of the adaptive loop. The algorithm performance is usually ruled by several input parameters, which establish the starting setting, the adaptive strategy, and the stopping criteria. Note that, once the initial configuration is chosen, the process is fully automatic, namely no user interaction is required.

We start Chapter 3 by reviewing established data fitting methods, i. e. interpolation and weighted least square schemes for an arbitrary approximation space [8]. Subsequently, we present a generalized formulation for reweighted least squares approximations, as convex combination of certain interpolants. We also provide a general strategy to iteratively update the weights according to the approximation error [67]. We then revisit the global Least Squares (LS) fitting scheme with THB-splines presented in [101], based on ordinary LS, by extending the automatic (error-driven) selection of the weights, within the adaptive procedure. Our reWeighted Least Squares (rWLS) scheme is able to tackle the presence of noisy or corrupted data, as well as data corresponding to fundamental geometric features to be represented in the final spline model, see again [67].

Subsequently, we concentrate on the approximation of scattered data sets with THB-splines, based on the two-stage adaptive Quasi-Interpolation (QI) method scheme presented in [19] and further developed in [17]. In particular, we propose to modify the the first stage of the existing scheme from polynomial least squares approximations to least squares B-spline approximations, exploiting also a suitable fairness functional to handle data distributions with a locally varying density of points. We then also adjust the adaptive refinement strategy to properly handle the novelties introduced in the first stage. Chapter 3 concludes with a selection of numerical results of industrial complexity.

Chapter 4 We address the parameterization problem for (TH)B-spline fitting schemes in Chapter 4. Standard parameterization methods both for *gridded* and *scattered* point clouds rely on a barycentric mapping induced from the local neighbourhoods of the points [58, 59, 61, 62, 145]. The complexity of the problem motivates the employment of DL as a viable option to predict an optimal choice of parameter values.

We develop data-driven parameterization methods for the parameterization problem. By introducing *PARamerization with Convolutional Neural Network (PARCNN)* we tackle the parameterization of planar/spatial point sequences and gridded point clouds, whereas *PARamerization with Convolutional Neural Network (PARCNN)* [41] and *Boundary Informed Dynamic Graph Convolutional neural Network (BIDGCN)* [70] address the parameterization of scattered data. In all these cases, we employ NNs characterized by convolutional operators, defined on the considered domain, for the parameterization learning problem and to assign parameter values at an *arbitrary* number of points for their subsequent use in (TH)B-spline (adaptive) approximation schemes. Differently from the other models, BIDGCN is able to process boundary conditions in addition to the standard vertex features of the discrete surface by a *new* graph convolution operator that contains two separate trainable message functions.

For each learning model, we provide details on the architecture developed, the training strategy, the model hyper-parameters and the data employed for training, test and validation phases. The performances of each method are illustrated by the related numerical examples.

All the proposed parameterization learning methods are agnostic to the dimension of the input, can generalize to point clouds of different dimensions, are robust to noise, outperform closed-form, heuristic and data-driven choices of the parameterization, and produce high-quality parameterizations for (TH)B-spline reconstruction schemes. In addition, BIDGCN overcomes the failing issues of standard meshless parameterization methods which happen in case of sparse data neighbourhoods. To the best of our knowledge, dimension-independent

data-driven models to address the parameterization problem of point sequences and data sets in a general multivariate setting were not previously proposed.

In Chapter 5, we introduce novel adaptive approximation schemes with THB-spline and *moving parameterization*. State-of-the-art (TH)B-spline fitting methods address the parameterization problem, the construction of the spline space and the definition of the control net separately. In particular, given $\mathcal{P}$, a suitable parameterization $\mathcal{U}$ is computed and, on such a *fixed* parameterization, both the spline space, and the approximating spline model is defined. Instead, we follow up on the idea that, as the refinement proceeds in an adaptive setting, not only the geometric model but also the parameter values of each data point should be optimized. In this Thesis we propose two strategies to properly embed the parameterization within the adaptive model construction and to deal with a *moving parameterization*, rather than a fixed one. *Chapter 5*

The first strategy consists of enriching the adaptive approximating loop with iterative applications of the Parameter Correction (PC) routine [90]. In addition, we propose different adaptive fitting algorithm by exploiting diverse approximation schemes involved in the SOLVE step, in combination with PC routine [69]. In particular, we develop the adaptive *Alternating Point Distance Minimization (A-PDM)* fitting scheme, as the adaptive multivariate extension of the Point Distance Minimization (PDM) method originally presented in [90] for B-spline curve fitting. Moreover, by taking into account within the error term also the normal direction of the current geometric model at each iterative step, we develop the adaptive *Alternating Tangent Distance Minimization (A-TDM)* fitting scheme, based on the Tangent Distance Minimization (TDM) [9, 123, 179]. As suggested in [10, 130], we then combine the PDM and TDM error measures and take into account the discrete curvatures and the point-wise error distribution of the current model, to formulate the adaptive Alternating Hybrid Distance Minimization (A-HDM) multivariate fitting technique [69]. Finally, we exploit the introduction of PC also within an adaptive hierarchical QI methods, based on two-stage approximation scheme with local B-splines [68].

The second strategy to move the parameters consists of addressing the parameterization problem within the SOLVE step of the adaptive loop. The computation of the approximation on the current mesh in this case is performed by the solution a non-linear *Joint Point Distance Minimization (J-PDM)*, which consists in *simultaneously* computing the optimal parameter sites and control points, as proposed in [189] for the case of B-spline curve fitting. Therefore, with this method, we avoid solving a linear system of equations and performing PC at every adaptive iteration.

Our study reveals that using a moving parameterization, instead of a fixed one, can improve the fitting results while also reducing the number of degrees of freedom required to achieve a certain accuracy. It can also lead to earlier termination of the adaptive process, thus providing more compact models with less refinement depth and outperforming state-of-the-art hierarchical spline model reconstruction schemes.

Chapter 6 Finally, Chapter 6 concludes the Thesis by summarizing the research results and providing new future research directions and perspectives.

The algorithms related to the learning parameterization models, see Chapter 4, have been implemented in `Python` with the open-source PyTorch and PyTorch Geometric libraries [57, 99, 143]. The novel graph convolution operator introduced for the BIDGCN model, together with the network architecture can be found at the following repository: `https://github.com/felixfeliz/BIDGCN`. The code for PARCNN and PARameterization with Graph Convolutional neural Network (PARGCN) will be made available on request.

The algorithms related to the adaptive fitting schemes with THB-splines, see Chapter 3 and Chapter 5, have been implemented in `C++` with the open-source G+Smo library [94, 129]. The developed code for the proposed algorithms has been integrated and will be available in the next releases.

The original results presented in this Thesis are based on [16, 67] (Chapter 3), [41, 70, 71] (Chapter 4), and [68, 69] (Chapter 5).

2

PRELIMINARIES

This Chapter provides a selection of preliminary notions for the contents of this Thesis. We provide CAGD and DL definitions, notations, and operations that will be essential throughout the following Chapters. After introducing univariate B-spline constructions and their tensor-product multivariate extensions, we focus on the hierarchical spline model by considering THB-splines. Subsequently, we illustrate NN architectures and mainly focus on CNNs and GCNs. The present Chapter is mostly based on [12, 21, 74, 76].

2.1 ADAPTIVE SPLINE CONSTRUCTIONS

CAGD is concerned with the mathematical and computational methods for geometric modelling. This discipline is primarily devoted to the construction and analysis of free-form curves, surfaces, and volumes, such as splines, meshes, and subdivision surfaces, as well as suitable modelling and/or approximation schemes for their generation, analysis, and manipulation [55, 91]. The range of CAGD applications is very wide, e. g., CAD, Computer Aided Engineering (CAE)/CAM, computer graphics, path planning and motion control, robotics, and scientific visualization, among others.

The CAGD methods considered in this Thesis concern splines and related approximation and modelling schemes. The computational CAGD tools employed are B-splines, piecewise polynomial functions with a certain regularity, and their multivariate adaptive extensions. In particular, in this Section we present the main concepts of polynomial B-splines and truncated hierarchical B-splines.

2.1.1 *B-splines*

The choice of the approximation space plays a fundamental role in the final accuracy of the geometric model. This is the case of polynomial interpolation models, which might be affected by oscillations for too high polynomial degree [156]. A solution to this issue is provided by considering spline models, which allow to keep the polynomial degree low, while improving the model accuracy by increasing the number of polynomial pieces for its definition.

Let $\Omega = [a, b] \subset \mathbb{R}$ be a real interval and let $\boldsymbol{\tau} \coloneqq \{\tau_0, \dots, \tau_L\}$ be a partition of $[a, b]$ so that $a \equiv \tau_0 < \cdots < \tau_i < \tau_{i+1} < \ldots < \tau_L \equiv b$ and define the *knot-vector* $\boldsymbol{\tau} \coloneqq [\tau_0, \dots, \tau_L]$. Moreover, let $\gamma_i \coloneqq [\tau_i, \tau_{i+1})$ for $i = 0, \dots, L-2$, $\gamma_{L-1} \coloneqq [\tau_{L-1}, \tau_L]$, choose $d \in \mathbb{N}$ a polynomial degree and set $k \coloneqq d + 1$ the corresponding polynomial order.

Definition 1. *The space of piecewise polynomial functions with degree d and knot-vector $\boldsymbol{\tau}$ is defined as follows*

$$P_{d,\boldsymbol{\tau}} \coloneqq \left\{ f : \Omega \to \mathbb{R} \mid f_{|\gamma_i} \in \Pi_d, \text{ for } i = 0, \dots, L-1 \right\}, \tag{2.1}$$

where Π_d is the space of polynomials of degree less or equal to d.

$P_{d,\boldsymbol{\tau}}$ is a vector space on the field $\mathbb{R}$ and its dimension is $\dim(P_{d,\boldsymbol{\tau}}) = L(d+1)$.

Remark 1. *No regularity conditions are imposed in Definition 1, which implies that the elements of $P_{d,\boldsymbol{\tau}}$ may also not be continuous.*

In many CAGD applications, flexible tools are often desirable. Thereby, let $\boldsymbol{\mu} = \{\mu_1, \dots, \mu_{L-1}\}$ the multiplicity vector with $1 \leq \mu_i \leq k$ for $i = 1, \dots, L-1$ and define the *extended knot-vector*

$$\boldsymbol{t} \coloneqq [t_0, \dots, t_{k-2}, t_{k-1}, \dots, t_{n+1}, t_{n+2}, \dots, t_{n+k}],$$

with

$$\boldsymbol{\tau} = [t_{k-1}, \dots, t_{n+1}] = [\tau_0, \underbrace{\tau_1, \dots, \tau_1}_{\mu_1}, \dots, \underbrace{\tau_{L-1}, \dots, \tau_{L-1}}_{\mu_{L-1}}, \tau_L].$$

Remark 2. *The knots $t_0, \dots, t_{k-2}$ and $t_{n+2}, \dots, t_{n+k}$ are the so called* auxiliary knots*; they can be chosen arbitrarily, and the only constraint they have to fulfill is to be ordered. A common choice consists in choosing them all coincident, respectively, to t_{k-1} and t_{n+1}, i.e. $t_0 \equiv \ldots \equiv t_{k-2} \equiv t_{k-1}$ and $t_{n+1} \equiv t_{n+2} \ldots \equiv t_{n+k-1}$. An extended knot vector with this auxiliary knot choice is said to be* open*.*

Definition 2. *The space of polynomial spline functions with degree d, global regularity $d - \max\{\boldsymbol{\mu}\}$ and knot-vector $\boldsymbol{t}$ is defined as follows*

$$V \coloneqq \left\{ f : \Omega \to \mathbb{R} \mid f^{(r)}(\tau_i^-) = f^{(r)}(\tau_i^+), \text{ for } r = 0, \dots, d - \mu_i,\ i = 1, \dots, L-1 \right\}. \tag{2.2}$$

V is a vector space on the field $\mathbb{R}$, $\Pi_d \subset V_{d,\boldsymbol{t}} \subset P_{d,\boldsymbol{\tau}}$ and its dimension is

$$n+1 := \dim V = L\,(d+1) - \sum_{i=1}^{L-1} (d-\mu_i+1) = (m+1) + \sum_{i=1}^{L-1} \mu_i = k+\mu,$$

where $\mu = \sum_{i=1}^{L-1} \mu_i$.

The vector space V can be generated by $n+1$ univariate B-splines of degree d over $[a,b]$, which can be recursively defined [12, 13].

Definition 3. *Let $\beta_{j,k} : \Omega \to \mathbb{R}$ indicate the j-th B-spline of order $k \geq 1$, for each $j = 0,\dots,n$. For $x \in \mathbb{R}$, if $k = 1$,*

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

and $\beta_{j,1}(t_{n+1}) = 1$. If $k \geq 2$,

$$\beta_{j,k}(x) = \omega_{j,k}(x)\beta_{j,k-1}(x) + \big(1 - \omega_{j+1,k}(x)\big)\,\beta_{j+1,k-1}(x),$$

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

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

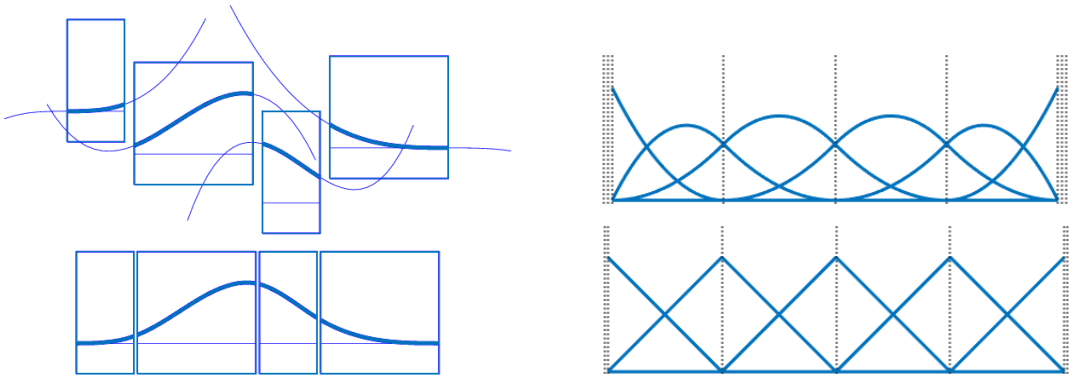
Note that each B-spline $\beta_{j,k}$ for $j = 0,\dots,n$ is a piecewise polynomial function of degree d, which has maximum local smoothness on each subinterval of the partition $\boldsymbol{t}$. On the other hand, the regularity at each unique knot $t_j \in \boldsymbol{t}$ is $d - \mu_j$, where $1 \leq \mu_j \leq k$ is the number of times the knot value t_j appears in $\boldsymbol{t}$. Moreover, the presence of repeated knots in $\boldsymbol{t}$ implies that choosing $t_{\hat{i}} \equiv t_{\hat{i}+1}$ for some $\hat{i}$ modifies the polynomial function $\omega_{\hat{i},\hat{r}}$ for some $\hat{r} \leq k$, which can result identically 0 and leads to a reduction of regularity of the B-splines $\beta_{\hat{i},r}$ for $r \geq \hat{r}$.

Univariate B-splines are characterized by three key properties, which follow directly from Definition 3, namely for each $j = 0,\dots,n$,

(i) non-negativity, i. e. $\beta_{j,k}(x) \geq 0$ for all $x \in \mathbb{R}$;

(ii) local support, i. e. $\beta_{j,k}(x) = 0$, if $x \notin [t_j, t_{j+k})$;

(iii) partition of unity, i. e. $\sum_{j=0}^{n} \beta_{j,k}(x) = 1$ if $x \in [t_{k-1}, t_{n+1}] \equiv [a,b]$.



(a) (b)

Figure 2.1: (a) A B-spline of degree $d = 3$ and its decomposition in 4 polynomial pieces. (b) B-splines of degree $d = 1$ (top) and $d = 2$ (bottom) on a uniform open knot-vectors.

Finally, the $n + 1$ B-splines defined on the open knot vector $\boldsymbol{t}$ are piecewise polynomials of degree d; they belong to $C^{d-s}(\Omega)$, where $s := \max_{1,\dots,L-1} \mu_i$, and form a basis for the space V [37, 38], hence

$$V := \operatorname{span}\{\beta_{0,k}, \dots, \beta_{n,k}\}.$$

Figure 2.1 (a) illustrates a B-spline of degree $d = 3$ and its four polynomial pieces. Figure 2.1 (b) illustrates a B-spline basis of degree $d = 1$ (top) and a B-spline basis of degree $d = 2$ (bottom) on open knot-vectors $\boldsymbol{t}$ with uniform interior knots $\boldsymbol{\tau}$.

Remark 3. *Let $d \in \mathbb{N}$ and $k = d + 1$ be a polynomial degree and order respectively, and let $[0, 1] = \Omega \subset \mathbb{R}$. If a knot vector with no interior knots, i. e. $\boldsymbol{\tau} = [0, 1]$ and the corresponding extended* open *knot vector, i. e.*

$$\boldsymbol{t} = [\underbrace{0, \dots, 0}_{k}, \underbrace{1, \dots, 1}_{k}], \tag{2.3}$$

are considered, the B-spline basis in Definition 3 corresponds to the Bernstein polynomial basis on Ω [55, 127], namely

$$\beta_{j,k}(x) := \binom{k-1}{j} x^j (1-x)^{k-j-1}, \text{ for each } j = 0, \dots, d. \tag{2.4}$$

Once the extended knot-vector $\boldsymbol{t}$, the degree d and the order k are set, any B-spline geometry $\boldsymbol{s} : \Omega \to \mathbb{R}^N$ can be represented as

$$\boldsymbol{s}(x) = \sum_{j=0}^{n} \boldsymbol{c}_j \beta_{j,k}(x), \quad \text{for} \quad x \in \Omega, \tag{2.5}$$

with coefficients $\boldsymbol{c}_j \in \mathbb{R}^N$, for $N \in \mathbb{N}_{\geq 1}$. In particular, for $N = 2$ and $N = 3$ the spline object $\boldsymbol{s}$ corresponds to a planar or spatial spline curve respectively. An illustrative example of these two cases is provided in Figure 2.2.

Remark 4. *By choosing the space configuration in Remark 3, the representation in (2.5) is called the Bernstein-Bézier form.*

For B-spline curves the following properties hold.

1. B-spline curves are invariant under affine transformations.
2. The value of a spline curve $\boldsymbol{s}$ at a site $x \in \Omega$ depends only on a convex combination of k coefficients $\boldsymbol{c}_j \in \mathbb{R}^N$. It follows that, if $x \in [t_i, t_{i+1}]$ for some $i = k-1, \ldots, n$,
$$\boldsymbol{s}(x) = \sum_{j=i-k+1}^{i} \boldsymbol{c}_j \beta_{j,k}(x).$$
3. Each segment of the B-spline curve belongs to the convex-hull of (only) k consecutive coefficients $\boldsymbol{c}_j \in \mathbb{R}^N$, hence it is usually addressed as *strong convex-hull*.

Remark 5. *The relationship between the value of the spline curve $\boldsymbol{s}$ and its coefficients $\boldsymbol{c}_j$, for $j = 0, \ldots, n$ has led to addressing them as* control points.

From these properties, it clearly follows that B-spline curves are extremely ductile, and consequently, they represent effective geometric modelling tools. Additional properties, e. g., *end-point interpolation* or *periodicity*, as well as algorithms, e. g., *knot-insertion* or *degree elevation*, to design and efficiently manipulate B-spline curves can be found in [13, 35, 55].

2.1.2 *Tensor-product B-splines*

Univariate B-splines can be easily extended to higher dimensions by considering a tensor-product construction. In particular, let Ω be a hypercube of $\mathbb{R}^D$, i. e.

$$\Omega := \bigotimes_{h=1}^{D} [a_h, b_h], \text{ with } [a_h, b_h] \subset \mathbb{R} \text{ for } h = 1, \ldots, D.$$

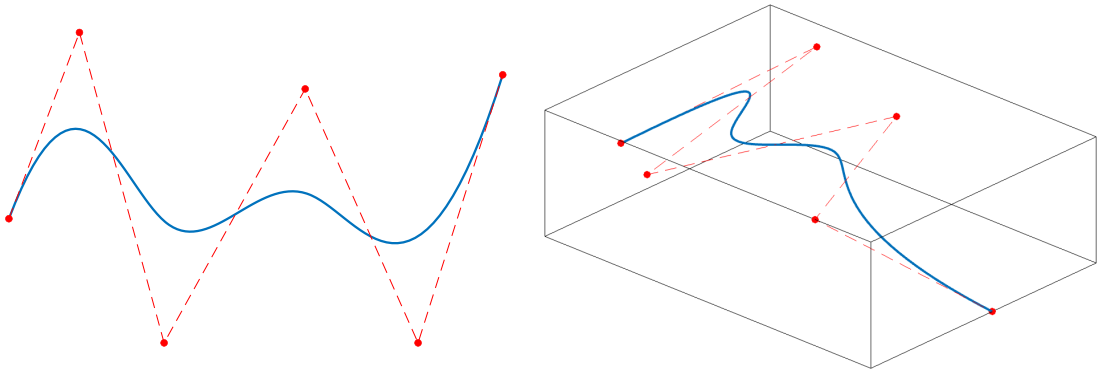


Figure 2.2: Planar (left) and spatial (right) B-spline curves, together with their control nets.

Let $\boldsymbol{d} = (d_1, \ldots, d_D)$ be the polynomial multi-degree, $\boldsymbol{k} = (k_1, \ldots, k_D)$ be the corresponding polynomial multi-order and a suitable knot-vector in each parametric direction, i. e. $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_D$. We define the tensor-product mesh G as $G := \bigtimes_{h=1}^{D} \boldsymbol{t}_h$. A tensor-product B-spline $\beta_j : \Omega \subset \mathbb{R}^D \to \mathbb{R}$ is then defined as the tensor-product of D univariate B-splines,

$$\beta_{j,\boldsymbol{k}} := \prod_{h=1}^{D} \beta_{j_h,k_h}, \text{ for } j_h = 0, \ldots, n_h, \ h = 1, \ldots, D.$$

More precisely, they can be all identified by a set of multi-indices, i. e.

$$\mathcal{B} = \left\{ \beta_j \mid j \in \Gamma_{\boldsymbol{k}} \right\},$$

with

$$\Gamma_{\boldsymbol{k}} := \{ j = (j_1, \ldots, j_D) \mid j_h = 1, \ldots, n_h, \ h = 1, \ldots, D \}. \tag{2.6}$$

Because of the tensor construction, many of the simple algebraic properties of univariate B-splines hold. In particular, non-negativity, locality, and partition of unity. Moreover, the tensor-product B-splines defined on the grid G are piecewise multivariate polynomials of multi-degree $\boldsymbol{d}$, their regularity is characterized by the multiplicity of the knot-lines of G and they form a basis for the tensor-product space, i. e.

$$V := \operatorname{span}\{\mathcal{B}\}.$$

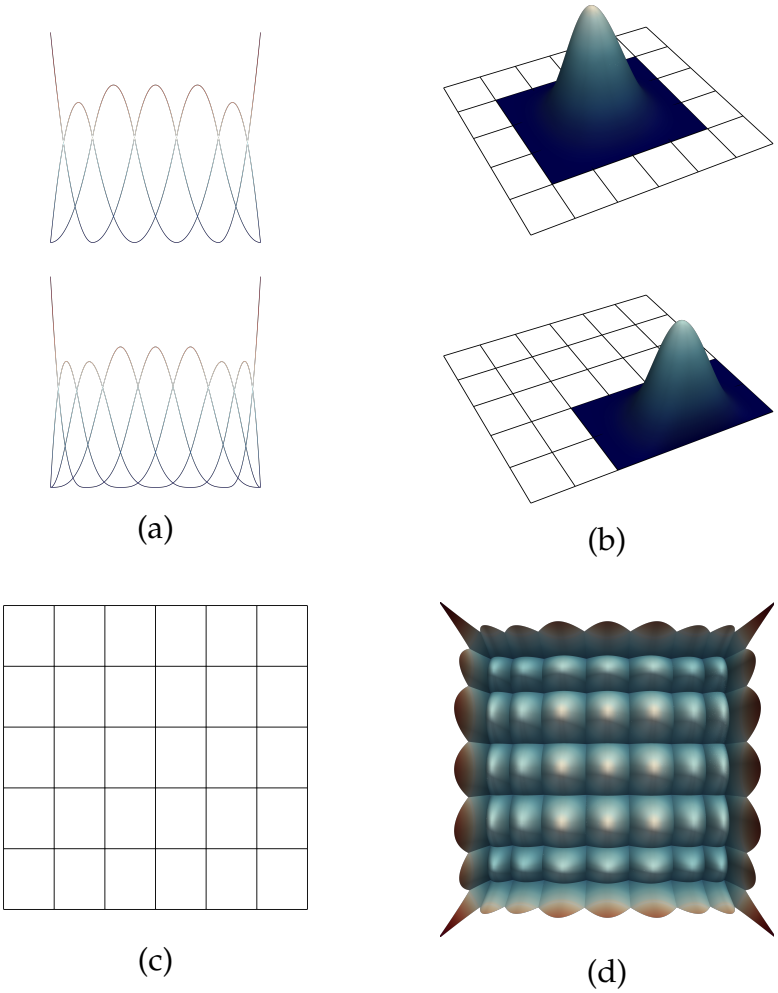


Figure 2.3: (a) Univariate B-spline basis of degree $d = 3$ on a uniform open knot-vector of 5 interior knots (top) and with degree $d = 2$ on a uniform open knot-vector with 4 interior knots (bottom). (b) Two example of the basis functions obtained by the tensor-product construction of the basis in (a). (c) The tensor-product grid G, built with the knot-vectors for the univariate basis in (a). (d) The final tensor-product basis of bi-degree $\boldsymbol{d} = (3, 2)$ on G.

In Figure 2.3, we show an example of bivariate tensor product B-splines. In particular, in (a) we show the univariate B-spline basis of degree 3 (top) and 2 (bottom). In (b) we show 2 of the final $8 \times 9 = 72$ basis functions, namely $\beta_{4,4}\beta_{1,3}$ (top) and $\beta_{3,4}\beta_{5,3}$ (bottom). Finally, the bivariate tensor-product grid is shown in (c) and the resulting tensor-product basis is reported in (d).

Similarly to the univariate case, once a tensor-product grid G is chosen and both the multi-degree $\boldsymbol{d}$ and the multi-order $\boldsymbol{k}$ are set, any tensor-product B-spline geometry $\boldsymbol{s} : \Omega \subset \mathbb{R}^D \to \mathbb{R}^N$ can be represented as

$$\boldsymbol{s}(x) = \sum_{j \in \Gamma_{\boldsymbol{k}}} \boldsymbol{c}_j \beta_{j,\boldsymbol{k}}(x), \quad \text{for} \quad x \in \Omega, \tag{2.7}$$

with coefficients $\boldsymbol{c}_j \in \mathbb{R}^N$, for $N \in \mathbb{N}_{\geq 1}$. In particular, if $D = 2$, for $N = 2$ and $N = 3$ the spline geometry $\boldsymbol{s}$ corresponds to a planar or surface spline patch, respectively.

Remark 6. *Likewise in the univariate scenario, see Remark 3 and 4, let $\Omega = [0,1]^D$ and a tensor-product grid G built on* open *knot-vectors with no interior knots, i. e. $\boldsymbol{t}_h$ as in (2.3) for each $h = 1, \ldots, D$. Then the tensor-product B-spline basis corresponds to the tensor-product Bernstein polynomial basis and the tensor-product B-spline geometries as in (2.7) are called tensor-product polynomial geometries in Bernstein-Bézier form.*

Also for the tensor-product multi-variate case, the properties addressed for B-spline curves hold, i. e. affine invariance, locality, and strong convex-hull. Therefore, tensor-product B-splines are quite flexible, and additionally, tensor-product B-spline basis have good numerical properties and are easy to evaluate [12, 105]. Note that the coefficients of a multivariate tensor-product B-spline geometry are called *control points*, as in Remark 5.

The main limitation of tensor-product B-spline structures arises when a refinement strategy occurs. More precisely, the space dimension tends to increase very fast when mesh refinement is considered, hence leading to computationally very expensive models. In particular, each time that a knot-line is inserted in one direction, the dimension of the tensor-product space increases a number of times equal to the product of the dimensions of the univariate spaces in all the other directions. Furthermore, when dealing with approximation problems, tensor-product structures are proven not to be flexible enough since they allow only global refinement guided by knot lines and preclude suitable adaptive and local refinement. Thereby, degrees of freedom are also added in regions far away from the area of interest. Moreover, these degrees of freedom in excess not only increase the computational costs in memory and time, but they are also unnecessary to improve the accuracy of the final model due to the local action of each control point. Figure 2.4 illustrates an example in the bivariate case of the non-locality of tensor-product refinement. In particular, a tensor-product grid (top left) is shown together with the corresponding tensor-product basis. Since the model needs to be refined, along a diagonal direction, the only possible way to refine the marked elements (top center) consists of performing a

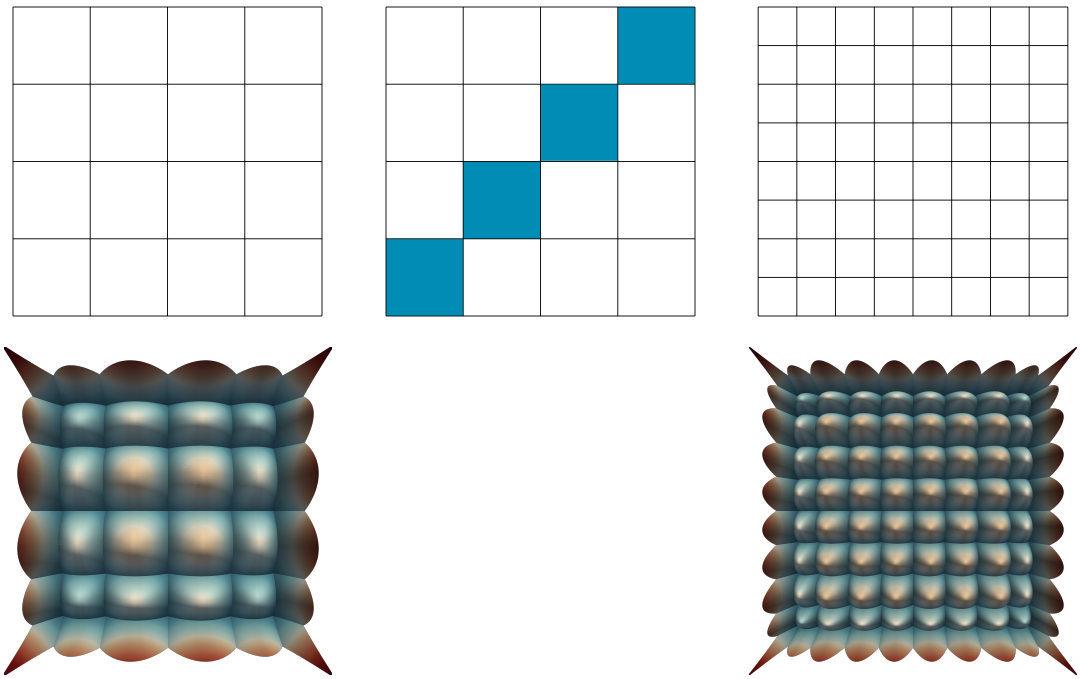


Figure 2.4: Tensor-product refinement: the initial mesh (top left) and basis (bottom left); the marked region of interest for refinement (top center); the final refined mesh (top right) and basis (bottom right).

global uniform refinement, as shown in the resulting mesh (top right) and basis (bottom right).

2.1.3 *THB-splines*

When dealing with multivariate settings, the tensor-product constructions allow only a global distribution of the Dregrees Of Freedom (DOFs) for the problem at hand, whereas local spline refinement enables the possibility of achieving flexible and accurate models by strongly reducing the total number of control points when compared with standard tensor-product B-spline representations. This type of local mesh refinement is intrinsically supported by hierarchical splines [64], where local refinement is achieved by introducing tensor-product B-splines on multiple hierarchical levels. A selection mechanism to properly identify a Hierarchical B-spline (HB-spline) basis was originally introduced in [103].

Let Ω be a hypercube of $\mathbb{R}^D$ and consider a nested sequence of L tensor-product B-spline spaces

$$V^0 \subset \ldots \subset V^{L-1}, \tag{2.8}$$

defined on $\Omega \subset \mathbb{R}^D$. More precisely for each level $\ell = 0, \ldots, L-1$, let $\boldsymbol{d}$ be a polynomial multi-degree and $\boldsymbol{k}$ is the corresponding multi-order. Moreover, let $\beta_j^\ell : \Omega \to \mathbb{R}$ be the $j-$th tensor-product B-spline basis function of the space V^ℓ. Hence,

$$V^\ell = \operatorname{span}\left\{ \beta_j^\ell \mid j \in \Gamma_{\boldsymbol{k}}^\ell \right\},$$

where $\Gamma_{\boldsymbol{k}}^\ell$ is the set of indices for the tensor-product B-spline basis of level ℓ, as defined in (2.6).

Remark 7. *As long as the tensor-product B-spline spaces are nested as in* (2.8)*, they can be characterized by different degrees/orders along the levels, see [177].*

In addition, each V^ℓ is associated with a rectilinear-grid G^ℓ and their (non-empty) quadrilateral elements q are commonly addressed as mesh cells of level ℓ. We also consider a nested sequence of closed domains

$$\Omega \equiv \Omega^0 \supset \ldots \supset \Omega^L = \emptyset,$$

so that each Ω^ℓ is the union of a subset of cells of G^ℓ, thus each domain boundary $\partial\Omega^\ell$ is aligned with the knot lines of V^ℓ, for each $\ell = 0, \ldots, L-1$. The hierarchical mesh is defined as

$$\mathcal{M} := \left\{ q \in \mathbb{G}^\ell \mid \ell = 0, \ldots, L-1 \right\},$$

where for each $\ell = 0, \ldots, L-1$,

$$\mathbb{G}^\ell := \left\{ q \in G^\ell \,|\, q \subset \Omega^\ell \setminus \Omega^{\ell+1} \right\}$$

is called the set of *active cells* of level ℓ.

Remark 8. *Note that the definition of a rectilinear-grid G or a hierarchical mesh $\mathcal{M}$ over the domain Ω induces a domain tessellation $T(\Omega)$, as considered in Chapter 3.*

The hierarchical spline construction consists in replacing any B-spline of level ℓ with support completely contained in $\Omega^{\ell+1}$ by B-splines at successively refined levels.

Definition 4. *The HB-spline basis is defined as follows*

$$\mathcal{H} := \left\{ \beta_j^\ell \mid j \in A_{\boldsymbol{k}}^\ell,\ \ell = 0, \dots, L-1 \right\} \tag{2.9}$$

where

$$A_{\boldsymbol{k}}^\ell := \left\{ j \in \Gamma_{\boldsymbol{k}}^\ell \mid supp\left(\beta_j^\ell\right) \subseteq \Omega^\ell \wedge supp\left(\beta_j^\ell\right) \not\subseteq \Omega^{\ell+1} \right\} \tag{2.10}$$

is the set of indices of active functions *and* $supp\left(\beta_j^\ell\right)$ *denotes the intersection of the support of* β_j^ℓ *with* Ω^0*. The corresponding hierarchical space is defined as*

$$V := span\left\{\mathcal{H}\right\}.$$

A univariate example of HB-spline space with 2 level of refinement is provided in Figure 2.5, where two *nested* univariate B-spline basis of degree $d = 2$ are defined on a domain $\Omega \subset \mathbb{R}$. More specifically, the basis displayed in (a) defined on an open uniform knot-vector with 7 interior knots, whereas the basis displayed in (b) it is built on an open uniform knot-vector with 16 interior knots, obtained by dyadically splitting each non-empty knot span of the coarse knot-vector. The subdomain Ω^1 is highlight in red in (c), (d) and (e), where $\Omega^1 \subset \Omega^0 \equiv \Omega$. In particular, (c) and (d) show the active univariate B-splines at level $\ell = 0$ and $\ell = 1$, respectively, according to the selection mechanism (2.10). Finally, (d) illustrates the 2 level HB-spline basis, according to Definition 4 and (2.10).

The hierarchical structure enables to localize the refinement only in the area of the domain where it is needed. The difference between bivariate tensor-product refinement and hierarchical dyadic mesh refinement on the same area of interest, as well the the respective basis, can be visualized in Figure 2.6. In particular, we show the refined mesh and the corresponding basis of bi-degree $\boldsymbol{d} = (2, 2)$, obtained by refining the same area of interest, up to 3 hierarchical levels. It is evident that in the hierarchical approach the refinement is local and adds less basis elements. In this case, a dyadic refinement procedure is employed, that is a single element is subdivided into four smaller elements. Again, the small element can be refined as well, so that we get a refined grid over several levels. Note that the refined tensor-product basis has 729 basis functions, whereas the refined hierarchical basis functions has 184 basis functions.

Directly from Definition 4 and from tensor-product B-spline properties, it immediately follows that HB-splines have a local support and are non-negative. Moreover, linear independence can be achieved with the help of the local linear independence of the spline basis at each hierarchical level [103, 177].

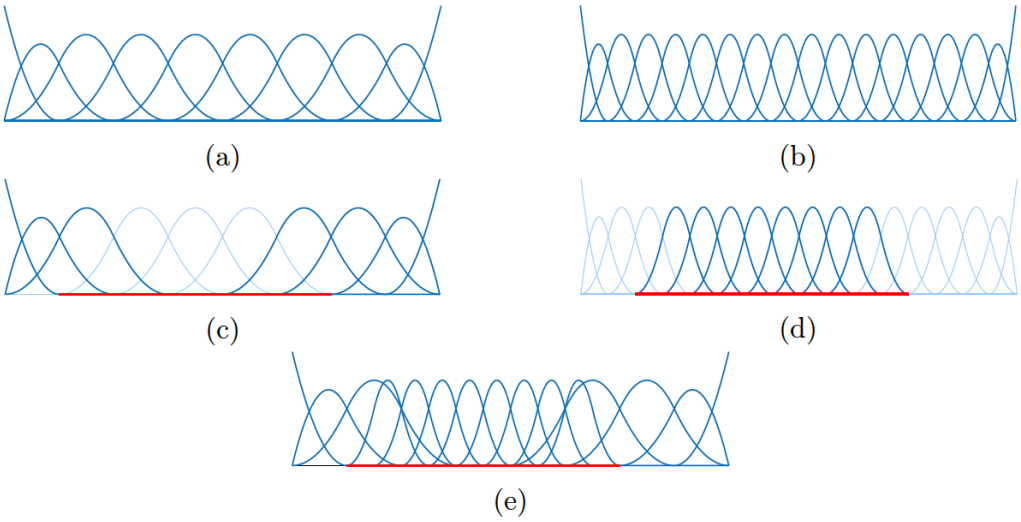


Figure 2.5: Example of univariate HB-spline basis of degree 2: (a) B-spline basis for V^0; (b) B-spline basis for V^1; (c) the active B-splines of level 0; (d) the active B-splines of level 1; (e) the HB-spline basis. The refinement area Ω^1 is also shown (red line).

On the other hand, HB-spline basis are characterized by different overlaps of coarse and fine B-spline basis function supports, and the partition of unity property of the basis is lost. A simple way to recover the partition of unity property consists in suitably weighting the basis, by applying a adequate scaling factor to the HB-spline basis functions [177]. Nevertheless, there is no guarantee that the weights would be positive or non-zero, and for some HB-spline configuration a proper scaling is not guaranteed to exists at all. In order to ensure the existence of a scaling and the positivity of the weights, additional constraints on the subdomain configurations need to be considered [177].

This motivates the construction of another basis for the hierarchical spline space, the Truncated Hierarchical B-spline (THB-spline) basis [74]. The THB-spline basis maintains the partition of unity property without any additional assumption on the considered subdomains. In addition to the partition of unity, THB-spline basis provides various useful properties in the mathematical framework of hierarchical splines, such as strong stability, full approximation power, and efficient implementation [100, 165]. In particular, a THB-spline basis can be obtained from a HB-spline basis, by applying a truncation mechanism to each HB-spline basis function. In particular, the truncation mechanism preserves all the properties of HB-spline, such as linear independence and non-negativity.

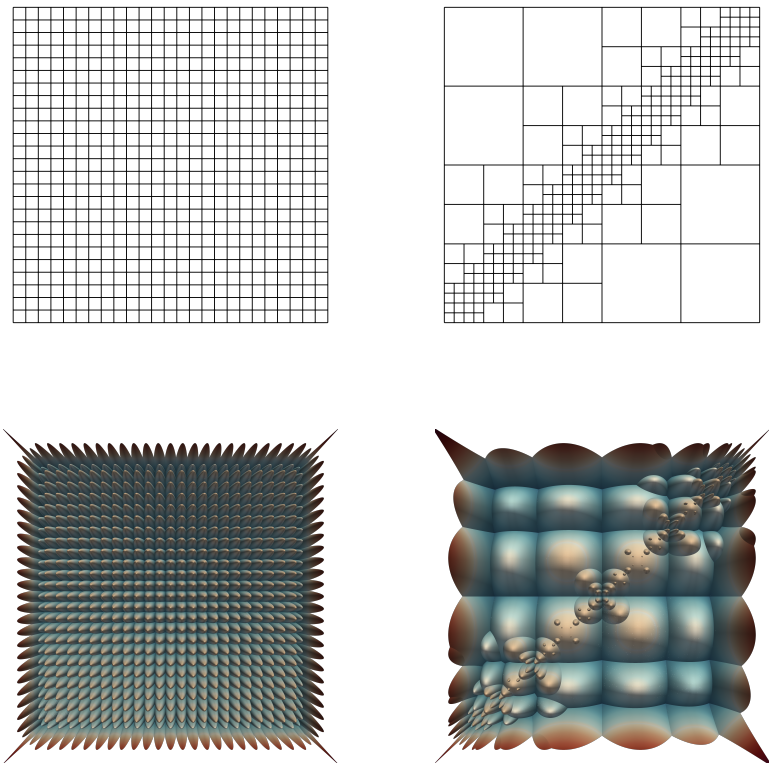


Figure 2.6: Comparison between tensor-product (left) and hierarchical refinement on 3 levels (right): the tensor-product mesh (top left) with the corresponding basis (bottom left) and the hierarchical mesh (top right) with the hierarchical basis (bottom right).

For any $\ell = 0, \dots, L-2$, let $s \in V^\ell \subset V^{\ell+1}$ be a tensor-product spline represented in terms of the basis of the refined space $V^{\ell+1}$ as

$$s(\boldsymbol{x}) = \sum_{j \in \Gamma_k^{\ell+1}} c_j^{\ell+1}(s) \beta_j^{\ell+1}(\boldsymbol{x}), \quad \text{for } \boldsymbol{x} \in \Omega,$$

for suitable coefficients $c_j^{\ell+1}(s)$, for each $j \in \Gamma_{\boldsymbol{k}}^{\ell+1}$. By separating the basis functions of $V^{\ell+1}$ whose supports are not completely contained in $\Omega^{\ell+1}$, from the remaining ones, it holds,

$$s(\boldsymbol{x}) = \sum_{\substack{j \in \Gamma_{\boldsymbol{k}}^{\ell+1} \\ \operatorname{supp}\left(\beta_j^{\ell+1}\right) \subseteq \Omega^{\ell+1}}} c_j^{\ell+1}(s)\beta_j^{\ell+1}(\boldsymbol{x}) + \sum_{\substack{j \in \Gamma_{\boldsymbol{k}}^{\ell+1} \\ \operatorname{supp}\left(\beta_j^{\ell+1}\right) \not\subseteq \Omega^{\ell+1}}} c_j^{\ell+1}(s)\beta_j^{\ell+1}(\boldsymbol{x}), \text{ for } \boldsymbol{x} \in \Omega.$$

Definition 5. *The truncation of $s \in V^\ell$ at level $\ell+1$ is defined as*

$$\operatorname{trunc}^{\ell+1}(s) := \sum_{\substack{j \in \Gamma_{\boldsymbol{k}}^{\ell+1} \\ supp\left(\beta_j^{\ell}\right) \not\subseteq \Omega^{\ell+1}}} c_j^{\ell+1}(s)\, \beta_j^{\ell+1} \tag{2.11}$$

and the cumulative truncation with respect to all finer levels is

$$\operatorname{Trunc}^{\ell+1}(s) := \operatorname{trunc}^{L-1}\left(\operatorname{trunc}^{L-2}\left(\ldots\left(\operatorname{trunc}^{\ell+1}(s)\right)\ldots\right)\right), \tag{2.12}$$

with $\operatorname{Trunc}^L(s) \equiv s$, for $s \in V^{L-1}$. Finally, the THB-spline basis for the hierarchical space can be defined as

$$\mathcal{T} := \left\{\tau_j^\ell = \operatorname{Trunc}^{\ell+1}\left(\beta_j^\ell\right) \mid j \in A_{\boldsymbol{k}}^\ell,\ \ell = 0, \ldots, L-1\right\}, \tag{2.13}$$

where the B-spline β_j^ℓ is the mother *B-spline of the truncated B-spline τ_j^ℓ.*

Figure 2.7 shows an example of the truncation mechanism for univariate B-splines over 2 hierarchical levels, for $\ell = 0, 1$. More precisely, the coarse B-spline (dashed) belonging to V^0 can be represented in terms of the 5 finer B-splines (in the background) belonging to V^1, for suitable coefficients $c_0, \ldots, c_4$. In addition, we choose a subdomain Ω^1, highlighted in red in the picture. The truncation of the coarser B-spline corresponds to setting to 0 the coefficients of the finer B-splines whose supports are fully contained in Ω^1, i. e. the last two finer B-splines on the right of the picture.

THB-splines are non-negative, have local support, form a partition of unity, and generate the same space of the HB-spline basis [74], i. e.

$$\operatorname{span}\{\mathcal{H}\} \equiv \operatorname{span}\{\mathcal{T}\}. \tag{2.14}$$

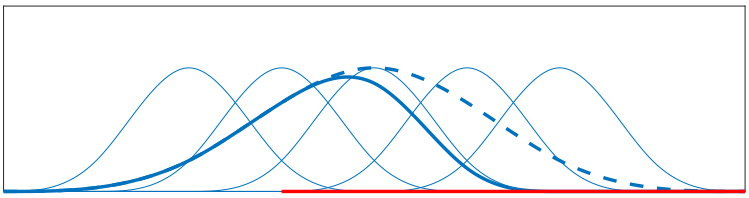


Figure 2.7: Truncation mechanism defined in (2.11) for a B-spline of degree 4. The B-spline of level $\ell = 0$ (dashed line) is truncated over the subdomain Ω^1 (in red) via 5 finer B-splines (thin lines) for $\ell = 1$ and results in a THB-spline (thick line).

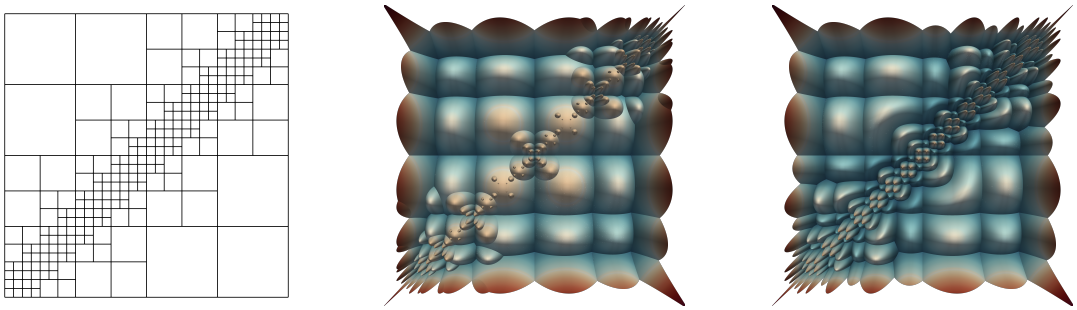


Figure 2.8: Comparison between hierarchical bases: the hierarchical mesh (left) and the corresponding HB-spline (center) and THB-spline (right) bases.

The effectiveness of employing THB-splines both for geometric design and isogeometric analysis has been shown in [72], among others. Figure 2.8 shows the comparison, for the same hierarchical space characterized by the hierarchical mesh on the left, of the HB-spline basis and THB-spline basis, in the center and on the right of the picture, respectively. Note that the THB-spline basis functions reduce the overlaps of the supports related to THB-splines introduced at different hierarchical levels.

Because of their properties, THB-splines are a desirable tool for building flexible geometric models. They have been here presented in their more general form, where their construction can be developed in any parametric and physical dimension, namely for any $D, N \in \mathbb{N}_{\geq 1}$. In particular, a (TH)B-spline geometry is a linear combination of (TH)B-splines, defined as

$$\boldsymbol{s}(\boldsymbol{x}) = \sum_{\ell=0}^{L-1} \sum_{j \in A_k^\ell} \boldsymbol{c}_j \tau_j^\ell(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \tag{2.15}$$

with $\boldsymbol{c}_j \in \mathbb{R}^N$ for $j \in A^\ell_{\boldsymbol{k}}$, $\ell = 0, \dots, L-1$. For $D = 1$ a planar $(N = 2)$ or spatial $(N = 3)$ (TH)B-spline curve can be specified, whereas for $D = 2$ and $N = 3$ a THB-spline surface can be constructed.

2.2 DEEP LEARNING WITH NEURAL NETWORKS

For centuries science has been an activity operated by humans for humans; nevertheless, with the advent of programmable computers, a new perspective has been developed and still subsists, in which science becomes an activity performed by humans and machines for humans and machines. The paradigm of using computers to simulate human intelligence was first described in [172], where the "Turing test" was proposed to determine whether computers were capable of human intelligence. Subsequently, the term Artificial Intelligence (AI) was coined [134] as the science and engineering of developing non-biological systems that mimic human behaviour and intelligence. AI began as a simple series of *if statements* and has developed over several decades, resulting in more advanced algorithms that perform similarly to the human brain. Nowadays, it is a flourishing field with many applications and active research topics. We exploit intelligent software to automate routine labour, understand speeches or images, make diagnoses in medicine, and support scientific research. Furthermore, the recent increase in data availability led to the development of new AI techniques capable of handling large data volumes. In this respect, Machine Learning (ML) is the field of AI consisting of algorithms based on numerical and statistical methods that directly *learn* from data without being explicitly programmed [137, 158]. More specifically, throughout this Thesis we will consider the following concept of learning.

Definition 6 ([137])**.** *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at task in T, as measured by P improves with experience E.*

Consequently, any learning model should be able to properly generalize based on its own experience. The core of any learning problem are *data*, which we assume to come in a finite amount, hence this results in an intrinsically difficult problem as we have to generalize from limited data information.

The data employed to define ML models are usually separated into different *pairwise disjoint* sets, which play a role at different stages of the creation of the models, i. e. *training*, *validation* and *test* sets. More precisely, the training dataset is a set of data used to tune the parameters of the ML model during the

so-called *training* phase. Many methodologies that search through training data for empirical relationships often may happen to overfit the data, meaning that they can identify and exploit apparent relationships in the training data that lack general applicability. For this reason, a validation data set is often used during the training phase to analyze an unbiased evaluation of the model and implement the early stopping of the training phase. Finally, the test data set is used to provide an unbiased evaluation of the *final* trained model and analyze its generalization capacity.

The nature of the data plays a role also for the definition of the learning paradigm. In particular, we recall the following learning types, among others.

(i) *Supervised Learning*: in this case, the data comprises examples of the input items, along with their corresponding target. The goal consists in building a function that maps (new) data on expected target values. Examples of its applications are email spam detection or hand-written digits recognition. In general, this is the most common scenario associated with classification, regression, and ranking problems.

(ii) *Unsupervised learning*: in this case, all the data is unlabelled. On the other hand, usually the data contains many features which are therefore exploited to learn useful properties of the structure of this data. Since in general no labelled example is available in this setting, it can be difficult to quantitatively evaluate the performance of a model. Clustering and dimensionality reduction are example of unsupervised learning problems.

The contributions of this Thesis related to learning problems mainly act within the variety of *unsupervised learning* framework. In order to provide a rough idea of the learning approaches, we list additional existing learning paradigms, which we do not tackle in this Thesis, i. e. Semi-supervised learning [28], Transductive inference [174], Online learning [157] and Reinforcement learning [170]. In practice, many more intermediate learning scenarios may been encountered, see [7] for more details.

The core of ML algorithms consists in their ability to process row-data and produce new proper representations, suitable for the problem at hand, by automatically set the value of some internal routine parameters. Despite the remarkable results which ML can achieve, these systems often require a careful domain expertise guidance to design a proper feature extractor procedures. Moreover, ML models also suffer of a lack of generalisation to different scenarios or input data. Subsequently, DL was designed to overcome the ML limitations, such as the need of an expert guidance and the lack of generalisation [76].

Given two measurable spaces $\mathcal{P}$ and $\mathcal{X}$, the final goal of DL models consists in approximating a certain function $f : \mathcal{P} \to \mathcal{X}$. The workhorse of DL are NNs, i. e. computational models that are composed of several processing layers to learn representations of data, with multiple levels of abstraction. Therefore, a NN approximates f with a parametric mapping $x = \phi(\boldsymbol{p}; \boldsymbol{\omega})$ by automatically performing a data feature extraction process, which discover *intrinsic* relations in large data sets, and using backpropagation algorithms [155] to determine how suitably change the internal parameters $\boldsymbol{\omega}$ to result in the best function approximation of f. More precisely, throughout this Thesis we will consider as DL models, only the ones which agree with the following definitions.

Definition 7 ([76])**.** *The term NN indicates a collection of functions that can be represented by Directed Acyclic Graphs (DAGs), where the vertices correspond to elementary almost everywhere differentiable parametric functions and the edges symbolize compositions of these functions. The vertices of these DAGs are usually called* nodes *or* units*.*

Definition 8 ([5])**.** *DL refers to techniques where deep NNs are used to define the model and their parameters are set with gradient-based methods.*

This latter definition synthesize the two essential cores of DL technologies, i. e. deep NNs and gradient-based optimization, which will be both subsequently addressed in this Chapter.

Different NN architectures determine different DL models, where the word *architecture* refers to the overall structure of the network: how many units it has and how these units are connected to each other. The amount of different possible NN architecture topologies is nowadays incredible, thereby in this chapter we provide insights of the first architecture ever developed, i. e. MLP and for then focusing on CNNs and GCNs, employed later on in this Thesis.

2.2.1 *Multi Layer Perceptron*

The most basic deep NN is the MLP, which has its foundations in the Perceptron [135, 154], i. e. an algorithm developed to solve the binary classification problem. More precisely, the problem to be solved can be formalised as in the following definition.

Definition 9. *Let $\mathcal{P}, \mathcal{X}$ be two measurable spaces and let*

$$\{(\boldsymbol{P}_i, x_i) \in \mathcal{P} \times \mathcal{X} \ : \ i = 1, \dots, m\}$$

be a dataset consisting of input features $\boldsymbol{P}_i \in \mathcal{P}$ *and corresponding labels* $x_i \in \mathcal{X}$. *A binary-classification task consists in finding a model* $f : \mathcal{P} \to \mathcal{X}$, *with* $\mathcal{X} = \{-1, +1\}$, *so that* $f(\boldsymbol{P}_i)$ *is a good prediction of the true label* x_i, *for* $i = 1, \ldots, m$.

The Perceptron defines a two-class model, which maps a real-valued input $\boldsymbol{P}_i$ into $x_i \in \{-1, +1\}$, by using a non-linear *activation* function σ. Its architecture contains a single input layer and an output node, respectively in green and red in Figure 2.9. The input layer contains m input units that transmit the features $\boldsymbol{P}_i = [p_1, \ldots, p_m]$ with edges of weight $\boldsymbol{\omega} = [\omega_1, \ldots, \omega_m]$ and *bias* b, to the output unit x. More precisely, the linear function

$$\boldsymbol{\omega} \cdot \boldsymbol{P}_i := \sum_{j=1}^{m} \omega_j p_j + b \tag{2.16}$$

is computed at the output node and subsequently, the activation function σ is applied to this real value in order to predict the dependent variable of $\boldsymbol{P}_i$. Specifically, for the Perceptron model, σ corresponds to the function sign : $\mathbb{R} \to \{-1, +1\}$, which maps a real value z to either $+1$ or -1, i. e.

$$\operatorname{sign}(z) = \begin{cases} +1 \text{ if } z \geq 0, \\ -1 \text{ if } z < 0, \end{cases}$$

hence resulting naturally appropriate for binary classification. Finally, the prediction x is computed as

$$x = \sigma\left(\sum_{j=1}^{m} \omega_j p_j + b\right) = \operatorname{sign}\left(\sum_{j=1}^{m} \omega_j p_j + b\right). \tag{2.17}$$

The error of the prediction is consequently a value drawn from the set $\{-2, 0 + 2\}$. In the case the error is non-zero, the weight values $\boldsymbol{\omega}$ and the bias b of the Perceptron need to be updated along the (negative) direction of the error gradient, to recover and avoid misclassification.

Putting together more Perceptron architectures and different activation functions results in a MLP model. For architectures with *multiple layers*, these are classified as *input*, *hidden* and *output* layers, depending on the topology of their connections. More precisely,

- *Input layer*: in this case, each unit corresponds to a feature of the input sample; hence, there are as many units as the number of features of the considered data. Moreover, the direct graph connections are only in output, from the units of the input layers towards the units of the following one.

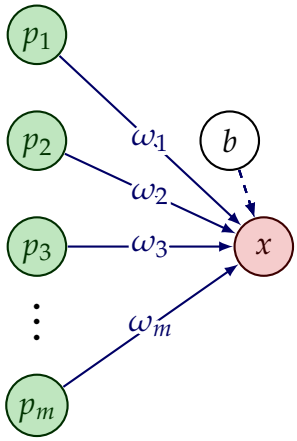


Figure 2.9: Example of a Perceptron with m input units and 1 output unit.

- *Hidden layers*: these layers are placed between the input and the output layers. Their number depends on the chosen architecture, and their amount of units is unconstrained and can vary within the layers. The number of hidden layers and hidden units per layer plays a role in the expressiveness of the model. Finally, any hidden unit can have connections in input and output, respectively, with the units of the previous or following layer.
- *Output layer*: the number of units of the output layers depends strictly on the function at hand, which the NN is required to approximate. In this case, there is no next layer, and the output unit connections are only along the incoming direction.

Figure 2.10 shows an MLP with 5 layers, i. e. an input layer with m units (green), 3 hidden layers with m_1, m_2, m_3 units per layer (blue) and 1 output layer with n units (red). In particular, for an MLP architecture, at any hidden layer, each unit receives the same real value from every unit of the previous layer and produces a real value that is passed in output to every unit of the following layer. Due to their edge topology, MLP architectures are also addressed as Fully Connected NN.

Definition 10 ([5]). *Let $L \in \mathbb{N}_{\geq 1}$, where $L+1$ is the total number of layers of a MLP architecture. For each $\ell = 0, \ldots, L$, let m_ℓ be the number of hidden units for the ℓ-th layer and let*

$$\sigma_\ell : \mathbb{R} \to \mathbb{R}$$

be an activation function. Moreover, let

$$W = \left\{ W^{(\ell)} \right\}_{\ell=1}^{L} \quad \textit{and} \quad b = \left\{ \boldsymbol{b}^{(\ell)} \right\}_{\ell=1}^{L},$$

where

$$W^{(\ell)} = \left(w_{ij}^{(\ell)} \right)_{i=1,j=1}^{m_\ell, m_{\ell-1}} \in \mathbb{R}^{m_\ell \times m_{\ell-1}} \quad \textit{and} \quad \boldsymbol{b}^{(\ell)} \in \mathbb{R}^{m_\ell}, \quad \textit{for} \quad \ell = 1, \dots, L$$

are the weight matrices, and the bias vectors of the architecture, respectively. More precisely, $w_{ij}^{(\ell)}$ is the weight associated with the edge between the i-th unit of the ℓ-th layer and the j-th unit of the $\ell - 1$ layer, whereas $b_i^{(\ell)}$ is the bias associated with the i-th unit of the ℓ-th layer, for $\ell = 1, \dots, L-1$.

Finally, let #LP the total number of learnable parameters of the architecture. The MLP realization function $\Phi : \mathbb{R}^{m_0} \times \mathbb{R}^{\#LP} \to \mathbb{R}^{m_L}$ can be defined as

$$\Phi\left(p, \{W, b\}\right) = \phi^{(L)}\left(p, \{W, b\}\right),$$

where

$$\begin{aligned}
\phi^{(1)}\left(p, \{W, b\}\right) &= W^{(1)} p + b^{(1)}, \\
\hat{\phi}^{(\ell)}\left(p, \{W, b\}\right) &= \sigma_\ell\left(\phi^{(\ell)}\left(p, \{W, b\}\right)\right) \quad \textit{for} \quad \ell = 1, \dots, L-1, \\
\phi^{(\ell+1)}\left(p, \{W, b\}\right) &= W^{(\ell+1)} \hat{\phi}^{(\ell)}\left(p, \{W, b\}\right) + \boldsymbol{b}^{(\ell+1)} \quad \textit{for} \quad \ell = 1, \dots, L-1.
\end{aligned}$$

The underlying directed acyclic graph of any MLP model is given by the composition of the affine linear maps $\phi| \to W^{(\ell)}\phi + \boldsymbol{b}^{(\ell)}$ $\ell = 0, \dots, L$, with the activation function σ_ℓ intertwined, for $\ell = 1, \dots, L-1$. An example of an MLP realization at general layer $\ell + 1$ is illustrated in Figure 2.11.

The choice of activation functions is a critical part of neural network design, and it depends on the problem at hand. In general, they are needed to be non-linear functions in order to avoid the trivial situation in which the output is only a linear combination of the input data [76]. In addition, their concurrence allows to represent arbitrarily complex functions [1]. A selection of common activation functions is presented in Figure 2.12.

Remark 9. *The Rectified Linear Unit activation function (ReLU) activation function is commonly used for its simple piecewise linear structure and high performance [187].*

The overall number of layers $L + 1$ gives the *depth* of the model. The name "deep learning" arose from this terminology. By adding more layers and more

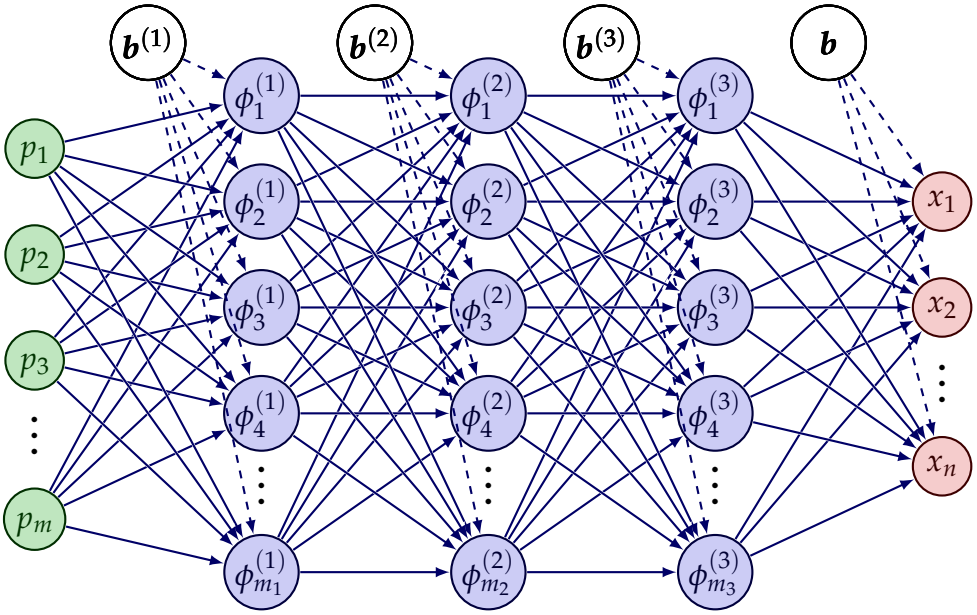
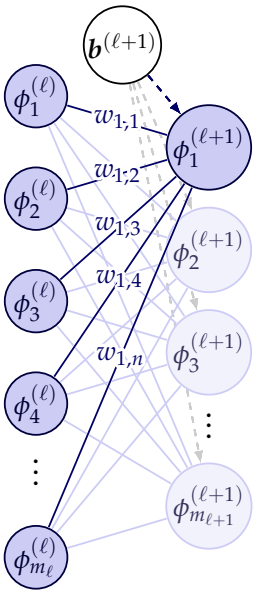


Figure 2.10: Example of an MLP with 5-layer Perceptron, m input units, m_ℓ hidden units for each ℓ^{th} hidden layer and n output units.



$$\underbrace{\begin{pmatrix} a_1^{(\ell+1)} \\ a_2^{(\ell+1)} \\ \vdots \\ a_{m_{\ell+1}}^{(\ell+1)} \end{pmatrix}}_{\hat{\phi}^{(\ell+1)}} = \sigma_\ell \left[\begin{pmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,m_\ell} \\ w_{2,1} & w_{2,2} & \dots & w_{2,m_\ell} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m_{\ell+1},1} & w_{m_{\ell+1},2} & \dots & w_{m_{\ell+1},m_\ell} \end{pmatrix} \underbrace{\begin{pmatrix} \hat{\phi}_1^{(\ell)} \\ \hat{\phi}_2^{(\ell)} \\ \vdots \\ \hat{\phi}_{m_\ell}^{(\ell)} \end{pmatrix}}_{\hat{\phi}^{(\ell)}} + \begin{pmatrix} b_1^{(\ell+1)} \\ b_2^{(\ell+1)} \\ \vdots \\ b_{m_{\ell+1}}^{(\ell+1)} \end{pmatrix} \right]$$

$$\hat{\phi}^{(\ell+1)} = \sigma_\ell \left(W^{(\ell+1)} \hat{\phi}^{(\ell)} + \boldsymbol{b}^{(\ell+1)} \right), \quad \text{for} \quad \ell = 1, \dots, L-1.$$

Figure 2.11: Example of MLP architecture realization, according to Definition 10.

units within a layer, a deep NN can represent functions of increasing complexity, but of course the computational costs increase. It is fundamental to find a proper

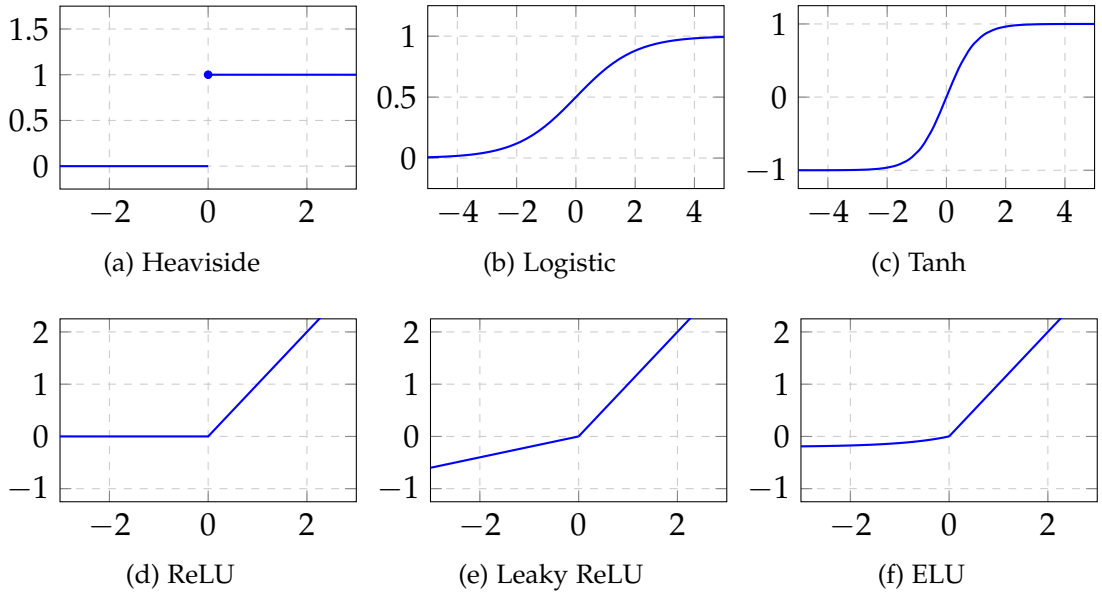


Figure 2.12: Some common activation functions, $\sigma(x)$ for $x \in \mathbb{R}$.

architecture that is a trade-off between computational training costs and the accuracy that can be achieved.

Remark 10. *With the notation introduced in Definition 10, the total number of learnable parameters #LP of a MLP architecture can be computed as,*

$$\#LP := \sum_{\ell=1}^{L} m_\ell \left(m_{\ell-1} + 1\right).$$

2.2.2 *Convolutional Neural Networks*

The development of CNN arose with the application of deep NN to image processing [109, 110]. More precisely, fully-connected networks ignore a key property of images, which is that nearby pixels are more strongly correlated than the distant ones. Many of the modern approaches to computer vision exploit this property by extracting *local* features that depend only on small subregions of the image. Subsequently, the information extracted from such features can then be merged into later processing stages.

In order to achieve more sparse learning models, the matrix multiplication operator, which characterized fully-connected architectures outlined in Defi-

nition 10, has been substituted by a matrix convolution operator [76]. More precisely,

Definition 11 ([76]). *CNNs are NNs that use convolution in place of general matrix multiplication in at least one of their layer.*

The general formulation of two matrix convolution $A, W \in \mathbb{R}^{m \times n}$ is

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mm} \end{pmatrix} \star \begin{pmatrix} \omega_{11} & \omega_{12} & \dots & \omega_{1n} \\ \omega_{21} & \omega_{22} & \dots & \omega_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{m1} & \omega_{m2} & \dots & \omega_{mm} \end{pmatrix} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{m-i,n-j} \omega_{1+i,1+j}. \tag{2.18}$$

Figure 2.13 (a) shows an example of $1D$ (univariate) convolution, whereas Figure 2.13 (b) shows and example of $2D$ (bivariate) convolution. In both cases, the input is processed by a filter through a matrix convolution operator, according to (2.18), which generates the output feature map, usually given as input to the successive convolutional operator. In particular, each output item is given by the sum of the element-wise product between the different *local* values of the input (green numbers in Figure 2.13) and the corresponding values of the filter (red numbers in Figure 2.13). Subsequently, the filter matrix slides on the input matrix, from left to right in the picture, until the input elements have all been completely scanned.

Thanks to the nature of convolutional operators, CNNs are characterized by three unique properties: sparse connectivity, weight sharing, and shift equivariant representations [7, 76]. Sparse connectivity means that the networks have *local* receptive fields, which collect information jointly from spatially neighbouring inputs. Weight sharing is achieved by requiring the receptive field to assume the same values on different instances of the input, and, consequently, the same parameters are involved to compute each output unit. Finally, shift equivariant representations follow the definition of the convolutional operations and the previous two properties. The locality and weight sharing properties for the one ($1D$) and the two dimensional ($2D$) convolutional operators can be seen again in Figure 2.13.

Finally, Figure 2.14 shows an example of CNN. More precisely, it represents a deep NN characterized by 1 input layer (green), 6 hidden layers, i. e. 4 *convolutional* layers (orange), and 2 fully connected layer (blue), and 1 output layer

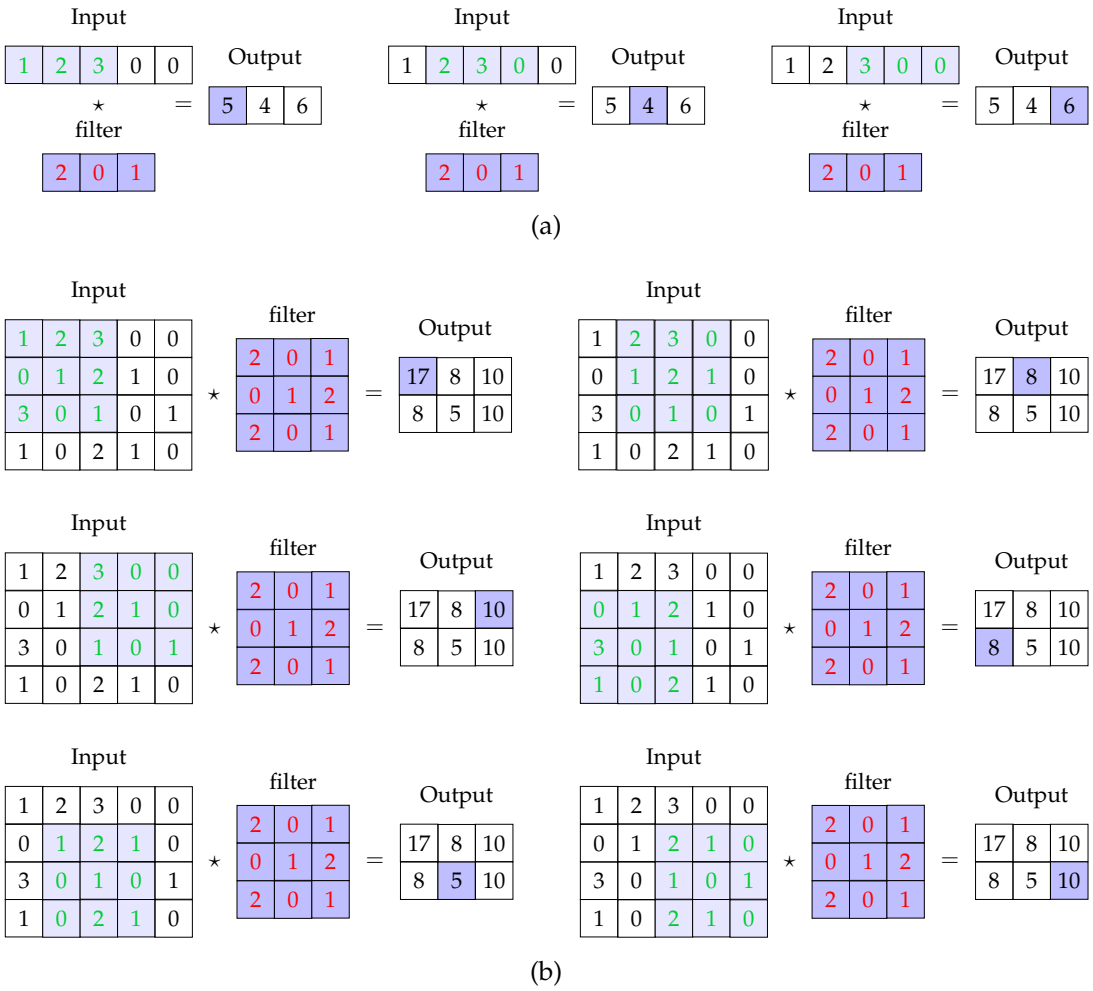


Figure 2.13: Examples of convolutional operators for $1D$ (a) and $2D$ (b).

(red). Note that the connections within the convolutional layers are less than the connections between the fully connected layers.

CNNs have been tremendously successful in practical applications for processing data with a known *grid-like* topology [76, 104, 167, 180]. Examples include time-series data, which can be thought of as a $1D$ grid taking samples at regular time intervals, and image data, which can be thought of as a $2D$ grid of pixels.

2.2.3 *Graph Convolutional Neural networks*

In many applications, the data does not exhibit a grid-like structure but may be characterized by a graph or mesh structure or be completely unorganized, as in

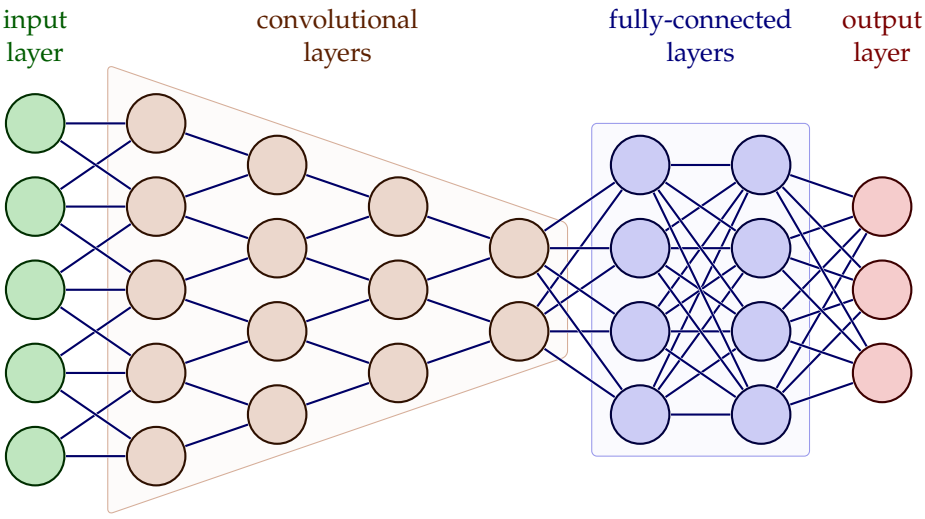


Figure 2.14: Example of a CNN.

the case of scattered data. In order to handle this kind of data, a new learning theory and related architectures have been developed.

Geometric deep learning refers to the generalization of convolutional neural networks to non-Euclidean data such as discrete manifolds, graphs and general point clouds [21]. While for data represented on a regular grid in a Euclidean space the convolution operator is uniquely defined for arbitrary dimensions, in the case of unstructured non-Euclidean data many different approaches for the definition of operators that mimic the behaviour of standard convolution have been proposed [186].

More precisely, GCN are the CNN counterparts for non-gridded-like domains. There are two approaches to designing convolutional filters on graphs, namely the *spatial* and *spectral* methods, which result in the definition of *spatial*-GCNs and *spectral*-GCNs, see again [186].

Spatial methods define graph convolutions based on spatial relations between graph vertices in a similar fashion as standard CNNs. The updated output vertex is obtained by its convolution with certain neighbours by means of a filter kernel that transfers the information along the neighbourhood edges; see [77, 138] as examples of spatial GCNs. Even if the definition of a spatial graph convolutional operator seems to be a natural extension of standard convolutions, it suffers from the issue of matching local neighbourhoods and uniquely defining translations on graphs from a spatial perspective [23].

Spectral-based methods have their foundations in the spectral graph theory, which links the discrete setting to the continuous one [32]. In particular, the

convolution theorem defines convolutions as linear operators by exploiting the Fourier basis represented in terms of eigenvectors of the Laplace operator. With suitable choices of filter parameterization, the filters defined in the spectral domain are naturally localized, and the costs arising from computing the Fourier transform can be contained [42]. Additional examples of spectral GCNs are [23, 116], among others. A comprehensive survey of graph convolutional operators can be found in [186].

2.2.4 *Optimization*

We compute an objective function that measures the error (or distance) between the output and the desired target. The machine then modifies its internal, adjustable parameters to reduce this error. In a typical deep-learning system, there may be hundreds of millions of these adjustable weights and hundreds of millions of (labeled) examples with which to train the machine.

The second ingredient, gradient-based optimization, is made possible by the observation that, due to the graph-based structure of any neural network, see e. g., Definition 10, the gradient of an objective function with respect to the parameters of the neural network can be computed efficiently. This has been observed in various ways; see [49, 98, 155].

3

DATA FITTING SCHEMES WITH HIERARCHICAL SPLINES

Constructing continuous and flexible geometric models that are easy to shape and manipulate is a fundamental requirement in many applications. For example, they can serve for visualization and direct measurements within a design phase or a manufacturing process, as well as for simulations, i. e. performing numerical computations directly on the model or its components.

In this Chapter, we consider the problem of constructing a parametric (spline) model $\boldsymbol{s}\colon \Omega \to \mathbb{R}^N$ on a domain $\Omega \subseteq \mathbb{R}^D$ in any dimension $D \in \mathbb{N}_{>0}$ from pointwise data and parametric values

$$\mathcal{U} \times \mathcal{P} \coloneqq \left\{ (\boldsymbol{u}_i, \boldsymbol{p}_i) \in \mathbb{R}^D \times \mathbb{R}^N \,|\, i = 1, \dots, m \right\} \tag{3.1}$$

where,

$$\mathcal{P} \coloneqq \left\{ \boldsymbol{p}_i \in \mathbb{R}^N \,|\, i = 1, \dots, m \right\}$$

for $N \in \mathbb{N}_{>0}$ are the point observations as in (1.1), at the parametric sites

$$\mathcal{U} \coloneqq \left\{ \boldsymbol{u}_i \in \Omega \subset \mathbb{R}^D \,|\, i = 1, \dots, m \right\},$$

as in (1.3). In CAGD applications, if $N = 2$ the data points lay on a plane, whereas if $N = 3$ the data points belong to the physical space. Moreover, we require the spline model $\boldsymbol{s}$ to approximate the data $\mathcal{P}$ within a certain tolerance $\epsilon \in \mathbb{R}_{>0}$, in the sense that

$$\operatorname{dist}(\boldsymbol{s}_i, \boldsymbol{p}_i) \leq \epsilon \text{ for each } i = 1, \dots, m,$$

where $\boldsymbol{s}_i$ denotes a point on the spline model associated with the data point $\boldsymbol{p}_i$, and $\operatorname{dist}(\cdot, \cdot)$ is a certain distance metric.

By virtue of their properties, such as locality, non-negativity and partition of unity and their suitability for adaptive refinement, THB-splines are a desirable tool for building flexible geometric representations, see Section 2.1. Therefore, the THB-spline fitting problem can be stated in the following way, see also (1.2). Given a point cloud as in (3.1) and an error tolerance $\epsilon > 0$, find a THB-spline model $\boldsymbol{s} : \Omega \subseteq \mathbb{R}^D \to \mathbb{R}^N$, so that

$$\operatorname{dist}(\boldsymbol{s}(\boldsymbol{u}_i), \boldsymbol{p}_i) \leq \epsilon, \quad \text{with } \boldsymbol{u}_i \in \Omega, \quad \text{for each} \quad i = 1, \dots, m. \tag{3.2}$$

The design of a THB-spline approximation depends on the identification of the hierarchical spline space $\mathcal{H}$, defined in Definition 4, the multivariate degree $\boldsymbol{d} \in \mathbb{N}^D_{>0}$, the hierarchical mesh, which defines a tessellation $T(\Omega)$, see also Remark 8, and finally also the computation of the control points.

In this Chapter, we exploit two main approximation methods to define the THB-spline model, i. e. the *reWeighted Least Squares (rWLS)* approximation and the *Quasi-Interpolation (QI)* schemes. In particular, we first focus in Section 3.1 on two very well established classical methods: interpolation and weighted least squares, and extend the latter as an adaptive rWLS fitting scheme. Subsequently, in Section 3.2, we present a hierarchical QI scheme with THB-splines. The present Chapter is mostly based on [16, 67].

3.1 INTERPOLATION AND WEIGHTED LEAST SQUARES

We here focus on two established classical methods: interpolation and weighted least squares; see e. g., [39]. Even though interpolation is well established, there is still ongoing research in this field, including the development of efficient algorithms with irregularly-spaced data, see e. g., [25, 26], among others. Weighted least squares methods [168] can be considered a more general instance of ordinary least squares methods, studied on weakly admissible meshes in [15]. The key difference between weighted and ordinary least squares is that in the former, fixed weight values are associated with the observations to incorporate different weightings in the least squares scheme.

Most of the time, interpolation and least squares methods are regarded as complementary techniques; however, despite this perception they share a strong connection in their polynomial formulation, as noted in [27]. Additionally, as detailed in [40, 45] they can also be used in combination to defeat the Runge phenomenon [156]. In this Section, we propose a general formulation of the weighted least squares approximant as a convex combination of suitable interpolants, for any finite-dimensional function space consisting of real-valued functions defined on a domain $\Omega \subseteq \mathbb{R}^D$ and address its consequences. As a special case, our formulation includes the vector space of polynomials up to a certain degree, for which a relation between weighted polynomial least squares and interpolation has been discussed in [27].

In addition, we also focus on the choice of the weights. Our main aim is to update these weights in the spirit of the Iterative Reweighted Least Squares (IRLS) method [8, 142], with convergence guarantees and efficient algorithms [3, 185]. IRLS also offers robust regression [88] and is used to smooth the reconstruction

and limit the influence of outliers in the input data. However, the main difference between our approach to the IRLS method is that our updates allow us to preserve the sharp features of the final model and are also suitable for adaptive approximations. Our numerical experiments show the performance of the proposed method within the spline framework, from curve to surface fitting with an appropriate underlying mesh, including adaptive spline constructions.

3.1.1 *Interpolation*

Throughout this Section, we assume that $\mathbb{R}^N$, $\mathbb{R}^D$, $\mathbb{R}^n$ and $\mathbb{R}^m$ are column vectors. Let $\Omega \subseteq \mathbb{R}^D$ and consider a set of given observations as in (3.1). Moreover, let V be a vector space of functions defined on Ω and taking values in $\mathbb{R}^D$. We denote its finite dimension by $n+1 = \dim V$ and assume that V is generated by a basis $\mathcal{B} = \{\beta_0, \ldots, \beta_n\}$, with basis functions $\beta_j : \Omega \to \mathbb{R}$, i. e. $V = \operatorname{span}\{\mathcal{B}\}$.

The *interpolation problem* aims to find an element $\boldsymbol{s} \in V$ such that $\boldsymbol{s}(\boldsymbol{u}_i) = \boldsymbol{p}_i$ holds for each $i = 1, \ldots, m$. If a solution $\boldsymbol{s}$ exists, it can be expressed as

$$\boldsymbol{s}(\boldsymbol{x}) = \sum_{j=0}^{n} \boldsymbol{c}_j \beta_j, \text{ for } \boldsymbol{x} \in \Omega, \tag{3.3}$$

where the coefficients $\boldsymbol{c}_j = (c_j^1, \ldots, c_j^N) \in \mathbb{R}^N$ can be determined by solving a linear system for each component $k = 1, \ldots, N$, of the form $B\boldsymbol{c}^k = \boldsymbol{p}^k$, where

$$B = B(\mathcal{U}) = \begin{pmatrix} \beta_0(\boldsymbol{u}_1) & \ldots & \beta_n(\boldsymbol{u}_1) \\ \vdots & \ddots & \vdots \\ \beta_0(\boldsymbol{u}_m) & \ldots & \beta_n(\boldsymbol{u}_m) \end{pmatrix} \in \mathbb{R}^{m \times (n+1)} \tag{3.4}$$

is the $m \times (n+1)$ collocation matrix defined by the basis $\mathcal{B}$ and the parametric sites $\mathcal{U}$,

$$\boldsymbol{c}^k = (c_0^k, \ldots, c_n^k)^\top \in \mathbb{R}^{n+1} \text{ and } \boldsymbol{p}^k = (p_1^k, \ldots, p_m^k)^\top \in \mathbb{R}^m.$$

The solution $\boldsymbol{s} \in V$ is unique, if and only if B is invertible, hence $n+1 = m$ is a necessary condition. However, even if the interpolant $\boldsymbol{s} \in V$ exists, it is well known that it can be affected by poor approximation quality. Several factors can contribute to this, such as the nature of the data (e. g., their quantity or distribution) and the intrinsic properties of the function space V (e. g., the presence of oscillatory behaviour [156]).

3.1.2 *Weighted least squares*

As an alternative to interpolation, a common approach is to define the approximant $\boldsymbol{s} \in V$ as the solution to a *weighted least squares problem* when $n+1 \leq m$. In this context, we assign a strictly positive *weight* value $\omega_i \in \mathbb{R}_{>0}$ to each $\boldsymbol{u}_i \in \Omega$ for $i = 1, \ldots, m$. Let $W \in \mathbb{R}^{m \times m}$ be the associated diagonal matrix with the i-th diagonal entry given by ω_i. Then, the weighted least squares solution $\boldsymbol{s} \in V$ is obtained by solving the minimization problem

$$\min_{\boldsymbol{v} \in V} \frac{1}{2} \sum_{i=1}^{m} \omega_i \left\| \boldsymbol{v}(\boldsymbol{u}_i) - \boldsymbol{p}_i \right\|_2^2. \tag{3.5}$$

By expanding the solution $\boldsymbol{s} \in V$ of (3.5) as (3.3) and letting $\boldsymbol{c} = (\boldsymbol{c}_0, \ldots, \boldsymbol{c}_n)^\top \in \mathbb{R}^{(n+1) \times D}$ be the coefficient matrix, we can re-write problem (3.5) as

$$\min_{\boldsymbol{c} \in \mathbb{R}^{(n+1) \times N}} \| W^{\frac{1}{2}} B \boldsymbol{c} - W^{\frac{1}{2}} \boldsymbol{p} \|_2^2, \tag{3.6}$$

where, $\boldsymbol{p} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_m)^\top \in \mathbb{R}^{m \times N}$,

$$W = \operatorname{diag}\{\omega_1, \ldots, \omega_m\} = \begin{pmatrix} \omega_1 & 0 & \ldots & 0 \\ 0 & \omega_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \omega_m \end{pmatrix} \in \mathbb{R}^{m \times m}$$

and $W^{\frac{1}{2}}$ denotes the element-wise evaluation of the square root of W. The minimization in (3.6) entails solving a system of linear equations for each column $\boldsymbol{c}^k = (c_0^k, \ldots, c_n^k)$, $k = 1, \ldots, N$, of $\boldsymbol{c}$. More precisely, for each $k = 1, \ldots, N$, $\boldsymbol{c}^k$ is obtained from the normal equation

$$B^\top W B \boldsymbol{c}^k = B^\top W \boldsymbol{f}^k. \tag{3.7}$$

3.1.3 *Weighted least squares via interpolation*

In this part, to alleviate the notation, we will assume that $N = 1$. Therefore, we avoid the superscript k introduced above and reduce the bold writing according to the dimension. Nevertheless, all the results apply in the more general case of $N \geq 1$. Consider $\Omega \subseteq \mathbb{R}^D$,

$$\mathcal{U} \times \mathcal{P} = \left\{ (\boldsymbol{u}_i, p_i) \in \mathbb{R}^D \times \mathbb{R} \,|\, i = 1, \ldots, m \right\}$$

a set of given observations with $\boldsymbol{u}_i \in \Omega$ and $p_i \in \mathbb{R}$ and finally $V = \operatorname{span}\{\mathcal{B}\}$ a vector space of dimension $n+1$, with basis $\mathcal{B} = \{\beta_0, \dots, \beta_n\}$ and functions $\beta_j : \Omega \to \mathbb{R}$, for $j = 0, \dots, n$.

For any set $K \subset \mathbb{N}$ we denote by $\#(K)$ the cardinality of K and we define

$$\mathcal{K}_{n+1} = \{K \subseteq \{1, \dots, m\} : \#(K) = n+1\},$$

the set of all subsets of $\{1, \dots, m\}$ with cardinality $n+1$, equal to the dimension of the vector space V. For each $K \in \mathcal{K}_{n+1}$, there exists $k_i \in \{1, \dots, m\}$ for $i = 0, \dots, n$, such that $K = \{k_0, \dots, k_n\}$. We denote by $s_K \in V$ the interpolant of p_{k_i} at points $\boldsymbol{u}_{k_i}$ for $i = 0, \dots, n$. For such K, we define

$$B_K = \begin{pmatrix} \beta_0(\boldsymbol{u}_{k_0}) & \dots & \beta_n(\boldsymbol{u}_{k_0}) \\ \vdots & \ddots & \vdots \\ \beta_0(\boldsymbol{u}_{k_n}) & \dots & \beta_n(\boldsymbol{u}_{k_n}) \end{pmatrix} \in \mathbb{R}^{(n+1)\times(n+1)} \tag{3.8}$$

which is called the *collocation matrix* at the points $\boldsymbol{u}_{k_i}$ with respect to the whole basis $\mathcal{B}$ of V. For $K \in \mathcal{K}_{n+1}$ and positive weights $\omega_1, \dots, \omega_m \in \mathbb{R}$, we write

$$\omega_K = \prod_{i \in K} \omega_i. \tag{3.9}$$

Following the notation in [27], we define $\lambda_K = \omega_K \left|B_K\right|^2$, where $\left|B_K\right|$ denotes the determinant of B_K. Finally, we define $\mathcal{K}^\star_{n+1} \subset \mathcal{K}_{n+1}$ as the set of all subsets of $\{1, \dots, m\}$ of cardinality $n+1$ with $|B_K| \neq 0$. In other words,

$$\mathcal{K}^\star_{n+1} = \{K \subseteq \{1, \dots, m\} : \#(K) = n+1, \text{ and } |B_K| \neq 0\}.$$

Theorem 1. *The weighted least squares approximant $s \in V$ of the set of points $\left\{(\boldsymbol{u}_i, p_i)\right\}_{i=1}^m$ is the weighted sum of the interpolants $s_K \in V$ for $K \in \mathcal{K}^\star_{n+1}$ which interpolate the points $(\boldsymbol{u}_i, p_i)$ for $i \in K$, i.e.*

$$s(\boldsymbol{x}) = \frac{\sum_{K \in \mathcal{K}^\star_{n+1}} \lambda_K s_K(\boldsymbol{x})}{\sum_{K \in \mathcal{K}^\star_{n+1}} \lambda_K}, \quad \forall \boldsymbol{x} \in \mathbb{R}^D. \tag{3.10}$$

Proof. By hypothesis, $s \in V$ minimizes (3.5). Let $\mathcal{B}$ a basis for V and write s as in (3.3), for some coefficients $c_j \in \mathbb{R}$, $j = 0, \dots, n$. Hence, we set

$$\boldsymbol{c} = (c_0, \dots, c_n)^\top \quad \text{and} \quad \boldsymbol{p} = (p_1, \dots, p_m)^\top,$$

and write the normal equation (3.7) as $A\boldsymbol{c} = \boldsymbol{b}$ with

$$A = B^\top W B = \begin{pmatrix} \omega_1\beta_0(\boldsymbol{u}_1) & \dots & \omega_m\beta_0(\boldsymbol{u}_m) \\ \vdots & \ddots & \vdots \\ \omega_1\beta_n(\boldsymbol{u}_1) & \dots & \omega_m\beta_n(\boldsymbol{u}_m) \end{pmatrix} \begin{pmatrix} \beta_0(\boldsymbol{u}_1) & \dots & \beta_n(\boldsymbol{u}_1) \\ \vdots & \ddots & \vdots \\ \beta_0(\boldsymbol{u}_m) & \dots & \beta_n(\boldsymbol{u}_m) \end{pmatrix},$$
$$\boldsymbol{b} = B^\top W \boldsymbol{p} = \left(\textstyle\sum_{i=1}^m \omega_i\beta_0(\boldsymbol{u}_i)p_i, \;\; \dots, \;\; \sum_{i=1}^m \omega_i\beta_n(\boldsymbol{u}_i)p_i \right)^\top.$$

By the Cauchy-Binet theorem [20, Section 4.6],

$$|A| = \sum_{K \in \mathcal{K}_{n+1}} \omega_K \, |B_K|^2$$

and by Cramer's rule, we obtain

$$c_j = \big|A_j\big| \Big/ |A| \,,$$

for each $j = 0, \dots, n$, where A_j is obtained from A replacing its j-th column with $\boldsymbol{b}$ for $j = 0, \dots, n$, namely, $A_j = B^\top W B_j$ with

$$B_j = \begin{pmatrix} \beta_0(\boldsymbol{u}_1) & \dots & \beta_{j-1}(\boldsymbol{u}_1) & p_1 & \beta_{j+1}(\boldsymbol{u}_1) & \dots & \beta_n(\boldsymbol{u}_1) \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta_0(\boldsymbol{u}_m) & \dots & \beta_{j-1}(\boldsymbol{u}_m) & p_m & \beta_{j+1}(\boldsymbol{u}_m) & \dots & \beta_n(\boldsymbol{u}_m) \end{pmatrix} \in \mathbb{R}^{m \times (n+1)}.$$

Again from the Cauchy-Binet theorem, we obtain

$$\big|A_j\big| = \sum_{K \in \mathcal{K}_{n+1}} \omega_K \, |B_K| \, \big|B_{j,K}\big|$$

for $j = 0, \dots, n$, where $B_{j,K}$ is obtained from B_K by replacing its j-th column with $(p_1, \dots, p_m)^\top$ for $j = 0, \dots, n$. If $K \in \mathcal{K}^\star_{n+1}$, the interpolation conditions satisfy

$$B_K \boldsymbol{c}_K = \boldsymbol{p}_K,$$

where $\boldsymbol{p}_K = (p_{k_1}, \dots, p_{k_n})^\top$ with $K = \{k_0, \dots, k_n\}$. The solution of the linear problem $\boldsymbol{c}_K = (c_{0,K}, \dots, c_{n,K})^\top$ is unique if $|B_K| \neq 0$, and can be obtained by Cramer's rule as follows,

$$c_{j,K} = \big|B_{j,K}\big| \Big/ |B_K| \,.$$

The associated interpolant $s_K(\boldsymbol{x})$ for $K \in \mathcal{K}^\star_{n+1}$ can be written as

$$s_K(\boldsymbol{x}) = \sum_{j=0}^{n} c_{j,K}\beta_j(\boldsymbol{x}),$$

hence by letting $\lambda_K = \omega_K |B_K|^2$, it holds

$$\begin{aligned} s(\boldsymbol{x}) = \sum_{j=0}^{n} c_j\beta_j(\boldsymbol{x}) = \sum_{j=0}^{n} \frac{|A_j|\,\beta_j(\boldsymbol{x})}{|A|} = \sum_{j=0}^{n} \frac{\sum_{K\in\mathcal{K}_{n+1}} \omega_K |B_K| \left|B_{j,K}\right| \beta_j(\boldsymbol{x})}{\sum_{K\in\mathcal{K}_{n+1}} \omega_K |B_K|^2} \\ = \frac{\sum_{K\in\mathcal{P}^\star_n} \omega_K |B_K|^2 \sum_{j=1}^{n} c_{j,K}\beta_j(\boldsymbol{x})}{\sum_{K\in\mathcal{P}^\star_n} \omega_K |B_K|^2} = \frac{\sum_{K\in\mathcal{P}^\star_n} \lambda_K s_K(\boldsymbol{x})}{\sum_{K\in\mathcal{P}^\star_n} \lambda_K}. \end{aligned}$$

□

Remark 11. *Note that Theorem 1 is applicable to any finite-dimensional (multivariate) vector space. This encompasses various function spaces, such as the space of polynomials up to a specific degree, spline spaces with fixed degree and order, spline spaces with varying locations of knot lines, or any other real function space of finite dimension equipped with a basis. Moreover, note that equation (3.10) in Theorem 1 is very similar to the formulation of multinode Shepard operators [44] in the case of polynomial spaces V.*

Remark 12. *In general, we have $\#\left(\mathcal{K}^\star_{n+1}\right) \leq \#\left(\mathcal{K}_{n+1}\right) = \binom{m}{n+1}$, due to the non-nullity condition on the determinants $|B_K|$, which guarantees the existence of the interpolants. In particular, in the case of splines, the condition $|B_K| \neq 0$ is equivalent to the Schoenberg-Whitney nesting condition [12]. In addition, the value of λ_K depends on the location of the knot lines.*

3.1.4 *Remarks on weighted least squares approximation*

For the univariate polynomial case, results on the upper and lower pointwise error bounds of the approximant derivatives up to a certain order as well as on the influence of the weights have been presented in [27, Section 3] and [27, Section 4], respectively. We extend these results to any vector space V of finite dimension $n+1$. This generalization extends the derived conclusions of [27] in a broader setting, beyond the polynomial scenario.

Let $r \in \mathbb{N}$ be given, if the r-th order derivative of $s \in V$ exists, it can be expressed as the weighted average of the r-th order derivatives of the interpolants, i. e.

$$\partial^\alpha s(\boldsymbol{x}) = \frac{\sum_{K \in \mathcal{K}^\star_{n+1}} \lambda_K \partial^\alpha s_K(\boldsymbol{x})}{\sum_{K \in \mathcal{K}^\star_{n+1}} \lambda_K}, \tag{3.11}$$

where $\alpha = (\alpha_1, \ldots, \alpha_D)$ is a suitable multi-index, with $\sum_{j=1}^{D} \alpha_j = r \geq 0$.

In addition, pointwise upper and lower bounds for the value of the r-th order derivative of $s(\boldsymbol{x})$ can be obtained from (3.11), i. e.

$$\min_{K \in \mathcal{K}^\star_{n+1}} \partial^\alpha s_K(\boldsymbol{x}) \leq \partial^\alpha s(\boldsymbol{x}) \leq \max_{K \in \mathcal{K}^\star_{n+1}} \partial^\alpha s_K(\boldsymbol{x}).$$

Likewise, the pointwise approximation error shares the same weighted average, given by

$$f(\boldsymbol{x}) - s(\boldsymbol{x}) = \frac{\sum_{K \in \mathcal{K}^\star_{n+1}} \lambda_K \left(f(\boldsymbol{x}) - s_K(\boldsymbol{x}) \right)}{\sum_{K \in \mathcal{K}^\star_{n+1}} \lambda_K}.$$

Furthermore, the following consequences on the influence of the weights ω_i for $i = 1, \ldots, m$ and derivative estimations can be inferred.

Remark 13. *Let $I \subseteq \{1, \ldots, m\}$ be of cardinality $\#(I) = r$, with $1 \leq r \leq n+1$. Define*

$$s_I(\boldsymbol{x}) = \lim_{\substack{\omega_i \to +\infty \\ \forall i \in I}} s(\boldsymbol{x})$$

and

$$\mathcal{K}^\star_{n+1} \setminus I = \{ K \subseteq \{1, \ldots, m\} \setminus I \mid \#(K) = n+1, \text{ and } |B_K| \neq 0 \}.$$

If $r = n+1$, then $s_I(\boldsymbol{x}) = s_I(\boldsymbol{x})$, represents the interpolant of the data points indexed by I. If $1 \leq r < n+1$, then

$$s_I(\boldsymbol{x}) = \frac{\sum_{K \in \mathcal{K}^\star_{n+1-r} \setminus \{I\}} \lambda_{I,K} s_{I \cup K}(\boldsymbol{x})}{\sum_{K \in \mathcal{K}^\star_{n+1-r} \setminus \{I\}} \lambda_{I,K}}, \quad \text{with} \quad \lambda_{I,K} = \omega_K \left| B_{I \cup K} \right|^2.$$

The proof can be obtained by considering $V = \operatorname{span}\{\mathcal{B}\}$, where $\mathcal{B} = \{\beta_0, \ldots, \beta_n\}$ with $\beta_j : \Omega \to \mathbb{R}$ for $j = 0, \ldots, n$, and substituting the Vandermonde matrices in [27] with the corresponding collocation matrices B in (3.4) and B_K in (3.8).

Another important implication of Theorem 1 is the possibility of rewriting ℓ^p–approximation problems as suitable convex combination of interpolants as outlined in Remark 14 where we exploit the IRLS method, see e. g., [8, Section 4.5].

Remark 14. *Let us consider the problem*

$$\min_{v \in V} \frac{1}{2} \sum_{i=1}^{m} \|v(\boldsymbol{x}_i) - f_i\|_p^p \tag{3.12}$$

with $1 < p < 2$*, where* $V = \operatorname{span}\{\mathcal{B}\}$ *is a vector space of dimension* $n+1$*, and* $\mathcal{B} = \{\beta_0, \ldots, \beta_n\}$ *its basis. The* IRLS *method approximates the exact solution of problem* (3.12) *through the iterative computation of weighted least squares problems. In particular, the weights are recursively updated for a maximum number iterations* $K_{\max}$ *by*

$$\omega_i^{k+1} = \left| s^k(\boldsymbol{x}_i) - f_i \right|^{\frac{(p-2)}{2}},$$

with weights ω_i^k *and solution* s^k *of problem* (3.5)*, for iterations* $k = 1, \ldots, K_{\max}$*. By direct application of* (3.10) *in Theorem 1, the outcome of each iteration can be rewritten as a convex combination of interpolants. Note that the interpolants and the determinant of the matrices* B_K *do not need to be recomputed. In other words, it is enough to update the weights* ω_K *according to* (3.9)*, to compute the solution of problem* (3.12)*.*

3.1.5 *Weighted least squares with splines*

We present the numerical verification of Theorem 1 for two different vector spaces V: the space of polynomials and the space of univariate splines with a specific degree and regularity. We consider a set of $m = 7$ observations in the form (u_i, p_i), for $i = 1, \ldots, m$, given by

$$\{(-4.5, -2), (-3.5, 0), (-2.2, -1), (-1.2, 2.8), (0.8, 2.9), (2.2, 0.5), (4.0, -2)\}.$$

We define the weights ω_i for $i = 1, \ldots, m$ as uniformly distributed on $(0, 1)$. In the polynomial case, we choose the polynomial degree $d = 2$, which implies

$$\#(\mathcal{K}_{d+1}) = \binom{m}{d+1} = \binom{7}{3} = 35.$$

Thus, we have a total of 35 interpolation problems to be solved. Moving on to polynomial spline spaces, in addition to the degree, we consider the spline order $k = d + 1 = 3$ and set the knot vector $\boldsymbol{t} = [-5, -5, -5, -5/3, 5/3, 5, 5, 5]$ of length 8, implying that the spline space has dimension $n + 1 = 8 - 3 = 5 \leq m$. Since $n \leq m$, there are at most

$$\#(\mathcal{K}_{n+1}) = \binom{m}{n+1} = \binom{7}{5} = 21$$

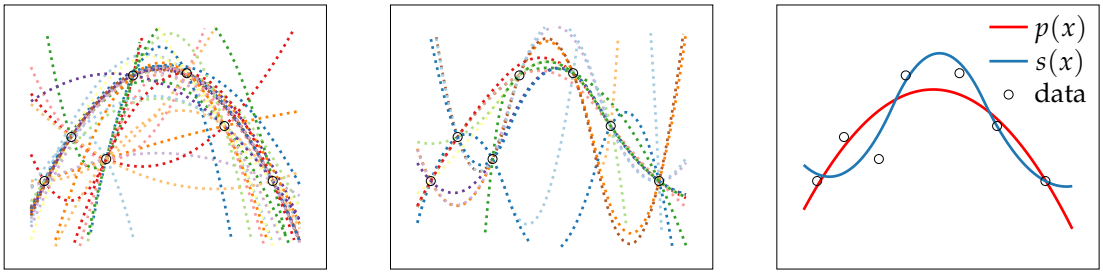


Figure 3.1: Numerical verification of (3.10), Theorem 1: $s_K(\boldsymbol{x})$ interpolants for polynomials (left) and splines (center) and the final weighted least squares approximations as sum of interpolants (right) for polynomials (red) and splines (blue).

interpolation problems required to fully reconstruct the spline least squares approximant. In the case of spline spaces, there is no guarantee that each point subsequence satisfies the Schoenberg-Whitney nesting conditions. To illustrate this, consider the data set identified by $K = \{1, 2, 3, 4, 5\} \subset \{1, \dots, m\}$, which does not satisfy these conditions. This is due to the absence of data points within the support of the last basis function, specifically $u_i \notin [5/3, 5]$ for $i = 1, \dots, 5$. Consequently, the interpolant s_K does not exist, and no interpolation problem needs to be solved for this particular subset. To fully reconstruct the final global least squares approximation in this example, we need to compute 20 interpolation problems instead of the original 21.

Figure 3.1 illustrates the interpolants and the global least squares approximation involved in (3.10) for the polynomial and spline spaces introduced for this numerical example. Note that we choose to show an example related to the univariate polynomial and spline space, but our results hold in general for any (multivariate) vector space of finite dimension.

3.1.6 *Weighted least squares with hierarchical splines*

In the following, we perform a numerical investigation of the role of the weights addressed in Remark 13, for a weighted least squares problem with hierarchical splines in the bivariate case.

We consider a point cloud of 64×64, i.e. $m = 4096$, uniformly gridded data, obtained by sampling the function

$$f(\boldsymbol{x}) = f(x, y) = \frac{2}{3\exp\left(\sqrt{(10x-3)^2 + (10y-3)^2}\right)} + \frac{2}{3\exp\left(\sqrt{(10x+3)^2 + (10y+)^2}\right)} + \frac{2}{3\exp\left(\sqrt{(10x)^2 + (10y)^2}\right)}, \tag{3.13}$$

for $\boldsymbol{x} = (x, y) \in [-1, 1]^2$ and we associate to each item of the point cloud a weight $\omega_i > 0$ for $i = 1, \ldots, m$. For different weight values we compute the weighted least squares approximation $s(\boldsymbol{x})$ of the input data in terms of $521C^2$ quartic box splines [14], in their hierarchical extension [96].

In order to investigate the role of the weights in the approximation process, we execute the following two steps for fixed choices of ω_0 and $\omega_\gamma \in \mathbb{R}_{>0}$,

(a) we perform a standard least squares approximation and individuate the data sites whose approximation error is above a certain threshold ϵ, thus define $K := \{i \in \{1, \ldots, m\} : |s(\boldsymbol{x}_i) - f(\boldsymbol{x}_i)| > \epsilon\}$;

(b) or $i \in \{1, \ldots, m\} \setminus K$, we set the corresponding weights values $\omega_i = \omega_0$ and similarly for $i \in K$ we set $\omega_i = \omega_\gamma$.

We then analyse the accuracy behaviour of the resulting approximant $s(\boldsymbol{x})$ for different choices of $(\omega_0, \omega_\gamma)$ in terms of MAXimum error (MAX). Setting $\epsilon = 5\text{e}{-4}$, then $\#(K) = 124$ and for $\omega_0 = \omega_\gamma = 1$, the resulting ordinary least squares approximation is characterized by MAX$= 2.15\text{e} - 3$. We then keep $\omega_0 = 1$ and vary ω_γ between 1 and 100. The behaviour of MAX is depicted in Figure 3.2, for $\omega_\gamma = 1, \ldots, 10$. In particular, we note that increasing the value of the weights ω_γ may help the final accuracy of the approximant. More specifically, we obtain MAX$= 2.15\text{e} - 3, 2.06\text{e} - 3, 2.00\text{e} - 3$ for $\omega_\gamma = 2, 3, 4$ and MAX decreases further when increasing ω_γ until achieving its minimum among the sample values, with MAX$= 1.89\text{e} - 3$, for $\omega_\gamma = 6$. However, if the weight values ω_γ are too large, the final approximant deteriorates. For instance, for $\omega_\gamma = 7$, MAX$= 1.91\text{e} - 3$, for $\omega_\gamma = 10$, MAX$= 2.63\text{e} - 3$. If ω_γ would be further increased, the approximation quality would also deteriorate, e. g., for $\omega_\gamma = 50$, MAX$= 0.92\text{e} - 2$ and for $\omega_\gamma = 100$, MAX$= 1.36\text{e} - 2$. Finally, the choice of

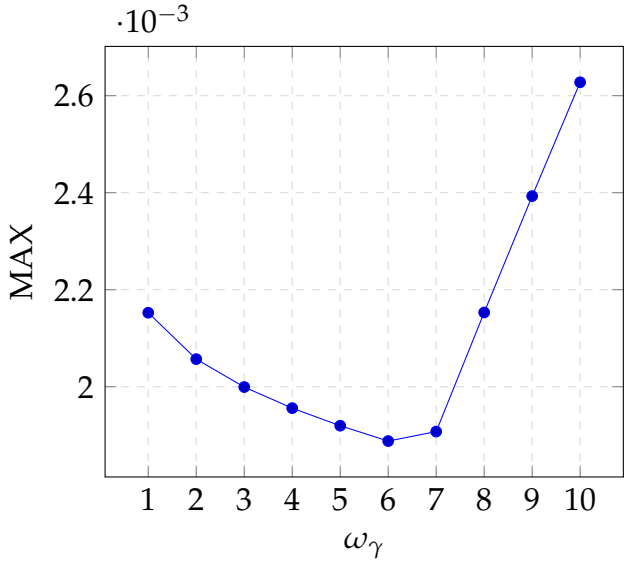


Figure 3.2: MAX trend with respect to the weights ω_γ for the weighted least suqares approximation of (3.13) with Hierarchical box splines.

weights can lead to either better or worse spline fitting results in terms of the MAX error compared to ordinary least squares, where all the weights are set to ones.

These results can also be observed in Figure 3.3, which depicts the hierarchical box spline mesh (left) together with two weighted least squares approximations resulting from two different choices of weights, namely $\omega_\gamma = 6$ (center) and $\omega_\gamma = 100$ (right). In particular, the scaled pointwise error color map indicates a worse approximation power for the latter weight choice.

As shown in this example, the choice of weights can lead to either better or worse spline fitting results in terms of the MAX error compared to ordinary least squares, where all the weights are set to ones. To the best of our knowledge, only a few deterministic methods address the use of weights associated with data points in the spline fitting literature. However, the topic is widely discussed in the statistics community, going back to locally weighted scatterplot smoothing (LOWESS) [34] for smoothing scatterplots by robust locally weighted regression [33] and multivariate adaptive regression splines (MARS) [66], with a more recent method proposed in [22] and the references therein, for example.

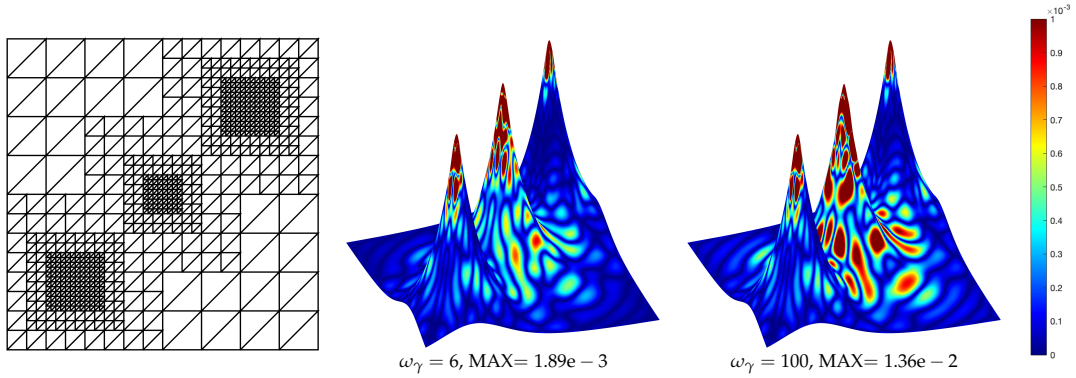


Figure 3.3: Hierarchical box spline mesh (left); error plots on the WLS hierarchical box spline approximations for $\omega_\gamma = 6$ (center) and 100 (right).

3.1.7 *Reweighted least squares spline fitting*

In this section, we propose a strategy to take effective advantage of the weights associated with each input data in the context of fitting problems. This approach is inspired by the notion of landmarks used in shape analysis [11], but we introduce a more general concept of *markers* and use them within the framework of curve and surface fitting. Landmarks can be understood as a set of labelled points that represent some physical identifiable parts of an object, as well as important features of the input data, which need to be encoded and reproduced in the final continuous approximation model. Figure 3.4 shows four point clouds (in blue) and the chosen landmarks (in black), which represent the features desired to be preserved by the fitted curve. However, depending on the acquisition process, data can be affected by noise and outliers, while the final approximation models should avoid the reproduction of corrupted data.

For the given set of observations as in (1.1), we generalize the concept of landmarks to *markers* of two types. More precisely, we define the index set

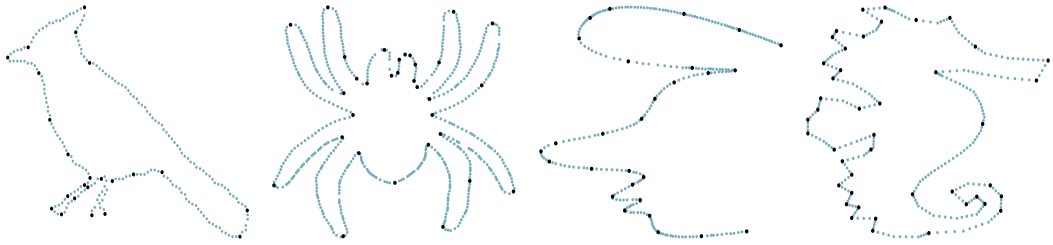
$$K_I \subseteq \{1, \dots, m\}$$

as *markers of type I* if the associated points $\{\boldsymbol{u}_i, \boldsymbol{p}_i\}$ for $i \in K_I$ represent data features to be preserved, while we define the index set

$$K_{II} \subseteq \{1, \dots, m\}$$

as *markers of type II* if the index indicates noisy data or outliers which should not be reproduced. In particular, we have

$$K_I \cap K_{II} = \emptyset \text{ and } K_I \cup K_{II} \subseteq \{1, \dots, m\}.$$



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

Figure 3.4: Point cloud (blue dots) and set of type I markers (black dots)

Note that the choice of type I and type II markers and their identification depends on the problem at hand, and it is still an open research topic; see, see e. g., for automatic feature selection [82, 190] or for outlier recognition [53, 54, 139].

The markers identification is out of the scope of the present Section and we assume the markers (of both types) to be known a priori. Nevertheless, for synthetic data, we devise an error-driven detection of markers through point cloud pre-processing. Finally, we provide fitting schemes that can address both types of markers simultaneously by leveraging the weight values associated with them depending on the approximation error, also within an adaptive approximation framework.

The fitting problem with markers consist of finding an element $\boldsymbol{s} \in V$ which approximate the points $\{\boldsymbol{u}_i, \boldsymbol{p}_i\}_{i=1}^m$, i. e. $\boldsymbol{s}(\boldsymbol{x}_i) \approx \boldsymbol{p}_i$, with

$$\|\boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i\|_2 < \mathrm{tol}_I, \text{ for } i \in K_I \text{ and } \|\boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i\|_2 > \mathrm{tol}_{II}, \text{ for } i \in K_{II},$$

for two different input tolerances tol_I and tol_{II}.

Remark 15. *Note that usual formulation of a fitting problem can be interpreted as a special instance of the present one. In particular, it is equivalent to setting* $K_I = \{1, \dots, m\}$, $K_{II} = \emptyset$ *and choosing* $\mathrm{tol}_I = \mathrm{tol}_{II} = \epsilon$.

The reweighted least squares algorithm with markers is described in Algorithm 1 and it consists of the following steps. For an initial choice of the weights

1. Solve the weighted least squares approximation problem (3.5).
2. Update the point-wise error $e_i = \|\boldsymbol{s}(\boldsymbol{x}_i) - \boldsymbol{f}_i\|_2$ for each $i = 1, \dots, m$.

3. Check whether the current fitting $\boldsymbol{s} \in V$ meets the requirements prescribed by the markers K_I and K_{II} and the respective error tolerances. If $e_i < \text{tol}_I$ for $i \in K_I$ and $e_i > \text{tol}_{II}$ for $i \in K_{II}$, stop and return the current approximant. Otherwise, if $e_i > \text{tol}_I$ and $i \in K_I$, set $\omega_i = \omega_i \cdot \alpha$ with $\alpha > 1$; if $e_i < \text{tol}_{II}$ and $i \in K_{II}$, set $\omega_i = \omega_i \cdot \alpha$ with $\alpha < 1$.

4. Start again from step 1 with the new weight values.

Note that the update choice of the weights in step 3 increases the values of the weights related to markers of type I, whereas it decreases them for markers of type II, for points not satisfying the tolerance conditions. In particular, we suggest an error-driven update of the weights

$$\alpha = \begin{cases} (1 + e_i), & \text{if } i \in K_I \\ \frac{1}{(1+e_i)}, & \text{if } i \in K_{II}. \end{cases} \tag{3.14}$$

The reweighted fitting scheme from Algorithm 1 is presented in its more general formulation, namely, the approximant is sought in any *fixed* finite-dimensional vector space V.

Finally, it is worth highlighting that the IRLS procedure [8, 142] falls within this general fitting scheme for the special choice of $\alpha = 1/\max\{\delta, e_i\}$ for each $i \in K_{II}$, with $\delta > 0$ necessary for the stability of the IRLS method.

3.1.8 *Reweighted least squares adaptive spline fitting*

In this section we show how to suitably combine the update of the weights with adaptive spline approximation schemes. More precisely, we will extend our scheme to advanced THB-spline constructions discussed in Section 2.1.3.

Once an initial configuration is chosen, any adaptive approximation procedure is characterised by four main steps which are successively repeated, to suitably identify the adaptive mesh to be used in the next iteration of the adaptive loop. In particular, any adaptive scheme as in (1.4), is characterized by 1. SOLVE, i. e. computation of the approximation on the current mesh; 2. ESTIMATE, i. e. error estimation; 3. MARK, i. e. mesh marking strategy; REFINE, i. e. mesh refinement strategy. We revisit the adaptive global least squares method proposed in [101], by suitably assigning weights values to the data observation within the adaptive routine. The main idea which drives an adaptive fitting algorithm consists in adding iteratively degrees of freedom in regions of the domain Ω where the approximation error exceeds a certain input tolerance ϵ.

Algorithm 1: General formulation of the reweighted least squares fitting.

Input: Point cloud $\{\boldsymbol{u}_i, \boldsymbol{p}_i\}_{i=1}^m$, the set of markers $K_I, K_{II} \subseteq \{1, \dots, m\}$, the tolerances tol_I, tol_{II}, a fixed vector space $V = \text{span}\{\beta_0, \dots, \beta_n\}$ and a maximum number of iterations $\text{M}_{\max}$;

1 Initialize the weights $\omega_i = 1$ and the point-wise errors $e_i = 1$ for each $i = 1, \dots, m$ and $\text{loop} = 0$

2 **while** $\max_{i \in K_I} e_i > \text{tol}_I$ *and* $\max_{i \in I \setminus K_{II}} e_i > \text{tol}_{II}$ *and* $\text{loop} < \text{M}_{\max}$ **do**

3 Solve the weighted least squares problem

$$\boldsymbol{s}(x) = \arg\min_{\boldsymbol{v} \in V} \frac{1}{2} \sum_{i=1}^{m} \omega_i \|\boldsymbol{v}(\boldsymbol{x}_i) - \boldsymbol{p}_i\|_2^2.$$

4 Compute the errors

$$e_i = \|\boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i\|_2, \text{ for } i = 1, \dots, m.$$

5 If $e_i > \text{tol}_I$ and $i \in K_I$, or $e_i < \text{tol}_{II}$ and $i \in K_{II}$, update the weights associated to the landmaks K_I and K_{II}, namely

$$\omega_i = \omega_i \cdot \alpha, \text{ with } \alpha \text{ as in (3.14).}$$

6 Set $\text{loop} = \text{loop} + 1$.

7 **end**

Output: $\boldsymbol{s} \in V$ the reweighted least squares approximant.

Remark 16. *In the presence of landmarks, the algorithm will be then characterized by three input threshold, i. e.* ϵ*, which guides the adaptive refinement and* tol_I *and* tol_{II} *which guide the weights update.*

Similarly to Algorithm 1, if the points with a too high error belong to K_I, then their weights will be augmented; otherwise, if they have a too low error and they belong to K_{II}, their weights will be diminished. In addition, at each iteration of the adaptive loop, not only the updates of the weight values but also of the sets K_I and K_{II} take place. This strategy is effective since, thanks to the adaptive refinement, in some regions of the domain the accuracy requirements $\text{tol}_I, \text{tol}_{II}$ are already locally achieved, and it is useless or even harmful to keep on modifying the weight values.

Once the initial configuration is chosen, i. e. for a fixed THB-spline space and for an initial choice of the weights and fixed tolerances the first step of the

adaptive fitting loop, i. e. SOLVE, consists in computing the control points $\boldsymbol{c}_j^\ell$ for each $j \in A_{\boldsymbol{k}}^\ell$ and $\ell = 0, \dots, L-1$, to define the geometric model in (2.15). As concerns the weighted least squared method, this is achieved by solving the penalized least squares problem

$$\min_{\substack{\boldsymbol{c}_j^\ell, j \in A_{\boldsymbol{k}}^\ell, \\ \ell = 0, \dots, L-1}} \frac{1}{2} \sum_{i=1}^{m} \omega_i \left\| \boldsymbol{s}\left(\boldsymbol{u}_i\right) - \boldsymbol{p}_i \right\|_2^2 + \lambda J\left(\boldsymbol{s}\right), \tag{3.15}$$

where the penalization term J is the thin-plate energy functional, whose influence is controlled by a weight $\lambda \geq 0$, i. e. for $\boldsymbol{x} = (x, y) \in \Omega$,

$$J\left(\boldsymbol{s}\right) = \int_\Omega \left(\left\| \frac{\partial^2 \boldsymbol{s}}{\partial x \partial x} \right\|_2^2 + 2 \left\| \frac{\partial^2 \boldsymbol{s}}{\partial x \partial y} \right\|_2^2 + \left\| \frac{\partial^2 \boldsymbol{s}}{\partial y \partial y} \right\|_2^2 \right) \mathrm{d}x\mathrm{d}y. \tag{3.16}$$

The second step of the adaptive fitting loop, i. e. ESTIMATE, consists of evaluating the THB-spline approximant on the data sites $\boldsymbol{u}_i \in \Omega$ related to the data points $\boldsymbol{p}_i$ to compute a suitable error indicator. In particular, we choose the point-wise error distance

$$\left\| \boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i \right\|_2$$

for each $i = 1, \dots, m$, among others. The error indicator indicates the region of the domain Ω where additional degrees of freedom are needed to meet the prescribed surface accuracy, by individuating the sites $\boldsymbol{u}_i \in \Omega$ where it exceeds a certain input threshold, i. e.

$$\left\| \boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i \right\|_2 \geq \epsilon.$$

For the making strategy, i. e. MARK, we select the cells of the current hierarchical level ℓ that contain the parameters $\boldsymbol{u}_i$ identified by the error indicator and mark them for refinement, together with two surrounding rings of cells in the hierarchical mesh. Subsequently, the marked cells are dyadically refined to effectively enlarge the hierarchical spline space, i. e. REFINE.

WEIGHTS UPDATE

Finally, if

$$\left\| \boldsymbol{s}(\boldsymbol{x}_i) - \boldsymbol{p}_i \right\|_2 < \epsilon$$

for each $i = 1, \dots, m$, we accept the approximation computed in (3.15), otherwise before performing another loop of the iterative procedure, starting again from SOLVE, we update the weights values and the marker sets K_I and K_{II}. In

particular, for each $i \in K_I$, if $e_i < \text{tol}_I$, then the i-th data satisfies the accuracy requirements and should not belong to the markers of type I any more, namely $K_I = K_I \setminus \{i\}$. On the contrary, if $e_i > \text{tol}_I$, then its corresponding weight needs to be enlarged, i. e. $\omega_i = \alpha \cdot \omega_i$, with $\alpha > 1$. Similar considerations hold for $i \in K_{II}$, with inverted inequalities and $\alpha < 1$. A reasonable error-driven choice for α can be again the one suggested in (3.14).

Note that in the SOLVE step we introduced the penalization term, see (3.15) and (3.16), usually addressed as *thin-plate energy*, whose influence is ruled by a weight $\lambda \geq 0$. The introduction of such a functional is a common practice in spline geometric modelling, in particular when reconstructing a spline geometry from a set of unorganized data. More precisely, a regularization term is commonly introduced to smoothen the solution and therefore avoid the presence of spurious oscillations and artefacts, which may affect the final geometric model. According to [101], we set $\lambda \geq 0$ to be a constant small ($\sim$ e$-$6) value and we keep it fixed during the entire fitting process. Non-constant regularization weight functions for data fitting via least-squares tensor-product splines have been recently proposed in [114, 136], see also the references therein, e. g., [36, 79, 178].

For more details about the marking and refinement strategies, we refer the reader to [101], whereas the pseudo-code of the reweighted least squares adaptive spline algorithm is reported in Algorithm 2.

3.1.9 *Numerical results for rWLS spline models*

In this Section, we present a selection of numerical experiments to show the performance of the proposed reweighted least squares fitting schemes. In particular, in Example 3.1.9 (a), we show the effectiveness of the proposed reweighted least squares spline fitting method to recover the sharp features of the four different point sequences with type I markers, depicted in Figure 3.4, and confront it with ordinary least squares spline fitting. Moreover, in Example 3.1.9 (b), for the task of curve fitting, we compare the proposed method also with smoothing splines approximations. Finally, in Example 3.1.9 (c) we extend the proposed method to adaptive spline spaces for surface reconstruction and suggests an automatic recognition of sharp features and type I markers.

The univariate examples have been implemented in an 8-core laptop (Apple M2) with 8 GB RAM using `MATLAB R2023b`, specifically by employing the `Curve Fitting Toolbox` [92]. The adaptive surface approximations have been implemented within the open source `C++` Geometry + Simulation (`G+Smo`) library [94,

Algorithm 2: Reweighted adaptive least squares spline fitting.

Input: Point cloud $\{\boldsymbol{u}_i, \boldsymbol{p}_i\}_{i=1}^m$, the set of markers $K_I, K_{II} \subseteq \{1, \dots, m\}$, the tolerances tol_I, tol_{II} and $\epsilon > 0$, the penalization weight $\lambda \geq 0$, a tensor product spline space V^0 and a maximum number hierarchical level L.

Initialize the weights $\omega_i = 1$ and the point-wise errors $e_i = 1$ for each $i = 1, \dots, m$ and loop $= 0$

while $\max_i e_i > \epsilon$ *and* loop $<$ L **do**

  Solve the penalized weighted least squares problem

$$\min_{\substack{\boldsymbol{c}_j^\ell, j \in A_k^\ell, \\ \ell = 0, \dots, L-1}} \frac{1}{2} \sum_{i=1}^m \omega_i \left\| \boldsymbol{s}\left(\boldsymbol{u}_i\right) - \boldsymbol{p}_i \right\|_2^2 + \lambda J\left(\boldsymbol{s}\right).$$

  Estimate the pointwise errors

$$e_i = \|\boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i\|_2, \text{ for } i = 1, \dots, m.$$

  For each $i \in K_I$, **if** $e_i > \text{tol}_I$ **then**

    update $\omega_i = \omega_i \cdot \alpha$, with $\alpha > 1$

  else

    $K_I = K_I \setminus \{i\}$.

  end

  For each $i \in K_{II}$, **if** $e_i < \text{tol}_{II}$ **then**

    update $\omega_i = \omega_i \cdot \alpha$, with $\alpha < 1$

  else

    $K_{II} = K_{II} \setminus \{i\}$.

  end

  Mark the domain elements where $e_i > \epsilon$ and two surrounding rings of cells in the hierarchical mesh.

  Perform dyadic refinment of the marked cells.

  Update the hierarchical mesh $\mathcal{M}$ and the hierarchical space V.

  Set loop = loop+1.

end

Output: $\boldsymbol{s} \in V$ the reweighted adaptive least squares approximant.

129], by suitably extending the `gsHFitting` class. Both libraries provide an efficient and robust implementation of least squares problems by exploiting suitable solvers for the linear systems at hand, which avoid the direct solution of a linear system of normal equations.

Example 3.1.9 (a)

Reweighted spline curve fitting with type I markers

To show the performance of Algorithm 1, we consider the point clouds and markers of type I depicted in Figure 3.4. The point clouds are similar to the ones presented in the following works: for Figure 3.4 (a) [141], (b) [106], (c) and (d) [125]. Our main goal is to improve the reconstruction while keeping the spline space intact. Each considered point sequence $\mathcal{P}$ is equipped with suitable uniform parameter values $\mathcal{U}$, and we compute the approximations of each dataset for the same spline space, i. e. with the same polynomial degree and the same amount of interior nodes in the *uniform* knot vector.

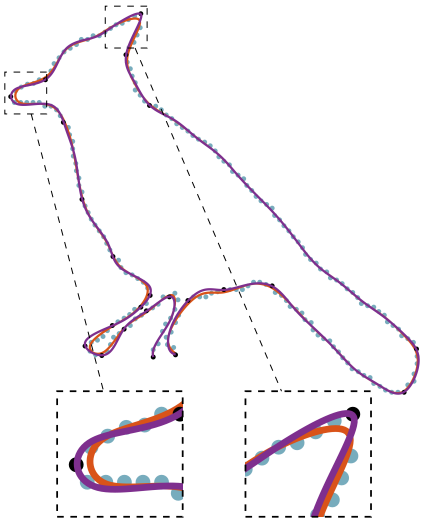
We set the tolerance of the proposed method as $\text{tol}_I = 1\text{e}{-3}$ and compare the solution of the rWLS problem with the solution of the ordinary LS problem, i. e. all the weights are equal to one. The solutions are depicted in Figure 3.5 (a), (b), (c), and (d), for the input data of Figure 3.4 (a), (b), (c), and (d), respectively. In particular, we can clearly see that the approximation using the reweighted least squares method (depicted in purple in Figure 3.5) shows better accuracy with respect to the solution of the ordinary least squares problem (shown in red). In particular, we can see that the marked data (illustrated in black) are better approximated by the fitted curves.

Regarding the approximation error for each experiment, we report that, if considering *all* the observation data, the Rooted Mean Squared Error (RMSE) and MAX are comparable with the ordinary least squares. However, if we compare only how well we can preserve the sharp features, the rWLS method considerably outperforms the ordinary LS. More precisely, in Table 3.1, we report the RMSE and MAX errors measured only for the set of type I marks, respectively for rWLS and LS.

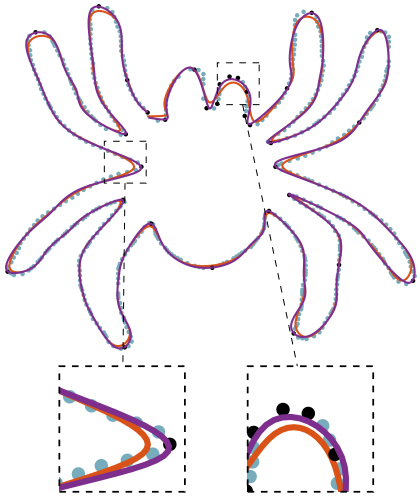
Example 3.1.9 (b)

Comparison with smoothing splines

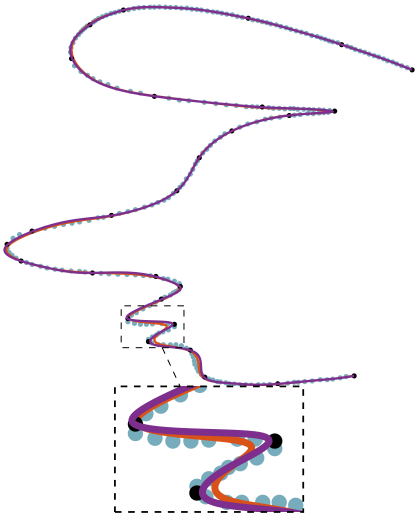
In addition to ordinary least squares, another widely used method for curve fitting is smoothing splines [79, 80]. This method is derived as the solution to the penalized least squares problem (3.15), in terms of cubic splines with knots corresponding to the x-coordinates of the observations.



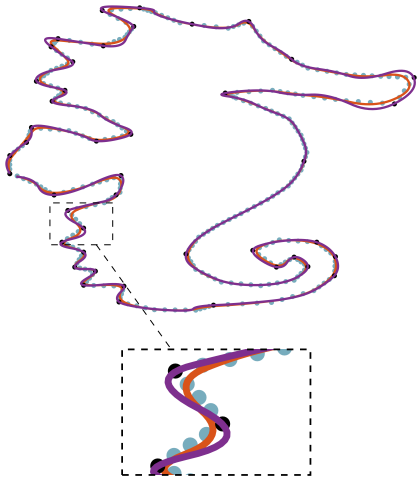
(a) Degree 3 and 50 interior knots



(b) Degree 3 and 60 interior knots



(c) Degree 3 and 40 interior knots



(d) Degree 2 and 84 interior knots

Figure 3.5: Curve fitting experiments described in Example 3.1.9 (a) using ordinary LS (red) and rWLS (purple) for data with markers of type I.

Therefore, we considered three additional point clouds sampled from the following functions:

$$f_1(x) = \left|\frac{9\sin(3\pi x)}{\tanh(-1.5x+1)+1}\right|, \quad f_2(x) = \frac{1}{0.02\sqrt{\pi}}\exp\left\{-\left(\frac{x-0.5}{0.02}\right)^2\right\}, \quad f_3(x) = \tanh\left(\frac{\cos(2\pi x)}{0.05}\right)$$

	Method	Iterations	Time	RMSE	MAX
(a)	LS	–	–	6.27e−3	1.30e−2
	rWLS	1442	1.35s	8.31e−4	8.95e−4
(b)	LS	–	–	2.25e−3	3.94e−3
	rWLS	1250	1.79s	6.63e−4	1.78e−3
(c)	LS	–	–	4.63e−3	1.22e−2
	rWLS	30	0.03s	1.47e−3	3.54e−3
(d)	LS	–	–	5.94e−3	1.25e−2
	rWLS	81	0.14s	3.77e−4	8.81e−4

Table 3.1: RMSE and MAX errors for type I markers of LS and rWLS solutions in Example 3.1.9 (a). For rWLS also the number of iterations (Iterations) and the computational time (Time) are reported.

and create 3 point clouds with 62, 88, 71 points, respectively. Specifically, we consider an irregular distribution of the abscissas to obtain more observations around the functions' sharp features.

We compute the smoothing spline fitting using the `fit` function in `MATLAB` and use the default "interesting range" value of the smoothing weight λ, which depends on the abscissa distribution of each experiment. Moreover, we set the weights of (3.15) to 1.

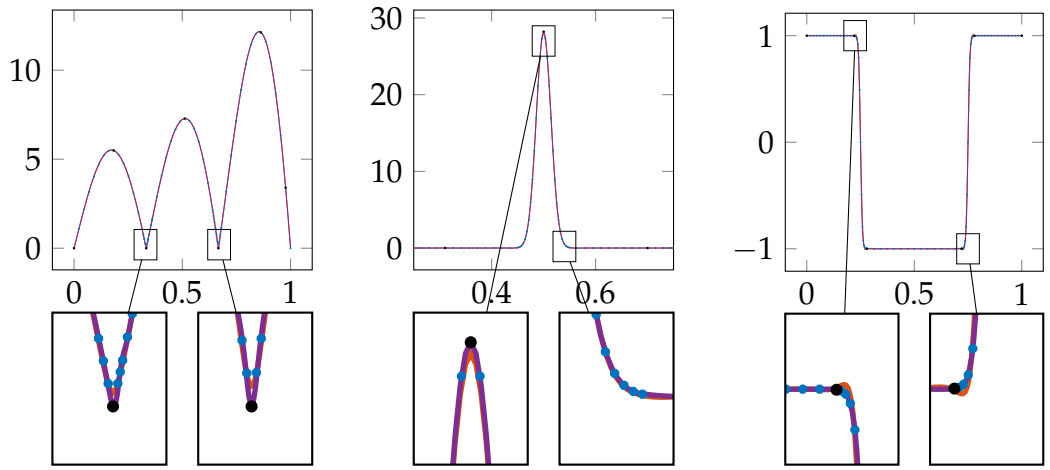
To ensure a fair comparison, for the rWLS algorithm, we fixed the spline degree to 3 and selected the knot vectors following the "averaging" technique described in [145, eq. (9.8)], known in the literature also as NKTP, with 37, 47, and 47 interior knots, respectively. This technique has been chosen since it also considers the values of the abscissas as in the case of smoothing splines.

The solutions obtained using rWLS from Algorithm 1 are depicted in Figure 3.6. Additionally, we report the RMSE and MAX measured only for the set of type I marks for both rWLS and smoothing splines in Table 3.2. To summarize, in the presence of sharp features, better results can be obtained using the proposed rWLS from Algorithm 1 with fewer degrees of freedom compared to the smoothing spline technique.

Example 3.1.9 (c)

Reweighted adaptive spline fitting for type I markers

In this numerical experiment, we apply Algorithm 2, combined with THB-splines, to a point cloud obtained by sampling 100×100 gridded data from function



(a) Approximation of f_1 with 37 interior knots (b) Approximation of f_2 with 47 interior knots (c) Approximation of f_3 with 47 interior knots

Figure 3.6: Curve fitting experiment described in Example 3.1.9 (b) using smoothing spline (red) and rWLS (purple) for data with markers of type I (black dots).

	Method	Iterations	Time	RMSE	MAX
f_1	smoothing spline	–	0.20s	1.02e−1	2.21e−1
	rWLS	10	0.09s	9.13e−6	1.86e−5
f_2	smoothing spline	–	0.19s	1.36e−1	3.04e−1
	rWLS	7	0.03s	1.43e−5	3.18e−5
f_3	smoothing spline	–	0.14s	2.96e−3	3.86e−3
	rWLS	4	0.01s	1.30e−5	2.58e−5

Table 3.2: RMSE and MAX errors for type I markers of smoothing spline and rWLS solutions of the experiment described in Example 3.1.9 (b). For rWLS also the number of iterations (Iterations) is reported.

(3.13). This function is characterized by sharp features that we want the final model to capture, i. e. we will deal with type I markers only, hence $K_{II} = \emptyset$.

To initialize the set K_I we perform an ordinary LS fitting with tensor-product B-splines, of bi-degree $\boldsymbol{d} = (3, 3)$ and a 15×15 mesh, and identify the data sites whose approximation error is above a certain threshold ϵ, namely

$$K_I := \{i \in \{1, \dots, m\} : \|v(\boldsymbol{u}_i) - f(\boldsymbol{u}_i)\|_2 > \epsilon\}.$$

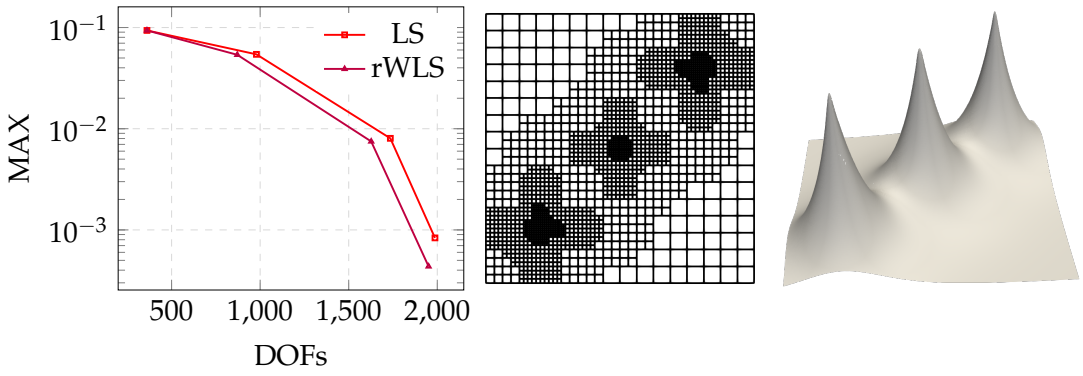


Figure 3.7: MAX error w.r.t. DOFs for the ordinary LS and rWLS fitting scheme with THB-splines (left); hierarchical mesh (center) and geometry (right) obtained in output from the rWLS method, for Example 3.1.9 (c).

Given the data points, the markers K_I, the tolerances, and the initial tensor-product B-spline space, we perform Algorithm 2 with $\text{tol}_I = 10\epsilon$, augmenting the weights value by 25% at each iteration of the adaptive loop. We then compare the output of the rWLS fitting method with the result of the ordinary adaptive LS fitting; the weights are fixed to one along the iterative procedure. More precisely, we compare the MAX of the final approximations with respect to the number of Dregrees Of Freedom (DOFs), choosing ϵ to achieve a comparable number of DOFs. The comparison is reported in Figure 3.7 (left), together with the final hierarchical mesh (center) and THB-spline approximation (right) obtained with the rWLS method. More precisely, LS and rWLS methods have the same starting point, and as long as the adaptive refinement proceeds, we can notice that rWLS is able to register a smaller MAX error with fewer DOFs in comparison to LS.

3.2 HIERARCHICAL QUASI-INTERPOLATION WITH ADAPTIVE SPLINE CONSTRUCTIONS

In this Section, we consider another fitting routine, which leads to the definition of a hierarchical QI model with THB-splines, particularly suitable for the reconstruction of unstructured data sets. The proposed scheme facilitates the computation of high-quality approximations with an increased level of resolution only in strictly localized areas. More specifically, the computation of the QI control net consists of implementing a two-stage data fitting algorithm with THB-splines, by combining *local* LS B-spline approximations (first stage) with the assembly of the hierarchical quasi-interpolant based on THB-splines (second

stage). Finally, to construct the adaptive spline model approximating the whole data set, a suitable strategy to guide the adaptive refinement is developed.

3.2.1 *Local least squares spline fitting*

By focusing on two-stage spline approximation schemes for THB-splines, a QI operator Q is defined, so that

$$Q\left(\mathcal{U},\mathcal{P}\right)\left(\boldsymbol{x}\right)=\boldsymbol{s}\left(\boldsymbol{x}\right) \tag{3.17}$$

where $\boldsymbol{s}:\Omega\rightarrow\mathbb{R}^{N}$ is as in (2.15). In particular, in this case every coefficient $\boldsymbol{c}_{j}\in\mathbb{R}^{N}$ is computed in the first stage of the scheme by using a certain *local* subset of data $\mathcal{U}_{j}\times\mathcal{P}_{j}\subset\mathcal{U}\times\mathcal{P}$ for each $j\in\mathcal{A}_{\boldsymbol{k}}^{\ell}$, with $\mathcal{A}_{\boldsymbol{k}}^{\ell}$ as defined in (2.10), and for each $\ell=0,\ldots,L-1$, so that

$$\boldsymbol{s}\left(\boldsymbol{u}_{i}\right)\approx\boldsymbol{p}_{i},\ \text{for } i=1,\ldots,m.$$

Note that by exploiting THB-spline constructions, it is possible to define hierarchical QI schemes without any additional efforts with respect to their tensor-product formulations, see [165].

More precisely, for each $j\in\mathcal{A}_{\boldsymbol{k}}^{\ell}$ and $\ell=0,\ldots,L-1$, let β_{j}^{ℓ} be the *mother* tensor-product B-spline of the truncated τ_{j}^{ℓ}, i.e. $\tau_{j}^{\ell}=\text{Trunc}^{\ell+1}\left(\beta_{j}^{\ell}\right)$, see Definition 5 in Section 2.1.3. In addition, let

$$\Omega_{j}\subset\Omega,\ \text{so that}\ \Omega_{j}\cap\text{supp}\left(\beta_{j}^{\ell}\right)\neq\emptyset,$$

namely, a suitably chosen local subdomain which has a non-empty intersection with the support of β_{j}^{ℓ}. Subsequently, the indices

$$I_{j}=\left\{i\,|\,\boldsymbol{u}_{i}\in\mathcal{U}\cap\Omega_{j}\right\}\subset\left\{1,\ldots,m\right\},\ \text{so that}\, n_{min}\leq\left|I_{j}\right|\ll m$$

have to be selected, together with the corresponding data

$$\mathcal{P}_{j}=\left\{\boldsymbol{p}_{i}\,|\,i\in I_{j}\right\},\ \mathcal{U}_{j}=\left\{\boldsymbol{u}_{i}\,|\,i\in I_{j}\right\}. \tag{3.18}$$

Finally, denote with

$$\mathcal{B}_{j}:=\left\{\beta_{r}^{\ell}\,|\,r\in\Lambda_{\boldsymbol{k}}^{\ell,j}\subset\Gamma_{\boldsymbol{k}}^{\ell}\right\}$$

the subset of tensor-product B-splines which do not vanish on Ω_{j} (by definition β_{j}^{ℓ} belongs to it). Therefore, the coefficient $\boldsymbol{c}_{j}^{\ell}\in\mathbb{R}^{N}$ associated to τ_{j}^{ℓ}, correspond

to the coefficient associated with β_j^ℓ, so that $\tau_j^\ell = \text{Trunc}^{\ell+1}\left(\beta_j^\ell\right)$, in a *local* spline approximation $\boldsymbol{s}_j \in V_j$, with $\boldsymbol{s}_j : \Omega_j \to \mathbb{R}^N$ and $V_j \coloneqq \text{span}\{\mathcal{B}_j\}$.

In the case of surface approximation, hence for $D = 2$ and $N = 3$, the local spline space V_j has a reasonable approximation power and it includes the restriction to Ω_j of any linear polynomial, as we prove in the following Proposition.

Proposition 1. *The following space inclusion holds true,*

$$\Pi_{d|\Omega_j} \subseteq V_j = \text{span}\{\mathcal{B}_j\},$$

where $\Pi_{d|\Omega_j}$ denotes the restriction to Ω_j of the tensor-product space of bi-variate polynomials of bi-degree $\boldsymbol{d} = (d_1, d_2)$.

Proof. Let q be a cell of the tensor-product mesh G^ℓ associated to V^ℓ, such that $q \cap \Omega_j \neq$. Then the definition of $\Lambda_{\boldsymbol{k}}^{\ell,j}$, implies that

$$\text{if } q \subseteq \text{supp}\left(\beta_r^\ell\right), \text{ then } r \in \Lambda_{\boldsymbol{k}}^{\ell,j}.$$

Thus,

$$\text{span}\{\beta_r^\ell \mid q \subseteq \text{supp}\left(\beta_r^\ell\right)\} \subset V_j.$$

Since

$$\text{span}\{\beta_r^\ell \mid q \subseteq \text{supp}\left(\beta_r^\ell\right)\} = \Pi_{d|q},$$

the proof is completed considering that q is any cells of G^ℓ with non-vanishing intersection with Ω_j. □

Consequently, if the sites in U_j are not collinear, we then approximate $\mathcal{P}_j$ with

$$\boldsymbol{s}_j(\boldsymbol{u}) = \sum_{r \in \Lambda_{\boldsymbol{k}}^{\ell,j}} \boldsymbol{\alpha}_r^\ell \beta_r^\ell(\boldsymbol{u}), \tag{3.19}$$

with control points $\boldsymbol{\alpha}_r^\ell \in \mathbb{R}^N$ estimated by solving the following penalized tensor-product B-spline *local least squares* approximation,

$$\min_{\boldsymbol{\alpha}_r^\ell, r \in \Lambda_{\boldsymbol{k}}^{\ell,j}} \sum_{i \in I_j} \left\|\boldsymbol{s}_j(\boldsymbol{u}_i) - \boldsymbol{p}_i\right\|_2^2 + \lambda J(\boldsymbol{s}_j), \tag{3.20}$$

where the penalization term J has been chosen again as the thin-plate energy in (3.16). Hence, we define the j-th control point $\boldsymbol{c}_j^\ell$ associated to τ_j^ℓ as $\boldsymbol{\alpha}_j^\ell$. Finally, If the sites in U_j are collinear, we define the j-th control point $\boldsymbol{c}_j^\ell \in \mathbb{R}^N$ associated to τ_j^ℓ as

$$\boldsymbol{c}_j^\ell = \frac{1}{|I_j|} \sum_{j \in I_j} \boldsymbol{p}_j.$$

The existence and uniqueness of the local approximant (3.20) in the case of surface approximation, i. e. for $D = 2$ and $N = 3$, is guaranteed by the following Theorem.

Theorem 2. *Let $\boldsymbol{d} = (d_1, d_2)$ be the polynomial bi-degree. If the sites $\boldsymbol{u}_j \in \Omega_j$, $j \in I_j$ are not collinear, there exists a unique local splines $\boldsymbol{s}_j \in V_j$ minimizing the following objective function,*

$$\sum_{i \in I_j} \left\| \boldsymbol{s}_j(\boldsymbol{u}_i) - \boldsymbol{p}_i \right\|_2^2 + \lambda J(\boldsymbol{s}_j),$$

where the penalization term J has been chosen again as the thin-plate energy in (3.16). If such points in Ω_j are collinear, then such minimizer does not exists or is not unique.

Proof. The objective function of Theorem 2 can be split in the sum in the sum of three analogous objective functions, one for each component $\boldsymbol{s}_j^{(k)}$, for $k = 1, 2, 3$. Therefore, to alleviate the analysis and the notation, we develop the proof in the scalar case, which can be extended to $\mathbb{R}^3$ in a straightforward way.

Let $s_j : \Omega_j \to \mathbb{R}$ an element of V_j, hence

$$s_j(\boldsymbol{x}) = \sum_{r \in \Lambda_{\boldsymbol{k}}^{\ell,j}} c_j \beta_r^\ell.$$

Moreover, let $\dim V_j = \left(n_j + 1\right)$. Therefore, the functional $J(s_j)$ can be computed as

$$J\left(s_j\right) = \boldsymbol{c}^\top G \boldsymbol{c},$$

where $\boldsymbol{c} = \left(c_0, \ldots, c_{n_j}\right)^\top \in \mathbb{R}^{\left(n_j+1\right)}$ and $G \in \mathbb{R}^{\left(n_j+1\right)\times\left(n_j+1\right)}$, so that

$$G_{i,r} = \int_{\Omega_j} \left(\frac{\partial^2 \beta_i^\ell}{\partial x \partial x} \frac{\partial^2 \beta_r^\ell}{\partial x \partial x} + 2 \frac{\partial^2 \beta_i^\ell}{\partial x \partial y} \frac{\partial^2 \beta_r^\ell}{\partial x \partial y} + \frac{\partial^2 \beta_i^\ell}{\partial y \partial y} \frac{\partial^2 \beta_r^\ell}{\partial y \partial y} \right) \mathrm{d}x\mathrm{d}y,$$

where we are assuming that, in the adopted ordering of the tensor-product B-spline basis $\mathcal{B}_j$ of V_j, β_i^ℓ and β_r^ℓ are the i-th and r-th basis functions.

In addition,

$$\sum_{i \in I_j} \left(s_j \left(\boldsymbol{u}_i \right) - p_i \right)^2 = \| B\boldsymbol{c} - \boldsymbol{p} \|_2^2 = \boldsymbol{c}^\top B^\top B \boldsymbol{c} - 2\boldsymbol{p}^\top B\boldsymbol{c} + \boldsymbol{p}^\top \boldsymbol{p},$$

where

$$\boldsymbol{p} = \left(p_1, \ldots, p_{|I_j|} \right)^\top \in \mathbb{R}^{|I_j|}$$

denotes the vector collecting all the p_i, for $i \in I_j$, and $B \in \mathbb{R}^{|I_j| \times \left(n_j+1\right)}$ is the collocation matrix defined on $\mathcal{U}_j$ as in (3.4), for the basis $\mathcal{B}_j$ generating V_j.

The objective function can then be written also in the following quadratic form,

$$\boldsymbol{c}^\top \left(B^\top B + \lambda G \right) \boldsymbol{c} - 2\boldsymbol{p}^\top B\boldsymbol{c} + \boldsymbol{p}^\top \boldsymbol{p}.$$

As well known, a quadratic function admits a global unique minimizer if and only if the symmetric matrix defining its homogeneous quadratic terms is positive definite and in such case the minimizer is given by its unique stationary point. In our case, such a matrix is $B^\top B + \lambda G$ and the stationary points are the solutions of the following linear system of $n_j + 1$ normal equations in as many unknowns,

$$\left(B^\top B + \lambda G \right) \boldsymbol{c} = B^\top \boldsymbol{p}.$$

For all positive λ, the matrix $B^\top B + \lambda G$ is symmetric and positive semidefinite, since for any vector $\boldsymbol{\zeta} \in \mathbb{R}^{\left(n_j+1\right)}, \boldsymbol{\zeta} \neq \boldsymbol{0}$, it holds

$$\boldsymbol{\eta}^\top B^\top B \boldsymbol{\zeta} \geq 0, \text{ and } \boldsymbol{\zeta}^\top G \boldsymbol{\zeta} \geq 0,$$

where the right inequality descends from the fact that

$$\boldsymbol{\zeta}^\top G \boldsymbol{\zeta} = J\left(s_{\boldsymbol{\zeta}} \right),$$

with

$$s_{\boldsymbol{\zeta}} \left(\boldsymbol{x} \right) = \sum_{r \in \Lambda_{\boldsymbol{k}}^{\ell,j}} \zeta_r \beta_{r.}^{\ell} \left(\boldsymbol{x} \right).$$

If the points $\boldsymbol{u}_i$, $i \in I_j$ are distributed on Ω_j along a straight line $ax + by + c = 0$, Proposition 1 implies that it is possible to find $\boldsymbol{\zeta} \in \mathbb{R}^{(n_j+1)}$, $\boldsymbol{\zeta} \neq \mathbf{0}$, such that $s_{\boldsymbol{\zeta}}(\boldsymbol{x}) \equiv ax + by + c$, for all $\boldsymbol{x} \in \Omega_j$. This implies that $s_{\boldsymbol{\zeta}}(\boldsymbol{u}_i) = 0$, for each $i \in I_j$, namely the vector $B\boldsymbol{\zeta} \in \mathbb{R}^{|I_j|}$ vanishes. Moreover, also $0 = J(s_{\boldsymbol{\zeta}}) = \boldsymbol{\zeta}^\top G\boldsymbol{\zeta}$, since $s_{\boldsymbol{\zeta}|\Omega_j}$ is a linear polynomial. This proves that the symmetric positive semidefinite matrix $\left(B^\top B + \lambda M\right)$ is not positive definite when all $\boldsymbol{u}_i$, for $i \in I_j$ are collinear. This is also the only possible data distribution associated to a non positive definite matrix. If the points $\boldsymbol{u}_i$, for $i \in I_j$ are not collinear, if $\boldsymbol{\zeta} \in \mathbb{R}^{(n_j+1)}$, $\boldsymbol{\zeta} \neq \mathbf{0}$ is associated to a non-vanishing linear polynomial, it holds $\boldsymbol{\zeta}^\top G\boldsymbol{\zeta} = 0$, but $B\boldsymbol{\zeta} \neq \mathbf{0}$, hence $\boldsymbol{\zeta}^\top B^\top B\boldsymbol{\zeta} > 0$. On the other hand, if $\boldsymbol{\zeta} \in \mathbb{R}^{(n_j+1)}$, $\boldsymbol{\zeta} \neq \mathbf{0}$ is not associated to a linear polynomial, then $\boldsymbol{\zeta}^\top G\boldsymbol{\zeta} > 0$.

□

The computation of all the local approximations (3.19) for each $j \in \mathcal{A}_{\boldsymbol{k}}^{\ell}$ and $\ell = 0, \ldots, L-1$ terminates the first stage. The pseudo-code related to the computation of the first stage is illustrated in Algorithm 3.

Remark 17. *Note that each control point $\boldsymbol{c}_j^{\ell}$ depends on a local subset of the input data, i.e. $\mathcal{U}_j \subset \mathcal{U}$ and $\mathcal{P}_j \subset \mathcal{P}$; hence, differently from the scheme presented in Section 3.1.8, the resolution of a global LS linear system is avoided.*

Remark 18. *Solving local LS* tensor-product spline *approximations allow to directly exploit the local tensor-product spline spaces, hence it significantly simplifies the algorithm originally proposed for the first stage in [17, 19], where a variable-degree local* polynomial *approximation was considered. More precisely, the scheme here proposed does not require the selection of a suitable degree for the computation of any coefficient and eliminates the conversion of the computed approximant from the polynomial to the local tensor-product B-spline basis.*

Since the scheme is locally applied, an automatic and eventually data-dependent selection of the parameter λ in (3.20) could be considered. For example, the choice may take into account the cardinality of $\mathcal{U}_j$ of the local sample, or the area of Ω_j, which influence the value of the first and second addend in (3.20), respectively. In view of this influence, we can also observe that a constant value of λ implies that the balancing between the fitting and the smoothing term in the objective function usually increases when the size of $\mathcal{U}_j$ or the area of Ω_j increases. This is true in the second case because second derivatives are involved in the smoothing term. Both of these choices seem reasonable and are confirmed by the quality of the results obtained in our

Algorithm 3: Local least squares spline fitting.

Input: Point cloud $\{\boldsymbol{u}_i, \boldsymbol{p}_i\}_{i=1}^m$, a tensor-product mother B-spline β_j^ℓ with basis index $j \in \mathcal{A}_{\boldsymbol{k}}^\ell$ for some $\ell = 0, \ldots, L-1$, the underlying tensor-product mesh M^ℓ of level ℓ, the smoothing parameter $\lambda \geq 0$ and a minimum required number of local data $3 \leq n_{\min} \ll m$;

Initialize $\Omega_j = \operatorname{supp}(\beta_j^\ell)$ and

$$I_j = \left\{ i \,|\, \boldsymbol{u}_i \in \mathcal{U} \cap \Omega_j \right\}, \; \mathcal{P}_j = \left\{ \boldsymbol{p}_i \,|\, i \in I_j \right\}, \; \mathcal{U}_j = \left\{ \boldsymbol{u}_i \,|\, i \in I_j \right\}.$$

while $\left|\Omega_j\right| < n_{\min}$, **do**
  Enlarge Ω_j with the first surrouding ring of cells of M^ℓ
  Update I_j, $\mathcal{P}_j$ and $\mathcal{U}_j$ accordingly.
end
if *The sites in $\mathcal{U}_j$ are not collinear,* **then**
  Compute the tensor-product B-spline

$$\boldsymbol{s}_j(\boldsymbol{u}) = \sum_{r \in \Lambda_{\boldsymbol{k}}^{\ell,j}} \boldsymbol{\alpha}_r^\ell \beta_r^\ell(\boldsymbol{u}),$$

  by minimizing (3.20), and set $\boldsymbol{c}_j^\ell = \boldsymbol{\alpha}_j^\ell$,
else

$$\boldsymbol{c}_j^\ell = \frac{1}{\left|I_j\right|} \sum_{i \in I_j} \boldsymbol{p}_i.$$

end

Output: The coefficient $\boldsymbol{c}_j^\ell \in \mathbb{R}^N$ associated to β_j^ℓ.

experiments, where a constant value for λ is suitably chosen. For completeness, adaptive and local choice of λ for least squares data fitting by tensor-product B-spline surfaces has been proposed in [114, 136].

Differently from [17, 19], in order to better avoid overfitting, a lower bound $n_{\min}$ for the cardinality of $\mathcal{U}_j$ is now required, being $n_{\min} \geq 3$ the only extra input parameter added to the algorithm, besides λ, see Algorithm 3. To fulfill this condition, $\mathcal{U}_j$ is initialized as $\operatorname{supp}(\beta_j^\ell)$ and enlarged until $\left|\mathcal{U}_j\right| \geq n_{\min}$. Moreover, to guarantee the locality of the method, the choice of $n_{\min}$ should be

kept reasonably small with respect to the amount of input data m. On the other hand, it is not necessary to set a maximum value for controlling the enlargement of the local data set due to the choice of the refinement strategy that takes this aspect into account, as detailed in the Section 3.2.2 (Remark 19), where the operator Q is extended to the hierarchical spline spaces.

3.2.2 *Quasi-interpolation with THB-splines*

The second stage consists of assembling the global approximation defined in (2.15). In particular, due to the THB-spline properties and by following the general approach proposed in [165], the hierarchical QI in (3.17) can be constructed as

$$\boldsymbol{s}(\boldsymbol{x}) = \sum_{\ell=0}^{L-1} \sum_{j \in \mathcal{A}_{\boldsymbol{k}}^{\ell}} \boldsymbol{c}_j^{\ell}(\mathcal{P}_j, \mathcal{U}_j) \tau_j^{\ell}(\boldsymbol{x}), \text{ for } \boldsymbol{x} \in \Omega, \tag{3.21}$$

where each coefficient $\boldsymbol{c}_j^{\ell}(\mathcal{P}_j, \mathcal{U}_j)$ directly corresponds to the coefficient $\boldsymbol{\alpha}_j^{\ell}$ associated with each mother B-spline β_j^{ℓ} in (3.19), as described in Section 3.2.1 and further detailed in Algorithm 3. The realization of the first and second stage concludes the first step of the adaptive fitting loop, i. e. SOLVE. Thereby, the iterative design of the hierarchical mesh $\mathcal{M}$ and space V follows.

As for the adaptive fitting procedure presented in Section 3.1.8, we choose as error indicator to drive the refinement the point-wise error distance. This concludes the second step of the adaptive loop, i. e. ESTIMATE. More precisely, given an input threshold $\epsilon > 0$, we evaluate the THB-spline approximant on the data sites $\boldsymbol{u}_i \in \Omega$ related to the data points $\boldsymbol{p}_i$ and compute the Euclidean distance $\|\boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i\|_2$ for each $i = 1, \dots, m$. This allows us to individuate the sites $\boldsymbol{u}_i \in \Omega$ where $\|\boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i\|_2 \geq \epsilon$.

The MARK step of the adaptive loop considers a a marking strategy based on basis functions. In particular, the basis function of level ℓ which are active on the data sites characterized by $\|\boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i\|_2 \geq \epsilon$ are marked for refinement and replaced by basis functions of the successive hierarchical level $\ell + 1$.

Finally, the refinement strategy, i. e. REFINE, of the THB-spline fitting scheme has to be considered. For some data distributions, e. g., for unevenly scattered data, the parameter values corresponding to the local data set $\mathcal{U}_j$ can be concentrated in a small part of the support of a marked function, thus splitting its support may affect the quality of the final approximation. To better handle these configurations and exploit adaptivity, the refinement procedure is ruled by the parameter n_{loc}, with $n_{\text{loc}} \leq n_{\min}$. Thereby, consider a basis function $\tau_{\boldsymbol{k}}^{\ell}$

marked to be refined and its mother $\beta_{\boldsymbol{k}}^{\ell}$. Subsequently, we consider a splitting of the two sides of $\operatorname{supp}\left(\beta_j^{\ell}\right)$ in n_1 and n_2 uniform segments, respectively, and subdivide $\operatorname{supp}\left(\beta_j^{\ell}\right)$ in the resulting $n_1 n_2$ subregions, where we then check the presence of at least $\lceil n_{\text{loc}}/(n_1 n_2) \rceil$ data points. The pseudocode for the adaptive hierarchical QI spline fitting is illustrated in Algorithm 4.

Remark 19. *The requirement $n_{\text{loc}} \geq n_{\min}$ guarantees that the points needed to compute the coefficients associated with the new functions in the first stage of the next iteration can be found not too far from the support of the functions themselves. Indeed, in Algorithm 3, after a few enlargements, Ω_j will surely include the support of a refined function of the previous level intersecting* $\operatorname{supp}\left(\beta_j^{\ell}\right)$.

As a consequence, analogously to $n_{\min}$, a high value of n_{loc} contributes to the reduction of oscillations deriving from overfitting, but this value should also be low enough to guarantee that the refinement strategy can generate a hierarchical spline space with enough degrees of freedom for satisfying the given tolerance ϵ. For this reason, some tuning is necessary for a good selection of n_{loc}.

The proposed adaptive approximation method also extends to the case of surfaces closed in one (or even two) parametric directions, not addressed in previous works. Note that the local nature of the considered approximation approach makes the implementation especially easy, since coefficients associated with a THB-spline present at successive steps of the adaptive refinement procedure (even if possibly further truncated) do not need to be recomputed.

3.3 INDUSTRIAL APPLICATIONS FOR HIERARCHICAL QI SPLINE FITTING

In this Section, we consider the reconstruction of scattered data of industrial complexity, obtained by an optical scanning process of four different aircraft engine parts, represented in terms of bivariate THB-spline models with bi-degree $\boldsymbol{d} = (d, d)$. For each of these surfaces, as a characterizing dimension, we report the length R of the diagonal of the minimal axis-aligned bounding box associated to the given point cloud. Note that, in industrial applications it is common to require a certain percentage ν of data points to satisfy the error tolerance ϵ.

The results highlight the effects of considering a minimum number of local data points $n_{\min}$ (also) in the first stage of the method, as well as the improvements obtained by introducing a regularized B-spline approximation for each local fitting with respect to the scattered data fitting scheme considered in

Algorithm 4: Hierarchical QI spline fitting

Input: Point cloud $\{\boldsymbol{u}_i, \boldsymbol{p}_i\}_{i=1}^m$, a tensor-product mesh M^0 and the corresponding tensor-product spline space V^0, the maximum number of hierarchical levels $L \geq 1$, the smoothing parameter $\lambda \geq 0$ and a minimum required number of local data $3 \leq n_{\min} \ll m$, the error threshold $\epsilon > 0$ and the percentage of data points required to satisfy the error tolerance ν, minimum number of local data required for refinement $n_{\min} \leq n_{\text{loc}} \ll m$ and the number of support splits n_1 and n_2.

Initialize the hierarchical mesh $\mathcal{M} = M^0$, the hierarchical spline space $V = V^0$, the point-wise errors $e_i = 1$ for each $i = 1, \dots, m$ and set loop $= 0$

while $|\{i \mid e_i > \epsilon\}| > \nu \cdot m$ *and* loop $< L$ **do**

  Compute the coefficients of the tensor-product QI with Algorithm 3 and define $\boldsymbol{s} \in V$ as in (3.21).

  Estimate the pointwise errors $e_i = \|\boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i\|$, for $i = 1, \dots, m$.

  Mark the basis functions τ_j^ℓ active on $\boldsymbol{u}_i$ such that $e_i > \epsilon$.

  Refine by dyadic split of the marked cells which contains at least $\lceil n_{\text{loc}}/(n_1 n_2) \rceil$ in any of the $n_1 n_2$ sub-rectangles which uniformly split $\operatorname{supp}\left(\beta_j^\ell\right)$, mother of τ_j^ℓ.

  Update accordingly the hierarchical mesh $\mathcal{M}$ and the hierarchical space V.

  Set loop = loop+1.

end

Output: The hierarchical QI as defined in (3.21).

[17]. In order to try to produce a very detailed reconstruction, a quite small value has been chosen for $n_{\min}$, more precisely, we suggest selecting it among the integers in the range $[d^2, (d+1)^2]$. By combining these two changes, the two-stage approximation algorithm is more stable and unwanted oscillations are further reduced.

3.3.1 *Tensile*

In this example,we consider THB-spline approximations to reconstruct a part of a tensile from the set of 9281 scattered data shown in Figure 3.8, which has reference dimension $R = 2.5\text{e}{-2}\text{m}$. We compare the new local scheme based on local B-spline approximations with the algorithm based on local polynomial approximations of variable degree presented in [17], where this test was originally considered. Note that for this example we have never dealt with local sets of collinear points in our experiments.

As a first test, we ran both algorithms with the same setting considered in [17], namely, by starting with a 4×4 tensor-product mesh with $\boldsymbol{d} = (2, 2)$, threshold $\epsilon = 5\text{e}{-5}\text{m}$, percentage bound $\nu = 95\%$ and $n_{\text{loc}} = 20$. The algorithm with local polynomial approximations with the parameter choice considered in [17] led to an approximation with 2501 DOFs, 96.25% of points below the tolerance and MAX error of $1.23\text{e}{-4}\text{m}$. The new scheme based on local B-spline approximations with $n_{\min} = 6$, $\lambda = 1\text{e}{-6}$, and $n_1 = n_2 = 1$ generated a THB-spline surface with 1855 degrees of freedom that satisfies the required tolerance in 98.88% of points with a MAX error of $8.06\text{e}{-5}\text{m}$. The number of levels used is 5, but all the cells of the first two levels are refined.

As a second test, we ran both algorithms by starting with a 16×4 tensor-product mesh with $\boldsymbol{d} = (2, 2)$, percentage $\nu = 95\%$, threshold $\epsilon = 5\text{e}{-5}\text{m}$, and $n_{\text{loc}} = 15$. The algorithm with local polynomial approximations led to an approximation with 5922 degrees of freedom, 98.18% of points below the tolerance, and a MAX error of $1.44\text{e}{-4}\text{m}$. The surface and the corresponding hierarchical mesh are shown in Figure 3.8 (center). This approximation clearly shows strong oscillations on the boundary of the reconstructed surface due to a lack of available data points for the local fitting in correspondence with high refinement levels. The scheme based on local B-spline approximations, with $n_{\min} = 7$, $\lambda = 1\text{e}{-6}$, and $n_1 = n_2 = 1$, produced a THB-spline surface with 1960 degrees of freedom that satisfies the required tolerance in the 99.36% of the data points with a MAX error of $8.11\text{e}{-5}\text{m}$. The surface, free of unwanted oscillations along the boundary, and the corresponding hierarchical mesh are shown in Figure 3.8 (bottom). The number of levels used is 4, with the cells of level 0 completely refined.

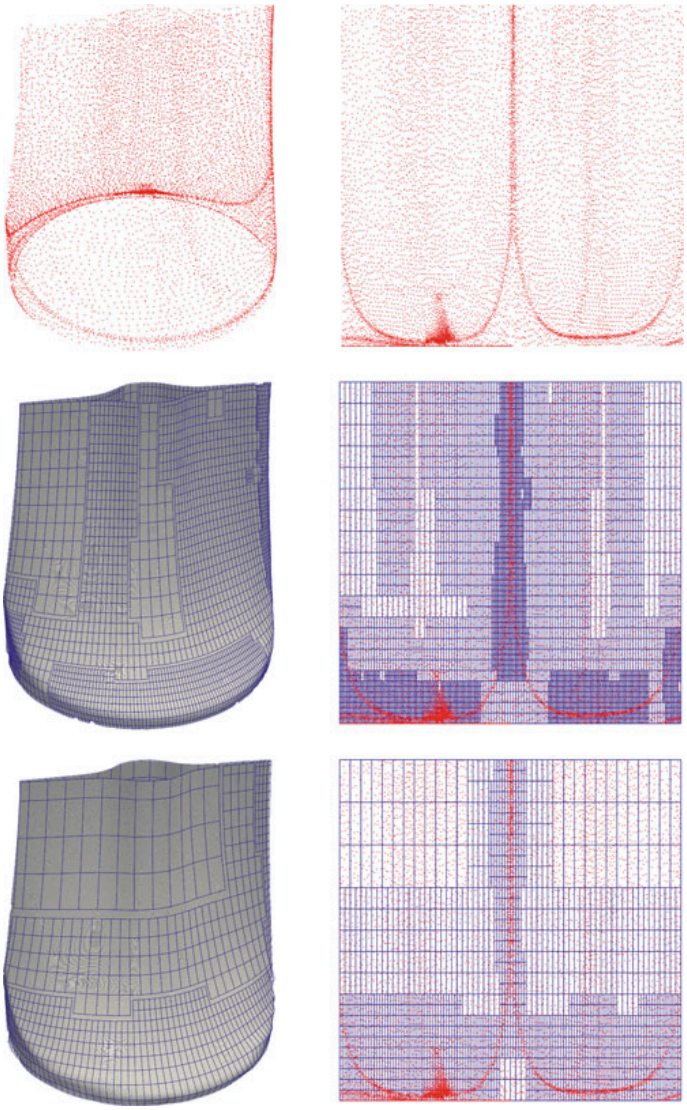


Figure 3.8: Scattered data set corresponding to a critical part of a tensile (top), the reconstructed surfaces and the corresponding hierarchical meshes obtained with the algorithm in [17] (center) and the new scheme (bottom) in Section 3.3.1.

3.3.2 *Blade*

In this example, we consider the second test discussed in [17] on the set of 27191 scattered data representing a scanned part of a blade shown in Figure 3.9 (top), whose reference dimension is $R = 5e{-}2m$. Again, to compare the new local scheme with the algorithm based on local polynomial approximations, we ran

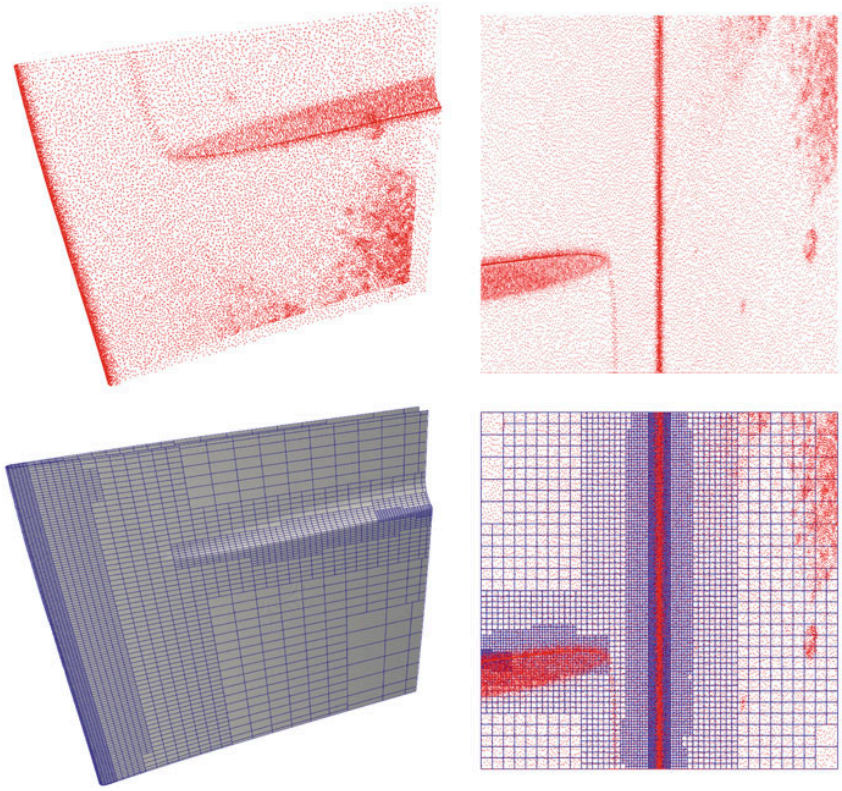


Figure 3.9: Scattered data set corresponding to a critical part of a blade (top), the reconstructed surface and the corresponding hierarchical mesh obtained with the new scheme(bottom) in Section 3.3.2

both algorithms with the same setting of [17], namely, by starting with a 4×4 tensor-product mesh with $\boldsymbol{d} = (3, 3)$, threshold $\epsilon = 2\text{e}{-5}\text{m}$, percentage bound $\nu = 95\%$ and $n_{\text{loc}} = 60$. The algorithm with local polynomial approximations with the parameter choice as in [17] led to an approximation with 12721 DOFs, 97.06% of points below the tolerance and MAX error of 1.08e−4m. The new scheme based on local B-spline approximations with $n_{\min} = 6$, $\lambda = 1\text{e}{-8}$ and $n_1 = n_2 = 1$ generated a THB-spline surface with 8314 DOFs that satisfies the required tolerance in 99.94% of points with a maximum error of 1.33e−4m. The surface and the corresponding hierarchical mesh are shown in Figure 3.9 (bottom). The number of levels used with the new scheme is 7, with all the cells of the first two levels completely refined, one less than with the old approach. Finally, for completeness, we precise that the local collinearity check in Algorithm 3 is active for 5 coefficients.

3.3.3 *Endwall*

In this example, we illustrate the behaviour of the adaptive algorithm on data sets with voids by considering the reconstruction of an endwall part from the scattered data set of 43.869 points shown in Figure 3.10 (top), whose reference dimension is $R = 5\text{e}{-1}\text{m}$. The figure shows that in this case, the data set represents a model with three different holes where no input data are available. The aim of this reconstruction is to avoid artifacts due to a lack of points and obtain a sufficiently regular surface, i. e. by avoiding self-intersections, that can be post-processed with standard geometric software tools to obtain a suitably trimmed model. Consequently, not only the number of points in the local data sets is important to reach this aim, but also their distribution. To properly address this issue, we consider a real density parameter with a value between 0 and 1, which determines whether the distribution of the points in the local set is reliable or not for the fitting. The distribution of the local data points is computed as the number of mesh cells of level ℓ inside the support of of the marked function β_j^ℓ or its enlargement, which contain at least one point, over the total number of mesh cells, either in the support of β_j^ℓ or its enlargement. If this ratio is below a density threshold, then more data points are required and the function support is enlarged for the computation of the local approximation in the first stage of the method. The approximation is developed by starting from a 32×32 tensor-product mesh, with $\boldsymbol{d} = (3, 3)$, $n_{\text{loc}} = 15$, $n_{\text{min}} = 11$, $\lambda = 1\text{e}{-6}$ and $n_1 = n_2 = 2$. A choice of the density parameter to 0.3 permits to take care of the difficult distribution of data points in the construction of the approximation. By considering a threshold $\epsilon = 5\text{e}{-5}\text{m}$ and a percentage bound $\nu = 95\%$, the refinement generated a THB-spline approximation with 3 hierarchical levels, 11211 DOFs, 98.70% of points below the threshold and MAX error of $5.69\text{e}{-4}\text{m}$. The surface and the corresponding hierarchical mesh are shown in Figure 3.10. In this case there are 18 coefficients of the last level and 15 of the last but one (all associated with B-splines whose support intersects a void) such that the related $\mathcal{U}_j$ is made up of all collinear points.

3.3.4 *Airfoil*

In this last example, we illustrate the behaviour of the new adaptive fitting algorithm with local B-spline approximations for surfaces closed in one parametric direction. In particular, we test the scheme to reconstruct a blade airfoil from a set of 19669 scattered data, shown in Figure 3.11, whose reference dimension

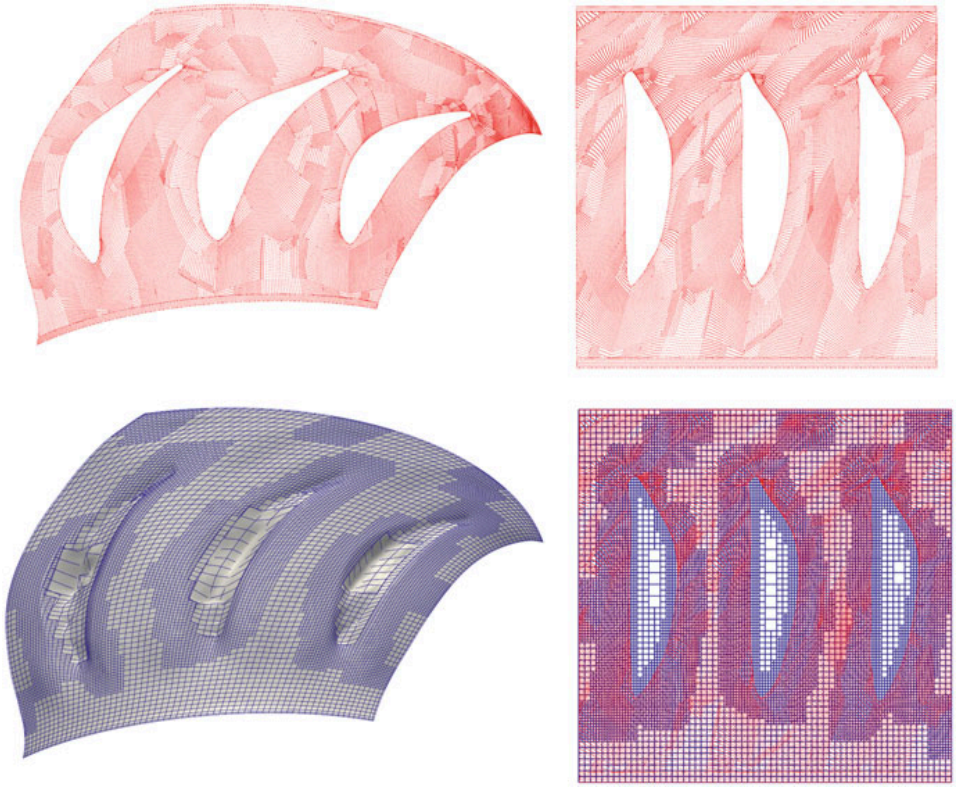


Figure 3.10: Scattered data set corresponding to a critical part of a endwall (top), the reconstructed surface and the corresponding hierarchical mesh obtained with the new scheme(bottom) in Section 3.3.3

is $R = 1\text{e}{-1}\text{m}$. We ran the method by starting from a 32×4 tensor-product mesh with $\boldsymbol{d} = (3, 3)$, setting the percentage bound $\nu = 0.95$, the threshold $\epsilon = 5\text{e}{-5}\text{m}$, $n_{\text{loc}} = 30$, $n_{\min} = 12$, $\lambda = 1\text{e}{-6}$, and $n_1 = n_2 = 1$. The adaptive scheme produces a THB-spline approximation with 3 hierarchical levels and 1865 DOFs distributed in the last two levels, and it satisfies the tolerance in 95.06% of the data points with a MAX error of $1.88\text{e}{-4}\text{m}$. The surface and the corresponding hierarchical mesh are shown in Figure 3.11 (bottom). By trying to force additional refinement, some oscillations appear. In this case, they are consistent with the data distribution since there are clusters of high-density points due to scan noise. Consequently, they do not represent artifacts caused by regions with very low density of data and cannot be prevented by exploiting the bound for cardinality of the local data sample governed by n_{loc}.

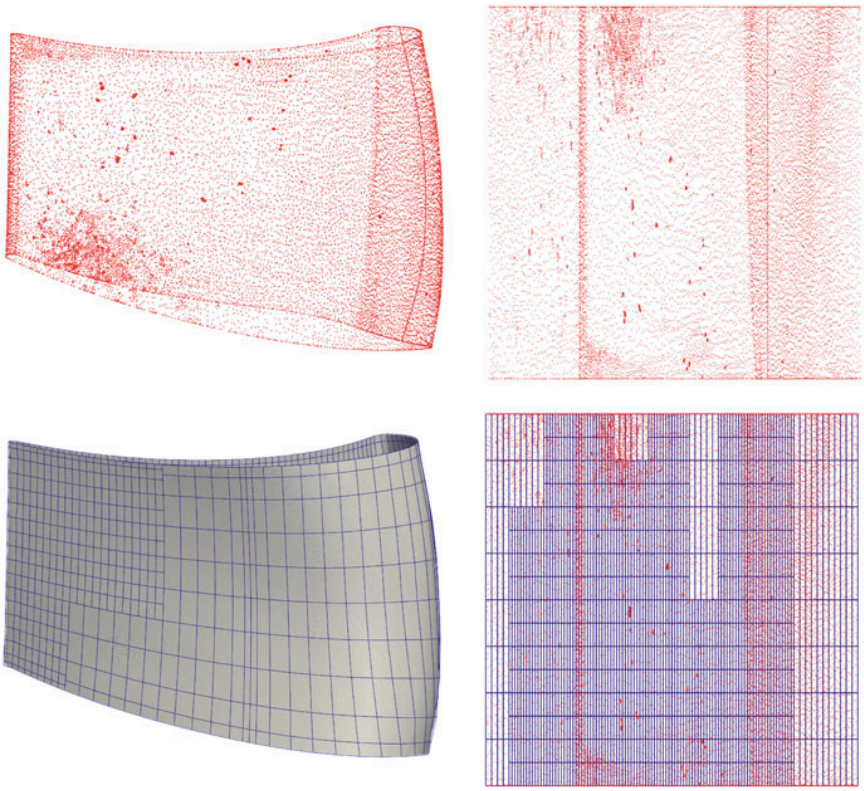


Figure 3.11: Scattered data set corresponding to a critical part of an airfoil (top), the reconstructed surface and the corresponding hierarchical mesh obtained with the new scheme(bottom) in Section 3.3.4

4

PARAMETERIZATION FOR POINT CLOUD SPLINE FITTING

In the previous Chapter we have introduced and analyzed the fitting problem of data observations of the form $\mathcal{U} \times \mathcal{P}$ as in (3.1), where the points $\mathcal{P}$ (1.1) in $\mathbb{R}^N$ are provided together with their locations $\mathcal{U}$ (1.3) over a subdomain $\Omega \subset \mathbb{R}^D$. Nevertheless, in many contexts, e. g., real-world applications, the data sites $\mathcal{U}$ are unknown, and the task of approximating a point cloud without data sites results in a complicated non-linear problem.

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

In this Chapter, we propose different *data-driven* parameterization procedures depending on the nature of the input data, i. e. whether they consist of point sequences or point clouds, as well as whether they are organized or scattered. More specifically, deep CNNs have led to improvements in image processing, see e. g., [104, 109, 110], but they also thrive when any kind of data involving spatial-correlated patterns, as for example audio files [167] and geometric data processing [180], have to be processed. Therefore, CNNs are employed for the parameterization learning problem of points on a rectilinear grid, as illustrated in Section 4.1. On the other hand, CNNs are not suitable to handle scattered data because of the lack of structure among the input data; hence, we propose to employ methods from geometric deep learning to properly address the parameterization problem in this setting. In particular, the parameterization learning problem for unstructured data configurations is solved by means of GCNs, which are a generalization of CNNs to non-Euclidean data, such as discrete manifolds, graphs, and general point clouds, as shown in Section 4.2 and Section 4.3. In all the considered examples, the proposed data-driven parameterization methods overcome state-of-the-art standard schemes. The present Chapter is mostly based on [41, 70, 71].

4.1 PARCNN: PARAMETERIZATION OF GRIDDED DATA WITH CONVOLUTIONAL NEURAL NETWORKS

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

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

Related works

Over the last decades several approaches have been developed to handle the parameterization of gridded data. Standard methods rely either on a UNIform (UNI) distribution of parameter values over the parametric domain or on scaled configurations of the physical points on the parametric space, as for example the CHord-Length (CHL) or CENtripetal (CEN) parameterizations [112]. Alternatives to these standard methods were presented in the literature, with special focus on the univariate setting. In particular, variants of the scaled parameterizations were proposed in [52, 131], whereas an example of an iterative procedure of different kind can be found in [2]. In addition, the so–called universal

parameterization method, closely related to the uniform parameterization, was presented in [31] in the context of univariate and bivariate spline interpolation. The potential of exploiting ML techniques for B-spline curve approximation was originally outlined in [107] and [106] by considering support vector machines and MLPs, respectively. While the focus of [107] was on identifying a suitable knot placement strategy, a scheme based on two interdependent deep neural networks to compute both the knot and parameter values was proposed in [106]. In particular, the neural network used to predict the parameters consists of a standard MLP that takes as input a planar point sequence of fixed length equal to 100 and returns as output the corresponding parametric values. Therefore, if the given number of data points does not match 100, pre- and post-processing steps for super- or sub-sampling the input data need to be performed. To overcome the computational limitations of this method, the authors in [160] proposed a method for univariate polynomial curve approximation based on a residual neural network. In this case, the network takes as input a planar point sequence of length $2d + 1$, where $d \in \mathbb{N}$ is the polynomial degree. The residual building blocks of the proposed networks are characterized by fully connected layers, which constrain the architecture to deal with an input of fixed size. Consequently, this approach also requires multiple network evaluations and a post-processing step to handle point sequences of arbitrary size.

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

4.1.1 *Problem presentation*

Let $\mathcal{P}$ be a point cloud, defined in (1.1) with items in $\mathbb{R}^N$ and characterized by a rectilinear grid structure along $D \leq N$ directions. More precisely, let $h = 1, \dots, D$ be the index associated with the parametric direction, and let m_h be the point sequence length along each direction h. We define the multi-index set

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

where each multi-index I uniquely identifies a point $\boldsymbol{p}_I$ of the point cloud $\mathcal{P}$. Consequently, the point cloud $\mathcal{P}$ in (1.1) can be rewritten as

$$\mathcal{P} = \left\{ \boldsymbol{p}_I \in \mathbb{R}^N \mid I = (i_1, \dots, i_D),\ i_h = 1, \dots, m_h,\ h = 1, \dots, D \right\}. \tag{4.1}$$

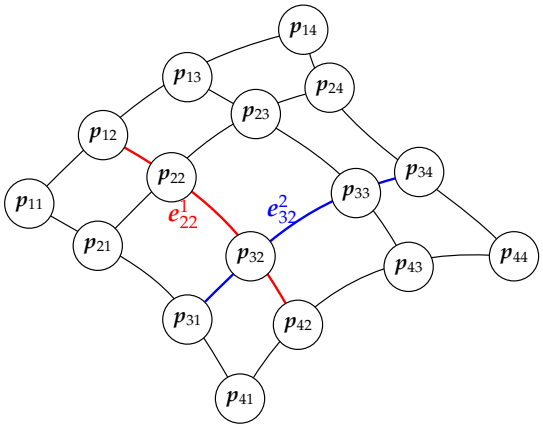


Figure 4.1: Example of a gridded point cloud with items in $\mathbb{R}^N$, with $N = 3$, and $D = 2$ parametric directions. The feature extraction step derives the edge vectors $\boldsymbol{e}_{ij}^h$ of the input grid.

We denote the cardinality of the point cloud as $m := \prod_{d=1}^{D} m_d$. Figure 4.1 shows an example of a point cloud $\mathcal{P}$, characterized by $D = 2$ parametric directions and elements belonging to $\mathbb{R}^N$, with $N = 3$. The point sequences in the two directions have lengths $m_1 = m_2 = 4$. Each item of the point cloud $\boldsymbol{p}_I$ is uniquely determined by the multi-index $I = (i_1, i_2)$, with $i_1, i_2 = 1, \dots, 4$.

To construct a gridded data approximation scheme, as for example the ones presented in Chapter 3, a parameterization of the point cloud $\mathcal{P}$ is needed. This problem requires to identify a set of suitable parametric values $\mathcal{U}$ as in (1.3), which can be rewritten as

$$\mathcal{U} = \{\boldsymbol{u}_I \in \Omega \mid I = (i_1, \dots, i_D)\,,\ i_h = 1, \dots, m_h,\ h = 1, \dots, D\}\,, \tag{4.2}$$

so that any point $\boldsymbol{p}_I \in \mathcal{P}$ in (4.1) is associated to the parameter $\boldsymbol{u}_I \in \mathcal{U}$. Moreover, without loss of generality, we set $\Omega = [0, 1]^D$.

The identification of a suitable parametrization method for the definition of the set $\mathcal{U}$ of parameter values is a challenging open problem within the polynomial spline fitting framework. In particular, it can naturally influence the quality of the final approximation, leading to sub-optimal approximation error results if not carefully handled; see, see e. g., [60, 63], which address the case of curve interpolation. In the tensor-product B-spline setting, see Section 2.1.2, once the univariate parameterization along each parametric direction is computed, the multivariate parameterization can be simply obtained by averaging across all the other parametric directions [145].

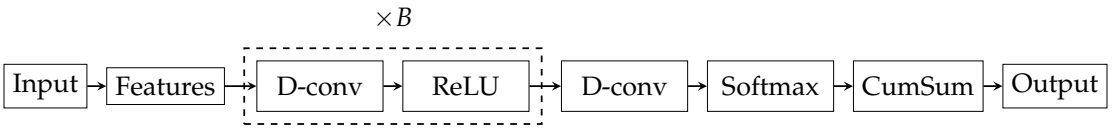


Figure 4.2: The CNN architecture.

The data connectivity information on proximity and distance plays a fundamental role in geometric data processing. Within the data learning framework, modelling the interactions between data relies on their weighted sums, computed via NNs convolutional operators, has become a building block for deep learning architectures to process data with a grid-like topology, for example when reconstructing a free-form model starting from a set of gridded observations $\boldsymbol{p}_I \in \mathbb{R}^N$, for $I \in \mathbb{I}$. Architectures that involve convolutional operators within their layers are addressed as CNNs; see Section 2.2.2. The architecture we designed is a *pure* CNN, hence consisting only of convolutional blocks. This choice has been made to take advantage of the locality of convolutional operators in order to be able to support variable input sizes without any additional effort and/or pre- or post-processing of the data. In particular, since the convolutional neural network is characterized by local operators, the particular size of the considered point clouds does not play a fundamental role, whereas the filter dimension does. Consequently, even if a convolutional model is trained on point cloud of fixed size, it can be easily generalised to point cloud of different dimensions. In the following, we detail the proposed architecture, together with information on the data generation, the hyperparameter selection, and the training process.

4.1.2 *Learning model architecture*

The proposed architecture, presented in Figure 4.2, is a sequential CNN, namely, a progression of layers which consists of a feature extraction layer, B convolutional building blocks (characterized by convolutional layers coupled with ReLU activation functions), a final block of one convolutional layer, the softmax activation function, and a cumulative sum layer connected to the output. We now give a detailed description of each component of our architecture to illustrate how to obtain the output parameterization result.

Input The learning model takes as input a point cloud $\mathcal{P}$ of gridded data, organised in a tensor structure, as in (4.1).

Features The feature extraction layer produces a relational representation of the point cloud that is translation invariant. It consists in the computation of $m_h - 1$ edges in $\mathbb{R}^N$ between each pair of consecutive points along each direction $h = 1, \dots, D$. Figure 4.1 illustrates the feature extraction process for a structured point cloud with $N = 3$ and $D = 2$. In particular, the edge associated to the element $\boldsymbol{p}_{22}$ along the direction $h = 1$ is $\boldsymbol{e}^1_{22} := \boldsymbol{p}_{32} - \boldsymbol{p}_{22}$ (shown in red in Figure 4.1), whereas the edge associated to the element $\boldsymbol{p}_{32}$ along the direction $h = 2$ is $\boldsymbol{e}^2_{32} := \boldsymbol{p}_{33} - \boldsymbol{p}_{32}$ (shown in blue in Figure 4.1). More precisely, for any $D \geq 1$, let $I = (i_1, \dots, i_D)$ for $i_h = 1, \dots, m_h - 1$, $h = 1, \dots, D$, and let $I + h$ be the multi-index I augmented by 1 along the h-th direction, for $h = 1, \dots, D$, namely $I + h := (i_1, \dots, i_h + 1, \dots, i_D)$. For all $I = (i_1, \dots, i_D)$ with $i_h = 1, \dots, m_h - 1$ and $h = 1, \dots, D$, we define the edges $\boldsymbol{e}^h_I$ as $\boldsymbol{e}^h_I := \boldsymbol{p}_{I+h} - \boldsymbol{p}_I$. Note that $\boldsymbol{e}^h_I \in \mathbb{R}^N$. By concatenating the edges $\boldsymbol{e}_I = \left(\boldsymbol{e}^1_I, \dots, \boldsymbol{e}^D_I\right)$, the extracted features are defined as the collection of edges

$$\mathcal{E} = \left\{ \boldsymbol{e}_I \in \mathbb{R}^{N \times D} \mid I = (i_1, \dots, i_D),\ i_h = 1, \dots, m_h - 1,\ h = 1, \dots, D \right\}. \tag{4.3}$$

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

D-conv The convolutional layers perform a D-dimensional convolution over the input, see e. g., Figure 2.13 in Chapter 2 for $1D$ and $2D$ convolution examples. Each block is characterized by discrete convolutional operators whose hyperparameters along each direction $h = 1, \dots, D$ are the number of input and output channels $c^h_{in}, c^d_{out} \in \mathbb{N} \setminus \{0\}$, the size of the convolving filter $\mathrm{ks}^h \in 2\mathbb{N} + 1$, the amount of padding added to the boundaries of the input $\mathrm{pad}^h \in \mathbb{N}$, the stride of the convolutional operator $\mathrm{s}^h \in \mathbb{N} \setminus \{0\}$, and the spacing between the filter elements $\mathrm{dil}^h \in \mathbb{N}$. By denoting with $\boldsymbol{u}^h \in \mathbb{R}^{c^h_{in} \times L^h_{in}}$ the input instance of any convolutional layer along the direction $h = 1, \dots, D$, and with $\boldsymbol{y}^h \in \mathbb{R}^{c^h_{out} \times L^h_{out}}$ its output, the relationship between the input (L^h_{in}) and the output (L^h_{out}) sizes is

$$L^h_{out} = \left\lfloor \frac{L^h_{in} + 2 \cdot \mathrm{pad}^h - \mathrm{dil}^h \cdot \left(\mathrm{ks}^h - 1\right)}{\mathrm{s}^h} + 1 \right\rfloor. \tag{4.4}$$

In order to fully convert the input along any direction $h = 1, \dots, D$, through the filter of dimension ks^h and obtain an output with the same

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

ReLU The ReLU is applied element-wise to the output of all except the last convolutional hidden layer. It is defined as $\sigma(z) := \max\{0, z\}$ and it is used for practical applications among most feedforward neural networks due both to its simple piecewise linear structure and its high performance [187].

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

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

$$\boldsymbol{u}^h_{(1,i_2,\dots,i_D)} = 0, \quad \boldsymbol{u}^h_{I+h} = \boldsymbol{x}^h_I + \Delta^h_I,$$

where $I + h$ is the multi-index I augmented by 1 along the h-th direction, for $h = 1, \dots, D$. Thanks to the application of the softmax activation function, for each $h = 1, \dots, D$, it is also ensured that the remaining boundary points are mapped to the boundary of the parameter domain, namely $\boldsymbol{u}^h_{(i_1,i_2,\dots,i_{D-1},m_D)} = 1$.

Output The output of the learning model consists of the parametric values $\mathcal{U}$ as defined in (4.2), associated to the input point cloud $\mathcal{P}$.

4.1.3 *Loss function*

Once the parameterization as in (4.2) is computed with the PARCNN model, we estimate the control points of the approximating geometry by solving the least squares minimization problem, see Section 3.1, in Bernstein-Bézier polynomial form. The reconstructed geometry $\boldsymbol{s} : [0,1]^D \to \mathbb{R}^N$ is then used to defined the loss function of our *unsupervised* learning model.

In particular, let

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

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

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

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

$$\mathcal{L}\left(\mathcal{U},\ \mathcal{P}\right) \coloneqq \frac{1}{2} \sum_{I \in \mathbb{I}} \| \widehat{\boldsymbol{p}}_I - \boldsymbol{p}_I \|_2^2 , \tag{4.5}$$

of the least-squares fitting problem (3.5) in Section 3.1, computed with the parameters $\mathcal{U}$ in output from the network. Note that our method falls into the unsupervised learning class, since the real parametric values are not considered to guide the optimization problem, hence our input data are unlabelled.

4.1.4 *Curve and surface parameterization learning*

In the following, we provide the information concerning the training of our models. The method we propose can be implemented for any spatial dimension $N \geq 1$ and parametric dimension $D \leq N$, thanks to the general definition of the convolutional operator. Note that we just set up and train a different model for any distinct pair of values N and D, independently of the considered point cloud. Moreover, since the input processed by the convolutional layers are the edges computed in (4.3), our parameterization method is affine invariant. Thereby, we focus on two cases: the parameterization of point sequences, when $N = 2$ or $N = 3$ and $D = 1$, and the parameterization of spatial gridded data, when $N = 3$ and $D = 2$, to subsequently compute polynomial and spline approximations in terms of (univariate) curves and (bivariate) surfaces, respectively.

Data generation

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

Hyperparameters selection and training process

Training our neural network consists in using our synthetic data to minimize the loss function (4.5), so that the model weights tend to be optimal for both seen and unseen input point clouds. In order to optimize the performance of the network, we set up an extensive grid search on optimizers, learning rates, filter sizes, and number of convolutional building blocks when applying the learning parameterization model in the context of planar polynomial curves, namely for $N = 2$ and $D = 1$. In particular, we tested three optimizers (Adam, SGD, RMSprop) with learning rates $\zeta = 1\mathrm{e} - i$, $i = 1, \dots, 6$. Inspired by [160], the filter size is set to be $2d + 1$, i. e. twice the polynomial degree plus one, and finally the number of building blocks is $B = 4$. In addition, we randomly select the point sequence length of each training batch in the range $m \in [2d + 1, 100]$. We obtained the best results using *RMSprop* as optimizer, learning rate equals to $1\mathrm{e} - 5$, and momentum 0.9 [169]. We then used these hyperparameter configurations also to train the model designed for higher dimensions, in particular for $N = 3$ and $D = 2$ to address the surface fitting problem. In this case we increase B to 5 and we fix the point cloud size to $m = 100$. Note that we train a different model for any value of N and D without a fixed number of steps. By following the so-called early stopping criteria, the training phase continues until the validation loss stagnates, a strategy commonly used to avoid overfitting.

Algorithm 5: Generation of rectilinear point clouds.

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

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

2 Sample

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

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

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

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

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

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

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

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

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

4.1.5 *Alternative deep learning parameterization models*

Even if the choice of the convolutional parameterization model is naturally inspired by the possibility of considering input data sets of variable dimensions, a comparison of its performance with alternative architectures is needed. We then also consider an MLP, a TRAnsformer encoder (TRA) and a RESidual neural network (RES) with convolutional layers. It should be noted that, except for MLP, both the TRA and RES architectures can also support point sequences of variable lengths. The performance comparison of these architectures presented in Table 4.1 motivates the choice of the CNN architecture, resulting in the proposed PARCNN model.

Multilayer Perceptron

The MLP we considered is represented in Figure 4.3 (left). It consists of a

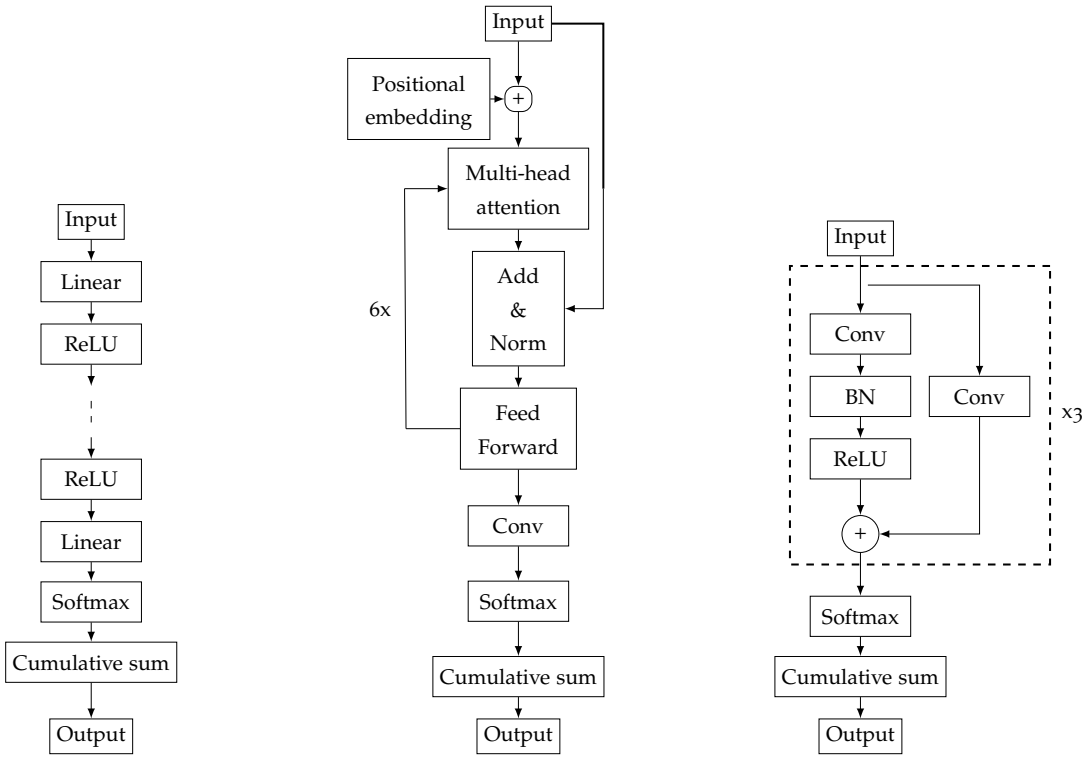


Figure 4.3: The MLP (left), TRA (center) and RES (right) architectures.

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

Transformer Encoder

By following recent advancements in sequence modelling for natural language processing with Bidirectional Encoder Representations from Transformers (BERT) [47], we applied a TRA to our point sequences. One of the most peculiar feature of Transformer is the so-called *Attention mechanism* [175], which introduces competition to select most valuable information from the input sequence. The structure of our transformer is illustrated in Figure 4.3 (center). In particular, we used a model architecture similar to BERT, which consists of 6 encoder layers with 2048-sized feed-forward layers, dropout probability 10% and N head attentions (instead of 8 as in BERT) to match our data dimension. The transformer takes as input the relative distances of the point sequences and its

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

Convolutional Residual Neural Network

In order to design our convolutional RES, whose configuration is illustrated in Figure 4.3 (right), we follow the structure of the residual neural networks presented in [160] and [85]. In particular, we assemble 3 convolutional residual blocks, each of them consisting in a feed-forward CNN with 75 (hidden) channels and skip connections, i. e. the 1st layer output has an additional convolutional layer connected with the 4th layer. More precisely, each residual block (represented with the dashed box in Figure 4.3(c)) has one D-convolutional layer to which the batch normalization and the ReLU follow. Furthermore, there is a skip connection, implemented as a D-convolution with kernel size 1, that takes the current residual block input and sums its output to the convolutional layer output. Softmax activation function and cumulative sum are then applied on the output of the residual blocks. The other convolutional hyperparameters are chosen as in Section 4.1.4 for the CNN proposed in Section 4.1.2. The motivation for choosing this architecture is to prevent the vanishing gradient problem on very deep architectures during the back-propagation phase, when the chain rule of derivatives can tend to 0 on top layers. Therefore, by adding skip connections we can shorten the distance between the starting layers and the last one. This architecture has similar advantages and disadvantages to a feed-forward CNN, i.e. can support variable lengths of point sequences but it is computationally slower compared to a feed-forward CNN due to the multiple branches.

4.1.6 *Numerical results for PARCNN*

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

Firstly, a comparison with state-of-the-art neural networks in the case of polynomial data is presented in Example 4.1.6 (a) and continues in Example 4.1.6 (b), which shows the flexibility of the proposed CNN-based model on input se-

quences of variable lengths. Trigonometric and noisy data approximation is addressed in Example 4.1.6 (c) in the context of adaptive spline curve approximation. The performance of the proposed CNN model on benchmark datasets for spline curve approximation is then illustrated in Example 4.1.6 (d).

By moving to the surface case, Example 4.1.6 (e) shows the results on polynomial data, together with the model robustness to noise. Subsequently, in Example 4.1.6 (f), we provide numerically evidence of the generalization capabilities of the proposed parameterization model with respect to datasets significantly different (in nature and size) from the synthetic data used during the training phase. Finally, in Example 4.1.6 (g) we investigate the performance of the learning parameterization model for tensor-product B-spline least squares fitting scheme.

Example 4.1.6 (a)

Architectures comparison

In this first example, we analyze the different learning parameterization models illustrated in the previous sections. More specifically, we compare our convolutional model (CNN) with the three deep learning models (MLP, TRA, RES) of Section 4.1.5 as well as with the results reported in [160, Table 1], i. e. Parameterization for Polynomial curve Network (PPN). In particular, we assembled all the new networks in order to have about the same number of learnable parameters (#LP) as PPN. As additional term of comparison, for each considered metric we also report the best value obtained with the STandarD (STD) schemes of the form

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

with

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

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

In all the required cases, the filter dimension is set to $2d + 1$. We trained MLP, TRA, RES and CNN on polynomial point sequences of length $m = 2d + 1$ generated with the algorithm reported in Section 4.1.4, and performed the tests on 100 data sequences of the same kind of the training set for degrees $d = 2, \dots, 5$. The results in terms of averaged MSE, MAX and Direct Hausdorff Distance (DHD) are reported in Table 4.1 together with #LP. The number of input

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

Table 4.1: Comparison of the different parameterization methods on polynomial data sets of $2d + 1$ points for $d = 2, 3, 4, 5$ in Example 4.1.6 (a). The learnable parameters (#LP) for the different neural networks are also shown.

units and the kernel size increase with the degree and, consequently, also the number of learnable parameters changes. As concerns TRA, the change is not evident within the first 3 significant digits, since the encoder architecture, which is the prevalent part of this network, is not affected by the input dimension and the kernel size. We can note that, except for TRA, the learning parameterization models outperform the standard parameterization techniques by gaining up to two (RES, PPN) or three (MLP, CNN) order of accuracy with respect to the different metrics. The best error results are highlighted in bold, in particular it should be noted that the CNN model always scores the lowest approximation errors, while also having the smallest amount of #LP.

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

Table 4.2: MSE error for Bézier approximation of degree $d = 2, 3, 4, 5$ on polynomial data for different input lengths $m = 20, 50, 80$ in Example 4.1.6 (b).

Example 4.1.6 (b)

Polynomial curve approximation

In this experiment, we exploit the PARCNN model on univariate polynomial data for different degrees d and variable point sequence lengths m. For each degree $d = 2, \dots, 5$, and length $m = 20, 50, 80$, we generate 100 polynomial point sequences $\mathcal{P} = \{\boldsymbol{p}_i \in \mathbb{R}^2 \mid i = 1, \dots, m\}$ as detailed in Section 4.1.4. Subsequently, we compute their parameterization $\mathcal{U} = \{u_i \in [0, 1] \mid i = 1, \dots, m\}$ with the proposed CNN-based approach and with standard parameterization schemes. We then compare the averaged MSE obtained building the Bézier fitting model at the parametric values given by the best standard technique (STD), the results presented in [160, Table 2] (PPN), and the ones obtained with our convolutional model (CNN). The outcome of this analysis is reported in Table 4.2, where the best approximation errors are highlighted in bold. The CNN outperforms STD methods and the PPN model in terms of MSE. In particular, the MSE obtained with CNN is reduced up to one and two orders of magnitude as regards PPN and STD, respectively. The advantage of our PARCNN model lies not only in the better performance shown in Table 4.2, but also in the novelty of its self-contained nature. For each input point sequence of any length, our convolutional learning method directly computes the corresponding point parameterization, without any additional computation. The approach proposed in [160] instead requires multiple network evaluations and a post processing step to handle arbitrary sequence lengths.

Example 4.1.6 (c)

Spline curve approximation of trigonometric and noisy datasets

In this example we investigate the generalization capabilities of our convolutional parameterization model to different datasets and adaptive B-spline curve fitting of various degrees ($d = 2, 3, 4, 5$), as well as its robustness to noise. By

considering the algorithms presented in Section 4.1.4, we generate two distinct datasets of size 100 by sampling trigonometric and polynomial curves. The trigonometric curves are defined as

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

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

We test the CNN parameterization by considering adaptive B-spline least-squares approximation obtained by performing dyadic refinement of the knot intervals which exhibit the highest error values. The resulting MSE is shown in in Table 4.3 for trigonometric data of degree $r = 5$ (top) and noisy polynomial data (bottom), for different input sequence lengths ($m = 20, 50, 80$) and adaptive B-spline approximations with a final number of 5 interior knots. The proposed CNN parameterization model always obtains better results, highlighted in bold, than the best standard methods (STD). In particular, in the case of trigonometric data (Table 4.3 (top)), it gains up to two order accuracy. For the results on noisy polynomial data (Table 4.3 (bottom)), the accuracy obtained with CNN model is improved up to one order of magnitude when compared to STD.

To better understand the generalization capabilities of the network when an increasing number of knots is considered, we present a final test with fixed input length equals to $m = 20$. We then perform an adaptive B-spline curve fitting with dyadic refinement, starting from the polynomial case of 0 internal knots up to 10 internal knots. The results in terms of MSE for the CEN, CHL, UNI and CNN parameterizations for $d = 3, 4, 5$ are shown in Figure 4.4 (top) for trigonometric data of degree $r = 5$ and in Figure 4.4 (bottom) for noisy polynomial data. We may observe that in Figure 4.4 (top) for $d = 3, 5$ the MSE error gap between CNN and STD is maintained or even increased with respect to the number of interior knots inserted. In Figure 4.4 (top) for $d = 4$ and in Figure 4.4 (bottom) for all the degrees, the MSE error gap between CNN and STD tends to reduce with the number of interior knots inserted, but the CNN parameterization model always achieves the best results.

Example 4.1.6 (d)

Spline curve approximation of benchmark datasets

In this experiment, we test our parameterization model on point sequences

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

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

Table 4.3: MSE for adaptive B-spline approximation with 5 interior knots of degree $d = 2, 3, 4, 5$ on trigonometric data (top) and polynomial data with random Gaussian noise (bottom) for different input lengths $m = 20, 50, 80$ in Example 4.1.6 (c).

sampled by the benchmark presented in [141]. By considering B-spline approximation of degree $d = 4$ with at most 15 adaptive interior knots, Figure 4.5 presents the results obtained employing the standard (UNI, CHL, CEN) and the CNN parameterizations on an increasing number of input points, namely $m = 20, 50, 80$. An analogous comparison on a sequence of $m = 65$ points, approximated with degree $d = 5$ B-splines and adaptive knot vectors with a varying number (4, 7, and 10) of interior knots is shown in Figure 4.6.

Example 4.1.6 (e)

Polynomial surface approximation

In this experiment we illustrate the performance of the proposed parameterization model on both exact and noisy data sampled from polynomial patches in the bivariate case. In particular, we collect structured point clouds by sampling Bézier surfaces of bi-degree $\boldsymbol{d} = (d, d)$. For each $d = 2, 3, 4, 5$, we generate 100 point clouds consisting of $m = 144$ gridded points following the algorithm described in Section 4.1.4. In addition, we also create a noisy version of this dataset by perturbing each point cloud with Gaussian noise, considering a factor $\epsilon = 5\text{e} - 3$. The results of the parametric polynomial fitting of bi-degree

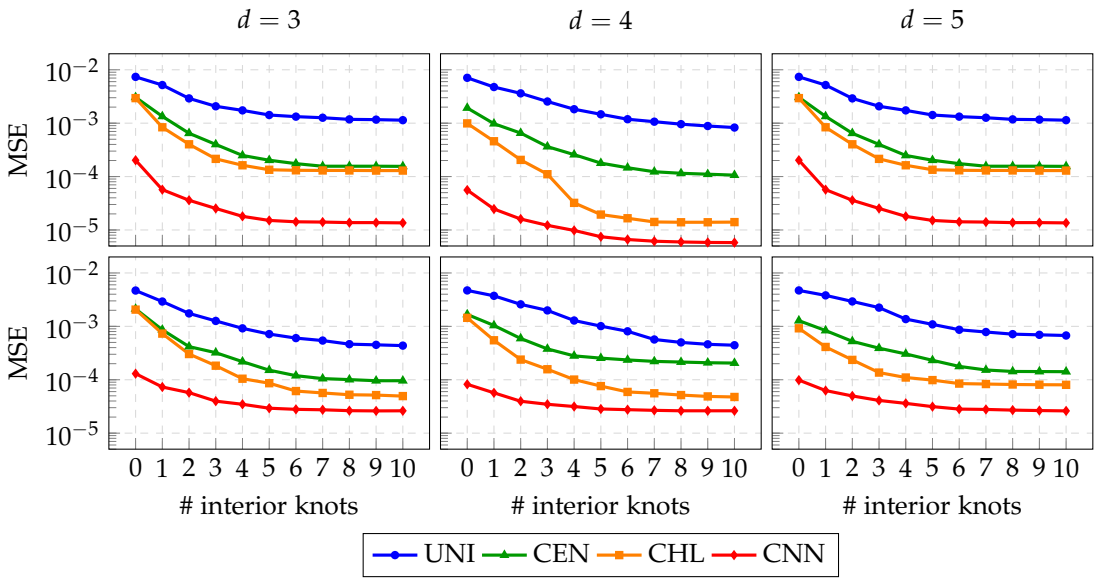


Figure 4.4: MSE for adaptive B-spline approximations with increasing number of knots tested on $m = 20$ trigonometric (top) and noisy (bottom) data for degree $d = 3$ (left), $d = 4$ (center) and $d = 5$ (right) in Example 4.1.6 (c).

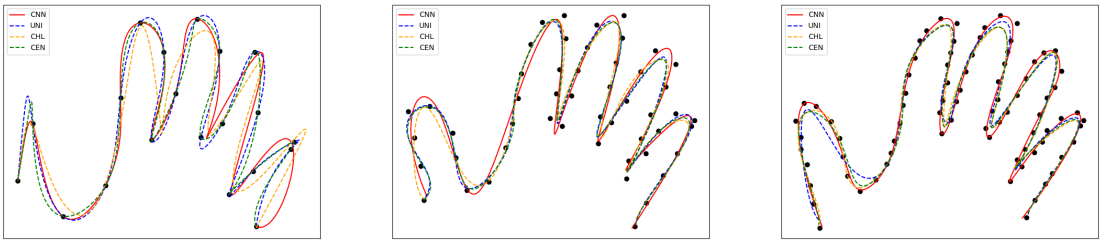


Figure 4.5: B-spline approximation of degree $d = 4$ with 15 interior knots for 20 (left), 50 (center), and 80 points (right) in Example 4.1.6 (d).

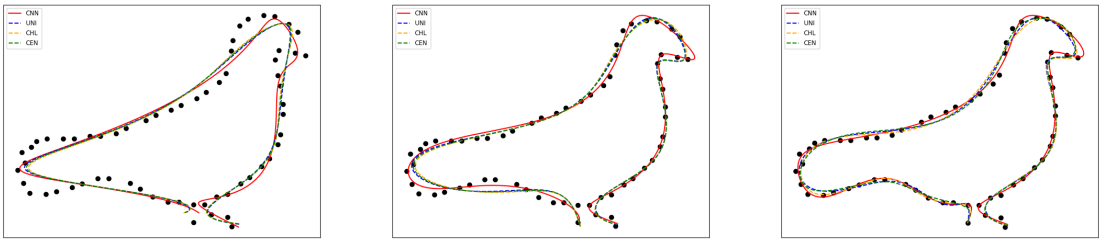


Figure 4.6: B-spline approximation of 65 points with $d = 5$ and 4 (left), 7 (center), 10 (right) interior knots in Example 4.1.6 (d).

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

Table 4.4: MSE error for polynomial least-squares approximation of polynomial (left) and noisy (right) data of bi-degree $= (d, d)$, for $d = 2, 3, 4, 5$ in Example 4.1.6 (e).

$\boldsymbol{d} = (d, d)$ of the aforementioned generated point clouds are reported in Table 4.4 in terms of MSE for the best standard parameterization technique (STD), among UNI, CHL and CEN, and the proposed convolutional model (CNN). The best approximation MSE values are highlighted in bold. The errors give numerically evidence of the generalization capabilities of the trained model with respect to datasets which are somehow similar but at the same time independent from the ones used in the training phase. In particular, the proposed model avoids potential over-fitting problems, it is robust to noise, and it always increases the accuracy of the polynomial approximation more than 50% over STD parameterization techniques in all the considered configurations.

Example 4.1.6 (f)

Polynomial approximation of geometric models

In this experiment, we analize the generalization capabilities of the parameterization learning model for data which are different both in dimension and nature from the one used in the training phase. More precisely, we sample point clouds of dimension either $m = 1089$ or $m = 3025$ from 3 different B-spline geometries of bi-degree $\boldsymbol{d} = (d, d)$, representing a car, a ship hull, and a wind turbine. The corresponding point clouds are illustrated in Figure 4.7 for the case of 1089 (top) and 3025 (bottom) samples. Moreover, for this experiment we have also stored the true parametric values $\mathcal{X}$ using during the sampling phase and we will refer to them as the ground truth (GT) in the comparisons. Once the data have been parametrized with the convolutional method (CNN), we performed polynomial approximation of bi-degree $\boldsymbol{d} = (d, d)$ and compare the MSE and MAX obtained with GT and CNN parameterizations. Note that the GT parameters come from a specific B-spline geometry with different degrees and knot lines configurations, therefore if used for different spline models they not necessarily lead to zero error at the evaluation sites.

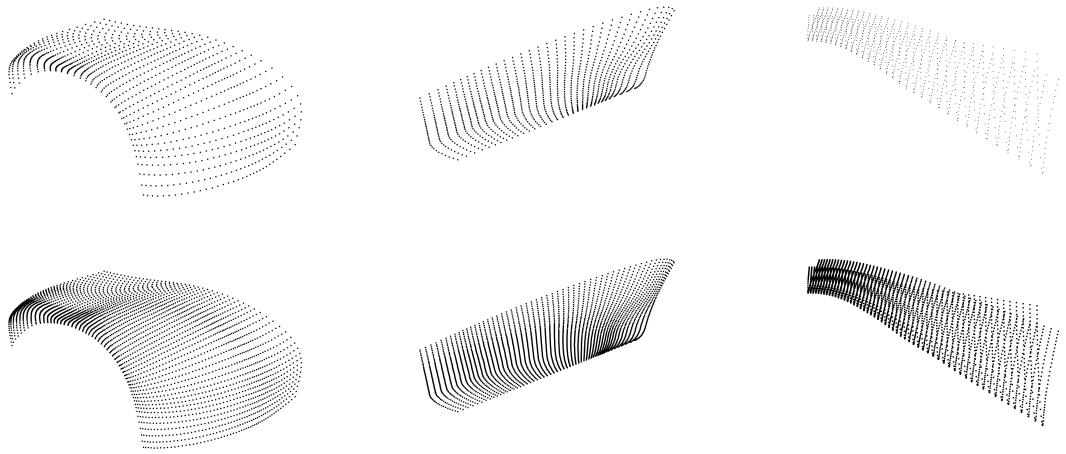


Figure 4.7: Point clouds obtained from a car (left), a ship hull (center), and a wind turbine (right) model of 1089 (top) and 3025 (bottom) points.

The results are reported in Table 4.5, where the best approximation values are highlighted in bold. They numerically show the generalization capabilities of the proposed model with respect to the size of the input point clouds and to the nature of the data. For the car geometry, in all the considered configurations, the MSE error of the final reconstructed polynomial approximation is smaller in the case of the CNN parameterization, with a gain in accuracy up to more than 50% for $d = 3$. When the ship hull geometry is considered, the best performances are achieved if the CNN parameterization is applied in combination with lower polynomial bi-degrees $\boldsymbol{d} = (d, d)$, namely $d = 2, 3$, and for $d = 2$ the final reconstructed geometry of the bigger point cloud gains more than 60% of accuracy. As concerns the final wind turbine geometry, the best advantage is registered on the point cloud of size $m = 1089$ among all the different bi-degrees, where the MSE is improved up to more than 60% for bi-degree $\boldsymbol{d} = (3, 3)$. The same bi-degree $\boldsymbol{d} = (3, 3)$ leads also to the best error results for the bigger wind turbine point cloud.

Example 4.1.6 (g)

Tensor product B-splines approximation

In this example we show the generalization capabilities of the convolutional

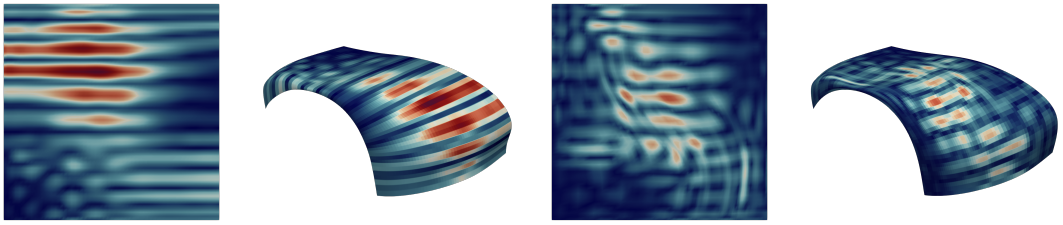
		car		ship hull		wind turbine	
	d	GT	CNN	GT	CNN	GT	CNN
$m = 1089$	2	$1.38e-3$	$\mathbf{8.28e-4}$	$2.53e-4$	$\mathbf{1.11e-4}$	$1.95e-3$	$\mathbf{1.30e-3}$
	3	$4.58e-4$	$\mathbf{1.94e-4}$	$7.28e-5$	$\mathbf{6.01e-5}$	$1.18e-4$	$\mathbf{3.94e-5}$
	4	$1.19e-4$	$\mathbf{7.63e-5}$	$2.78e-5$	$\mathbf{1.88e-5}$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

Table 4.5: MSE for polynomial least-squares surface approximation with different bi-degrees $\boldsymbol{d} = (d, d)$ of the car shell, the ship hull, and the wind turbine point clouds of size $m = 1089$ and $m = 3025$ in Example 4.1.6 (f).

parameterization for tensor product B-spline models. In particular, we consider the point clouds obtained from the car shell geometry introduced in Example 4.1.6 (f) for which we perform a tensor product B-spline approximation of bi-degree $\boldsymbol{d} = (3, 3)$ with 3 levels of uniform refinement, both considering the given parameterization (GT) and the convolutional parameterization (CNN).

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

The scaled error distributions of the final approximation plotted on the parametric and physical domain, for both GT and CNN parameterizations, are illustrated in Figure 4.8 for the point cloud of size 3025. More precisely, plots (a) and (b) in Figure 4.8 are the error distributions for the GT parameterization, whereas the corresponding results for the CNN parameterization are shown in plots (c) and (d) of the same figure.



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

Figure 4.8: Point-wise error on the parametric (a-c) and geometric (b-d) domain of the tensor product B-spline approximation of bi-degree $\boldsymbol{d} = (3,3)$ for the car shell point cloud of size 3025 using the given GT (a-b) and the CNN (c-d) parameterizations in Example 4.1.6 (g).

4.2 PARGCN: PARAMETERIZATION OF SCATTERED DATA WITH GRAPH CONVOLUTIONAL NEURAL NETWORKS

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

GCNs have become the counterpart of CNNs to process data belonging to discrete manifolds, graphs, or general point clouds, and many different approaches to defining convolutional operators on graph domains have been proposed [186]. In particular, the translation of standard filter operators to graph operators relies on suitable aggregations of vertex and neighbour features; see for more details Section 2.2.3. GCNs are employed in the next two Sections, i. e. Section 4.2 and Section 4.3, to address the parameterization problem of scattered point clouds.

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

[62], thus introducing a geometry-informed learning procedure that produces parameterizations of high quality.

While graph neural networks have been successfully applied to classification tasks such as shape recognition, segmentation and registration [83, 152, 163, 182], regression tasks where the network should predict a (potentially vector-valued) function on the input geometry are much less studied [6, 51, 71]. One difficulty is that GCNs aim to perform classification, segmentation or regression on the vertices $\mathcal{V}$ of a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. However, we are interested in performing regression on the edges $\mathcal{E}$ instead of the vertices $\mathcal{V}$ of $\mathcal{G}$. To overcome this problem, given the directed graph $\mathcal{G}$, we extract its line graph (dual-graph) [78, 84], which represents the adjacencies between edges of the original graph. The line graph is used as input to a geometry-informed graph convolutional neural network trained to learn optimal parameterizations.

4.2.1 *Meshless barycentric parameterization*

Let $\mathcal{P}$ be a scattered point cloud, as in (1.1). The adjective *scattered* usually indicates that the points do not have a regular structure. Thereby, the only assumption we make consists in assuming that the unorganized point cloud $\mathcal{P}$ can be partitioned into two disjoint subsets, i. e. $\mathcal{P} = \mathcal{P}_I \dot{\cup} \mathcal{P}_B$, with

$$\mathcal{P}_I = \left\{ \boldsymbol{p}_i \in \mathbb{R}^3 \mid i = 1, \dots, n \right\}, \tag{4.6}$$

the set of interior points, and

$$\mathcal{P}_B = \left\{ \boldsymbol{p}_i \in \mathbb{R}^3 \mid i = n+1, \dots, m \right\}, \tag{4.7}$$

the set of boundary points. Our objective consists in finding a suitable parameterization $\mathcal{U}$ as in (1.3) for the unorganized data in $\mathcal{P}$.

In [62], the authors propose a method for parameterizing an unstructured point cloud over a convex polygonal domain $\Omega \subset \mathbb{R}^2$. It consists of two steps:

1. *Boundary point cloud parameterization*. The boundary points $\mathcal{P}_B$ in (4.7) are parameterized over the boundary of the parameter domain Ω, resulting in parameters

   $$\mathcal{U}_B = \{ \boldsymbol{u}_i \in \partial\Omega \mid i = n+1, \dots, m \}, \tag{4.8}$$

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

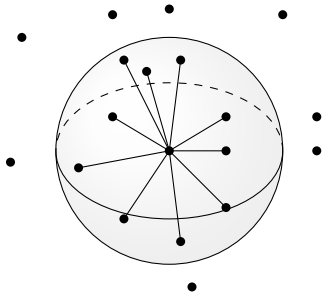


Figure 4.9: A radius neighbourhood.

2. *Interior point cloud parameterization*. A directed graph $\mathcal{G}$, whose vertices are the points in $\mathcal{P}$, is determined. The parameters that correspond to the points $\mathcal{P}_I$ in (4.6) are assumed to be convex combinations of their neighbours in the graph with certain weights. This leads to a sparse linear system whose solution is the parameterization of $\mathcal{P}_I$, i. e.

$$\mathcal{U}_I = \{\boldsymbol{u}_i \in \Omega \mid i = 1, \ldots, n\}\,. \tag{4.9}$$

As far as the first step of the method is concerned, several approaches are available for the parameterization of point sequences, as already discussed in Section 4.1. We then assume that the parametric values $\mathcal{U}_B$ in (4.8) corresponding to the boundary points $\mathcal{P}_B$ in (4.7) are known.

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

$$N_i = \left\{j \in \{1, \ldots, m\} : 0 < \|\boldsymbol{p}_i - \boldsymbol{p}_j\| < r\right\}, \quad i = 1, \ldots, n, \tag{4.10}$$

for some radius $r > 0$, as illustrated in Figure 4.9. Choosing a neighbourhood for each point in $\mathcal{P}_I$ leads to a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, whose vertices $\mathcal{V} = \{1, \ldots, m\}$ are the indices of $\mathcal{P}$ and whose directed edges $\mathcal{E}$ are all ordered pairs (i, j) such that $i \in \{1, \ldots n\}$ and $j \in N_i$. Each directed edge $(i, j) \in \mathcal{E}$ connects an interior index i with either an interior or a boundary index j. The graph $\mathcal{G}$ is the so-called fixed-radius near neighbour graph [4] from $\mathcal{P}_I$ into $\mathcal{P}$ or, for short, the *radius graph* of the points $\mathcal{P}$.

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

$$\boldsymbol{u}_i = \sum_{j \in N_i} \lambda_{ij}\, \boldsymbol{u}_j, \quad i = 1, \ldots, n, \tag{4.11}$$

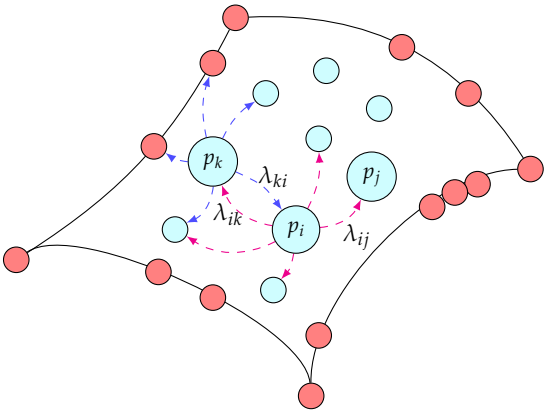


Figure 4.10: Examples of two directed neighbourhoods for the interior points $\boldsymbol{p}_i$ and $\boldsymbol{p}_k$, for any $i, k = 1, \ldots, n$. The interior points $\mathcal{P}_I$ in (4.6) are cyan, while the boundary points $\mathcal{P}_B$ in (4.7) are red.

of its neighbours in $\mathcal{G}$, where the *parameterization weights* fulfill $\lambda_{ij} > 0$ and $\sum_{j \in N_i} \lambda_{ij} = 1$. Figure 4.10 shows an example for two interior points $\boldsymbol{p}_i, \boldsymbol{p}_k \in \mathcal{P}_I$ (in blue) and their corresponding neighbourhoods N_i and N_k, where we identify the vertices in $\mathcal{G}$ with the corresponding points in $\mathcal{P}$.

We collect the linear equations (4.11) in a linear system

$$A\, U_I = B, \tag{4.12}$$

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

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

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

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

and $U_\mathrm{I} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n)^\top$ is the $n \times 2$ matrix that contains the unknown parameters of the interior points.

In order to guarantee that the matrix A in (4.12) has full rank, the choice of the radius r is crucial. As proved in [62, Proposition 3.3], each interior parameter

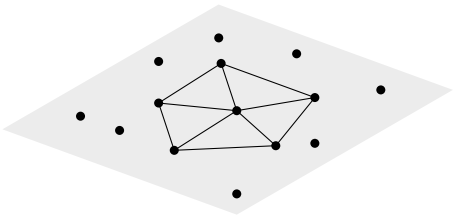


Figure 4.11: A Delaunay neighbourhood.

needs to be related to at least one boundary point by a sequence of linear equations, which corresponds to the existence of a path in $\mathcal{G}$ from each interior index $i = 1, \dots, n$, to any boundary index.

What remains to be determined is the specific choice of the parameterization weights λ_{ij} for $i = 1, \dots, n$ and $j \in N_i$. The latter determine the parameterization of $\mathcal{P}_I$ and it is therefore important to choose them carefully in order to obtain good results. The particular choice of λ_{ij} is still open, while some heuristics have been proposed. More precisely, in [62] two heuristic choices are suggested: UNIForm parameterization weights (UNIF) and RECiprocal Distance parameterization weights (RECD). The uniform weights aim at generalizing the concept of uniform parameterization of curves to surfaces, hence the parameterization weights are simply chosen as

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

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

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

A third choice that is proposed in [62] are the Local Projection Shape-Preserving parameterization weights (LPSP), which are not defined on the radius graph. Instead, for all interior points $\boldsymbol{p}_i$ the neighbourhood (4.10) is projected onto its best-approximating plane. The resulting planar point cloud is then triangulated by a Delaunay triangulation [43, 81]. The neighbours of $\boldsymbol{p}_i$ in the Delaunay triangulation are its neighbours in the *local projection* graph $\mathcal{G}$, see Figure 4.11. The corresponding weights are found based on the shape preserving weights for triangulated surfaces presented in [58]. For each interior point $\boldsymbol{p}_i$, the neighbours are ordered counter-clockwise and intermediate local parameters $\tilde{\boldsymbol{u}}$ are determined such that for all $j \in N_i$

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

and

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

i. e. the lengths and the angle ratios around $\boldsymbol{p}_i$ are preserved. Finally, corresponding weights λ_{ij} are computed from the parameters $\tilde{\boldsymbol{u}}_j$. For details about the implementation we refer to [58].

In general, there is no constraint on the choice of the weights, besides all weights being positive and weights that belong to a common interior point to sum up to one, i. e.

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

Consequently, the space of possible weights, and resulting parameterizations, is very large. This motivates us to train a deep neural network to predict the optimal choice of parameterization weights.

In the following, we present the details of our neural network based method for predicting the optimal parameterization weights λ_{ij} in (4.11). Our method takes as input an unstructured point cloud with parameterized boundary curves. The feature extraction step proceeds as follows. First, the radius graph associated to the unstructured point cloud is computed, and then its line graph, together with appropriate vertex features suitably identified as described in Section 4.2.2, is derived. The line graph is then given as input to our graph convolutional neural network, detailed in Section 4.2.3. The output of the network is a set of parameterization weights λ_{ij}, for any interior point, $i = 1, \dots, n$ and neighbouring point, $j \in N_i$. Those are mapped back to the radius graph, and finally, a system of the form (4.12) is solved to obtain the parametric values.

4.2.2 *Preprocessing and feature extraction*

As a preprocessing step, the input point cloud is normalized by translation and scaling so that it is contained inside the unit cube $[0, 1]^3$. As outlined in Section 4.2.1, the interior points in $\mathcal{P}_I$ are parameterized using neighborhood relationships in a directed graph $\mathcal{G}$, which is the radius graph from $\mathcal{P}_I$ into $\mathcal{P}$, based on the ball neighborhoods (4.10). Therefore, predicting optimal parameterization weights corresponds to predicting edge weights on the directed graph. However, while a number of graph neural network architectures can

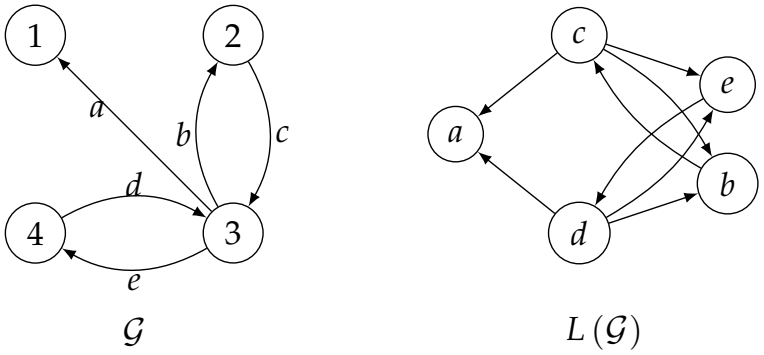


Figure 4.12: Examples of a directed graph $\mathcal{G}$ (left) and its corresponding directed line graph $L(\mathcal{G})$ (right).

incorporate given *edge weights* in their convolution, their predictions live on the vertices of the graph. In order to predict an output on the directed edges of $\mathcal{G}$, we transform $\mathcal{G}$ into a *line graph* $L(\mathcal{G})$ whose vertices are the directed edges of $\mathcal{G}$. The formal definition of the line graph follows.

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

An example of a directed graph $\mathcal{G}$ and its corresponding directed line graph $L(\mathcal{G})$ is shown in Figure 4.12.

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

$$\boldsymbol{e}_{ij} := \left(\boldsymbol{p}_i, \boldsymbol{p}_i - \boldsymbol{p}_j\right)^\top \in \mathbb{R}^6, \tag{4.13}$$

which are attached to the vertices of $L(\mathcal{G})$. This choice of features was suggested in [180], and motivated by the aim to achieve partial translation-invariance.

4.2.3 *Architecture*

The architecture developed for this study is a graph convolutional neural network, characterized by a suitable choice of fast and localized spectral convolutional operators introduced in [42]. The mathematical foundations of spectral

convolutional graph neural networks are rooted in graph *signal* processing and graph Fourier transforms. In view of the convolution theorem [128], spectral convolutions are defined as linear operators that diagonalize self-adjoint operators of the graph in the Fourier (i. e. eigenvector) basis of the spectral space. State-of-the art graph convolutional operators are implemented by means of functional calculus.

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

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

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

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

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

$$\Delta \coloneqq D - W$$

or its normalized form

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

where I_v is the $v \times v$ identity matrix. Let $\{\mu_\ell\}_{\ell=1}^{v}$, $\mu_\ell \in \mathbb{R}_{\geq 0}$ for each ℓ, be the ordered set of eigenvalues for Δ and let $\{\boldsymbol{x}_\ell\}_{\ell=1}^{v}$, $\boldsymbol{x}_\ell \in \mathbb{R}^v$, their associated orthonormal eigenvectors. It follows that for $\boldsymbol{s} \in \mathbb{R}^v$,

$$\Delta \boldsymbol{s} = \sum_{i=1}^{v} \mu_i \langle \boldsymbol{s}, \boldsymbol{x}_i \rangle \boldsymbol{x}_i,$$

where $\langle \cdot, \cdot \rangle$ is an inner product in $\mathbb{R}^v$. In addition, let $g_\theta : \mathbb{R} \to \mathbb{R}$ be a filter function depending on the parameter $\theta \in \mathbb{R}$. We can apply g_θ on Δ, resulting in

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

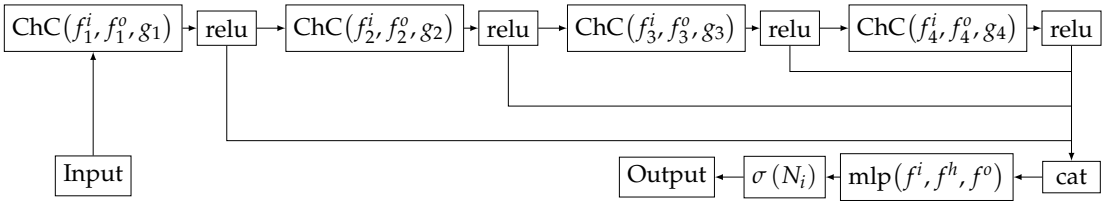


Figure 4.13: PARGCN Architecture design.

For suitable choices of g_θ, its application to Δ, reported in the above equation, has an explicit formulation. In particular, this is the case of [42], where g_θ is defined as a polynomial filter, so that

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

where θ_j are polynomial coefficients. To develop a recursive and fast formulation of spectral filter, the authors in [42] exploit the Chebychev polynomials. During the training phase, described later in Section 4.2.6, optimal values for θ_j will be computed. For further details about spectral convolutional operators and their properties we refer to [42] and [115].

PARGCN architecture

To predict the weights needed for constructing a meshless parameterization of an unorganized point cloud, we design a *sequential* graph convolutional neural network to which we add *shortcut* connections. The layout of the learning architecture is illustrated in Figure 4.13. It consists of the input (Input), 4 spectral convolutional layers (ChC), ReLU activation functions after each convolution (relu), a concatenation layer (cat), an MLP (mlp), composed by 3 fully connected layers, the softmax activation function (σ) and, finally, the output (Output).

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

Input The directed line graph $L(\mathcal{G}) = (\mathcal{V}', \mathcal{E}')$ together with the (line graph) vertex features (4.13) form the input of the neural network. The line graph is built on the radius graph of the unorganized point cloud $\mathcal{P}$, for which we want to compute the parameterization.

ChC The spectral graph convolutional layers are characterized by the Chebychev polynomial filters proposed in [42], based on the normalized graph Laplacian Δ. For each layer $\ell = 1, \dots, 4$, the corresponding convolutional layer ChC$(f_\ell^{in}, f_\ell^{out}, g_\ell)$ is defined by declaring the size of input and output features $f_\ell^{in}, f_\ell^{out}$ and the dimension of the convolving filter g_ℓ. More

precisely, for each $\ell = 1, \dots, 4$, we choose $g_\ell = 2d + 1$, depending on the polynomial bi-degree $\boldsymbol{d} = (d, d)$ used in the loss function in (4.14). As concerns the dimension of the layer features, for the first layer $f_1^{in} = 6$, accordingly to (4.13), whereas we choose $f_1^{out}, f_\ell^{in}, f_\ell^{out} = 64$ for $\ell = 2, \dots, 4$.

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

cat In order to prevent the degradation problem, due to numerical instabilities related to a potentially too high number or learnable parameters [85, 166], we introduce short-cut connections by concatenating the output of each ReLU function, enabling the network to skip the sequence of layers in between.

mlp The content of the concatenation layer is processed by an MLP characterized by three hidden layers: the input layer, the hidden layer, and the output layer, of dimensions $f^{in} = 262$, $f^h = 64$, and $f^{out} = 1$, respectively. Note that f^{in} is a constrained dimension due to the dimension of the previous concatenation layer, f^h is an author's choice, and f^{out} corresponds to the dimension of the output features related to the problem. In this case, we want to predict one weight for each vertex of the line graph $L(\mathcal{G})$, corresponding to each edge of the graph $\mathcal{G}$.

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

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

Output The output of the model is a vector of length $|\mathcal{V}'|$, whose components correspond to the predicted parameterization weights λ_{ij} for $i = 1, \dots, n$ and $j \in N_i$ to be used to assemble and solve the linear system in (4.12). Due to the final softmax activation function σ, they are positive and we have $\sum_{j \in N_i} \lambda_{ij} = 1$, for all $i = 1, \dots n$.

4.2.4 *Loss function*

There are multiple strategies for training the neural network on our data set, consisting of synthetically generated data. One possibility is to compute the exact parameterization weights λ_{ij} from (4.11) and use them as labels, minimizing the MSE of the output weights predicted by our network. We can alternatively solve the linear system (4.11) based on the predicted parameterization weights and minimize the MSE with respect to the exact parameters. These two approaches result in a supervised learning method, as the exact parameterization for the training data needs to be available.

In this Thesis, we instead pursue an *unsupervised* learning approach: from the predicted parameterization weights, we first solve the linear system (4.11) to obtain a suitable parameterization $\mathcal{U}$ as in (1.3) of the input point cloud $\mathcal{P}$ as in (1.1). When training the network, we use this parameterization to fit a tensor product Bézier surface $\boldsymbol{s}$ to the data by solving the linear least-squares problem, see (3.5) in Section 3.1. The loss function is the residual

$$\mathcal{L}(\lambda_{ij}, \mathcal{P}) = \sum_{i=1}^{m} \|\boldsymbol{s}(u_i) - \boldsymbol{p}_i\|_2^2, \tag{4.14}$$

of the least-squares fitting problem corresponding to the parameter $\boldsymbol{u}_i$ that were obtained from the predicted parameterization weights λ_{ij}, by solving the system in (4.11). Note that in the implementation of the method it is necessary to obtain the gradients of the solution to the linear problem (4.11) and the linear least-squares problem, with respect to the input λ_{ij} and $\mathcal{U}$.

One advantage of this unsupervised learning approach is that the network can be trained on arbitrary point clouds, even if the points are not sampled from a tensor product Bézier surface or if no parameterization is known. Moreover, even for training data sampled from tensor product Bézier surfaces, minimizing the loss function (4.14) leads to better experimental results than minimizing the MSE with respect to the parameterization weights or the parameters.

4.2.5 *Shape preserving correction*

From the predicted parameters λ_{ij}, we obtain the parameters $\mathcal{U}$ by solving the linear system (4.12). While we can directly use these parameters for fitting a polynomial surface to the input point cloud by solving the least-squares problem, we also investigate the use of a further correction step inspired by the shape preserving parameterization from [58]. To this end, we first obtain a global

Delaunay triangulation of the planar parameters $\boldsymbol{u}_i$, $i = 1, \dots, m$. Then, for each interior point $\boldsymbol{p}_i$, $i = 1, \dots, n$, we determine the shape preserving weights λ_{ij} with respect to the corresponding neighbours $\boldsymbol{p}_j$ in the planar triangulation, as described in Section 4.2.1. By solving the linear system (4.12) once more based on the neighbourhoods defined by the Delaunay triangulation and the *shape preserving corrected weights*, we obtain the corrected parameterization. In Example 4.2.7 (a), we empirically analyze the difference between the parameters obtained using the predicted parameterization weights and the shape preserving corrected weights.

4.2.6 *Learning meshless parameterization*

In this section, we present the necessary steps for implementing the PARGCN method described in the previous sections. Since our method follows a data-driven approach, it is necessary to obtain a large enough data set, which we generate synthetically, similarly to the procedure described in Section 4.1.4. In what follows, we also summarize the hyperparameters used for the training process.

Data generation

Data-driven methods naturally depend on data, and in particular on their availability, amount, and nature. The network architecture takes as input a directed line graph based on the radius graph of the point cloud. Since there are no public data sets available that are suitable for the parameterization of unorganized point clouds, we generate synthetic data by randomly sampling points from randomly generated polynomial parametric surfaces of bi-degree $\boldsymbol{d} = (d, d)$. The precise algorithm for data generation follows, see Algorithm 6. In particular, as concerns step 1, we generate an initial polynomial tensor product surface with control points $\boldsymbol{c}_j = [c_j^1, c_j^2, c_j^3]^T$, where $[c_j^1, c_j^2]^T$ are chosen as the tensor product Greville abscissae for bi-degree $\boldsymbol{d} = (d, d)$, and c_j^3 are sampled randomly accordingly to the uniform distribution in $[0, 1]$. We then choose a random rotation axis $\boldsymbol{x} \in \mathbb{R}^3$ by sampling each of its components with respect to the random uniform distribution on $[0, 1]$ and a random angle ψ according to the random uniform distribution on $[0, \pi]$. The initial surface is rotated around $\boldsymbol{x} \in [0, 1]^3$ by ψ.

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

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

Note that the surfaces created by this algorithm are graph surfaces of tensor product polynomial functions of arbitrary orientations in space. While this

Algorithm 6: Generation of scattered point clouds.

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

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

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

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

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

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

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

may seem like a restriction of the training data, the network has no problems generalizing to arbitrary surface data, as we will demonstrate in the numerical experiments in Section 4.2.7.

Hyperparameter selection and training

We use Algorithm 6, to generate a training data set of 10.000 point clouds sampled from polynomial parametric surfaces of bi-degree $\boldsymbol{d} = (2,2)$, each consisting of $m = 264$ points of which $n = 200$ were sampled from the surface interior. Before training, we performed the preprocessing and feature extraction steps described in Section 4.2.2. For computing the radius graph, we set the radius to $r = 0.2$ and se 32 as the maximum number of neighbours for each vertex in the graph, to reduce the computational complexity and the size of the line graph computed from the radius graph. This choice of hyperparameters leads to a preprocessed training set of size 29 GB.

We additionally generated a validation data set of 2.500 (7.2 GB) preprocessed point clouds with the same properties as the training set in order to choose the final model. The architecture we design has 80.641 learnable parameters, whose (optimal) values are set by solving the stochastic optimization problem using the Adam optimizer with learning rate $1e-3$ and momentum 0.9. In order to compute the loss function described in Section 4.2.4, we solve the polynomial least-squares problem with bi-degree $\boldsymbol{d} = (2,2)$, namely the same polynomial degree that characterizes the training and validation data. According to the learning curves in Figure 4.14, showing the training and validation errors across the training epochs, we choose the learning model corresponding to the 50th epoch. The performance on the test set is reported in Example 4.2.7 (a).

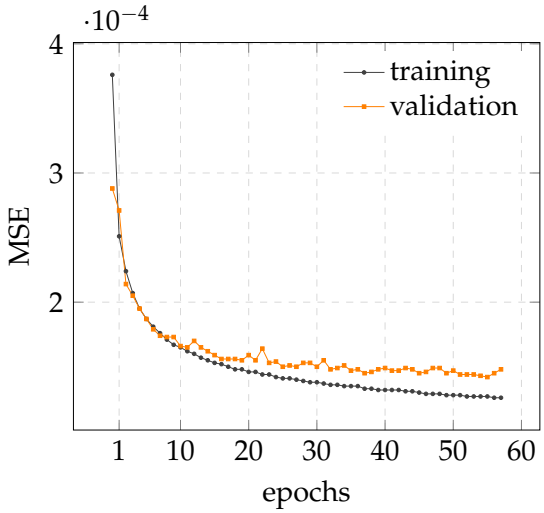


Figure 4.14: Training and validation error converngence rate across the epochs.

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

4.2.7 *Numerical results for PARGCN*

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

The quality of the proposed meshless learning parameterization model PARGCN is evaluated in terms of geometric model reconstruction accuracy based on MSE and the one-sided Direct Hausdorff Distance (DHD). For each test, we compare the error measures MSE and DHD of the geometric models obtained with PARGCN and the parameterization choices introduced in [62] and briefly discussed in Section 4.2.1 (UNIF, RECD and LPSP). We remind that after the preprocessing step, each point cloud is contained in the unit cube $[0,1]^3$; hence, a few digits of difference in the error measures correspond to a significant gain in accuracy.

The numerical experiments reported in Example 4.2.7 (a), show the performance of the predicted meshless parameterization on a test set of the same nature as the training and validation sets described in Section 4.2.6. In Ex-

ample 4.2.7 (b), we show the generalization capabilities of the learning model on polynomial data sets of varying bi-degree $\boldsymbol{d} = (d, d)$, together with its robustness when tested on noisy data configurations. Subsequently, in Example 4.2.7 (c), we illustrate the generalization capability of PARGCN with respect to non-polynomial point clouds of arbitrary dimension, together with its suitability for polynomial least-squares fitting. Finally, in Example 4.2.7 (d), we numerically demonstrate that the trained meshless parameterization model is capable of properly generalizing to tensor product B-spline fitting of arbitrary point clouds.

Example 4.2.7 (a)

Test error

In this experiment, we analyze the quality of the learning model by its evaluation on the so called *test set*, i. e. a data set of the same nature as training and validation sets, but whose items have never been seen by the network during the training phase. Furthermore, we motivate the choice of introducing the shape preserving correction step described in Section 4.2.5.

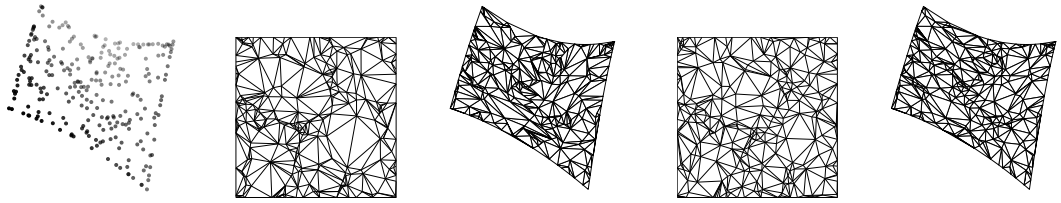
More precisely, we generate 100 unorganized point clouds of $m = 264$ points, with $n = 200$ interior points as detailed in Section 4.2.6, and we preprocess them as described in Section 4.2.2, building their radius graph for a fixed radius $r = 0.2$ and allowing each interior point to have at most 32 neighbours.

After evaluating the network on this data set and solving the system in Equation (4.12), without performing the shape preserving correction step, the MSE resulting from the polynomial least-squares fitting with bi-degree $\boldsymbol{d} = (2, 2)$, averaged on 100 point clouds, is $1.59\text{e} - 4$ and the DHD, averaged on the whole data set, is $1.90\text{e} - 2$. These results prove the transferability of the network with respect to unseen data of the same complexity as the training and validation sets, since they are in line with the values obtained during the training phase, already discussed in Figure 4.14. By adding the shape preserving correction step explained in Section 4.2.5, we obtain MSE and DHD equal to $4.45\text{e} - 5$ and $1.12\text{e} - 2$, respectively. This improvement motivates us to include the shape preserving correction as the final step in the PARGCN method. Figure 4.15 shows the Delaunay triangulation for the parametric and spatial domains built from the PARGCN parameterizations without (b-c) or with (d-e) the parameter correction step for an item from our test set.

Example 4.2.7 (b)

Polynomial approximation of synthetic data

In this section, we numerically prove the generalization capability of the proposed learning model with respect to the approximation of data characterized by



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

Figure 4.15: An input point cloud (a) and the Delaunay triangulations for the parametric (b,d) and spatial (c,e) domain related to the PARGCN parameterization obtained with (d-e) or without (b-c) the shape preserving correction step for Example 4.2.7 (a).

different attributes. More precisely, we consider polynomial fitting for different bi-degrees, and the robustness of the model in presence of noise.

We evaluate the PARGCN model on unorganized point clouds sampled from polynomial surfaces of various bi-degrees. We generate 100 unorganized point clouds by sampling $m = 264$ points, of which $n = 200$ are interior points and 64 are boundary points, from tensor product polynomial surfaces of bi-degree $\boldsymbol{d} = (d, d)$ for $d = 2, 3, 4, 5$. In addition, we generate a test set similar to the previous one but corrupted with random Gaussian noise by a factor $\epsilon = 1\mathrm{e} - 2$. For each point cloud we perform the feature extraction described in Section 4.2.2, setting the radius to $r = 0.2$ and allowing each interior point to have at most 64 neighbours. Finally we run the least-squares fitting scheme for the corresponding polynomial bi-degree $\boldsymbol{d} = (d, d)$.

The results for exact and noisy data are shown in Table 4.6. For each bi-degree and for each parameterization methods, we report the MSE and the DHD obtained on the two test sets. The best approximation results are highlighted in bold. We observe that the accuracy gained by PARGCN with respect to LPSP when considering the MSE, is between 52% $(d = 3)$ and 70% $(d = 2)$ on exact data. For noisy data, PARGCN is able to outperform the LPSP method gaining on average for each degree 41% of accuracy with respect to MSE. The metric values for PARGCN prove that performing the training with a fixed polynomial degree does not induce a bias in the final learning model. Furthermore, PARGCN's results are suitable for solving the parameterization problem of scattered data for a variety of degrees being in addition robust to noise.

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

Table 4.6: Exact and noisy data polynomial least squares fitting for different bi-degree $\boldsymbol{d} = (d, d)$ with $d = 2, 3, 4, 5$. Average MSE and DHD on 100 point clouds with $m = 264$ points, of which $n = 200$ interiors, and a fixed number of maximum neighbours 64, parameterized with UNIF, RECD, LPSP, and PARGCN parameterization weights for Example 4.2.7 (b).

Example 4.2.7 (c)

Polynomial approximation of a ship hull

In this example we show the capabilities of the PARGCN model to generalize with respect to data sets of different nature and size. We consider the point cloud shown in Figure 4.16 obtained by randomly sampling $m = 596$ points (500 interior and 96 boundary points) from a B-spline model of a ship hull. We then preprocess the data as described in Section 4.2.2 building a radius graph with $r = 0.1$ while allowing a maximum number of 72 neighbours. Finally,

Figure 4.16: Point cloud sampled from the model of a ship hull.

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

Table 4.7: Polynomial least-squares fitting of bi-degree for $\boldsymbol{d} = (d, d)$ with $d = 2, 3, 4, 5$ for Example 4.2.7 (c) parameterized with UNIF, RECD, LPSP, and PARGCN parameterization weights.

we parameterize the input data with the RECD, LPSP, and PARGCN methods and compute the least-squares tensor product polynomial approximation of bi-degree $\boldsymbol{d} = (d, d)$ with $d = 2, 3, 4, 5$. The average MSE and DHD errors are reported in Table 4.7, whereas the polynomial reconstructed models are shown in Figure 4.17. The best approximation error values are highlighted in bold. While the approximations obtained with RECD parameterization show a significant mesh distortion, the LPSP and PARGCN parameterizations lead to effective reconstructions, always with a reduced MSE for the second one. When comparing the MSE for PARGCN and LPSP, we register an average gain of 49% accuracy for the first method with respect to the second one.

Example 4.2.7 (d)

B-spline approximation of a face

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

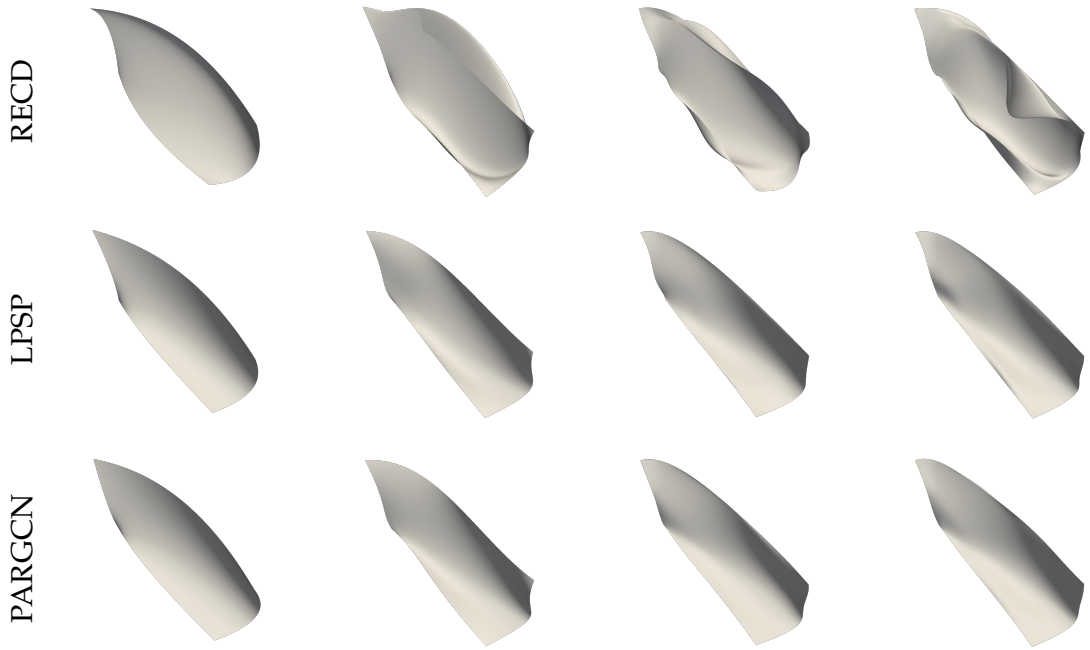


Figure 4.17: Polynomial least-squares approximation of the ship-hull point cloud shown in Figure 4.16 for RECD (top), LPSP (center), and PARGCN (bottom) parameterization weights and bi-degree (d, d) with $d = 2, 3, 4, 5$ from left to right for Example 4.2.7 (c).

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

We process an unorganized point cloud consisting of $m = 543$ points (458 interior points and 85 boundary points) sampled from a face model; see Figure 4.19 (left). In this case, we build a radius graph with $r = 0.2$, allowing a maximum number of 64 neighbours for each interior point. The parametric values in $[0, 1]^2$, resulting from different parameterization methods, namely RECD, LPSP, and PARGCN, together with the resulting triangulations, are shown in Figure 4.18. Note that the parametric values for RECD and LPSP are more clustered and therefore lead to bigger voids (i. e. lack of data) in the parametric domain. The low quality results obtained with RECD and LPSP in this example are visible also by analyzing the triangulations in Figure 4.18 (left and center). This leads to mesh distortion phenomena and the presence of artifacts when these parameterizations are considered for polynomial and spline surface reconstruction. The MSE for polynomial approximation for different bi-degrees $\boldsymbol{d} = (d, d)$, with $d = 2, 3, 4, 5$, is shown in the left column of Table 4.8, while the right column

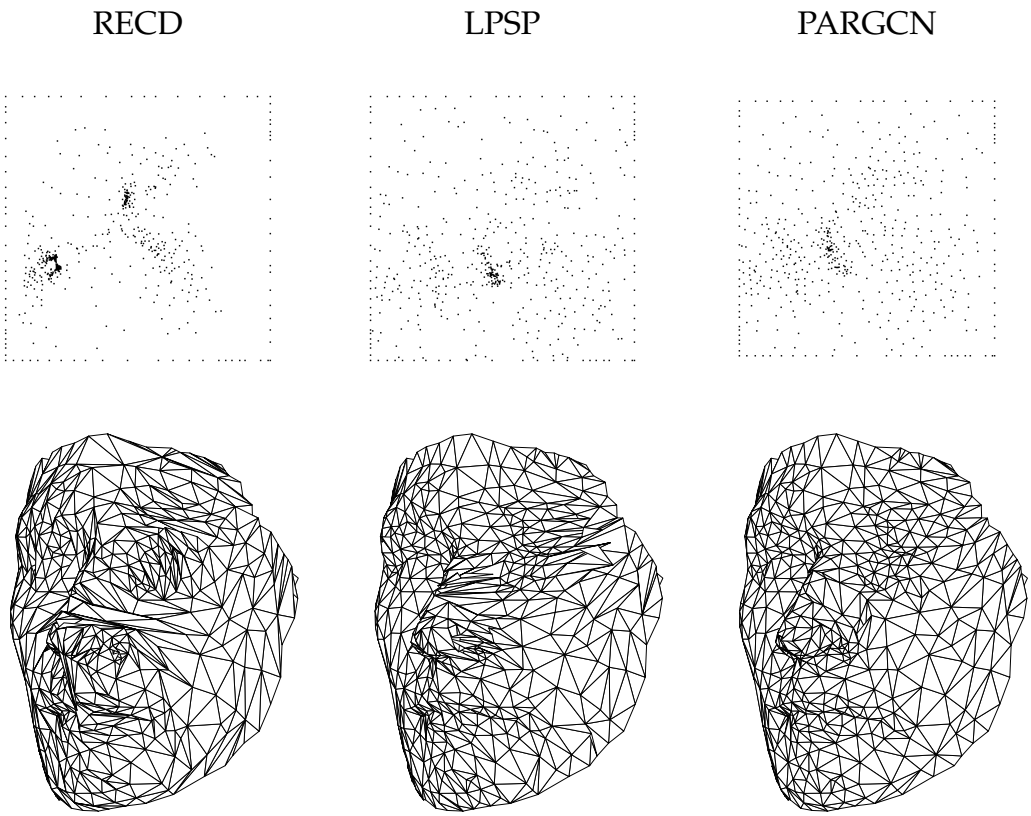


Figure 4.18: Parametric values on $[0,1]^2$ (top) for the face example, and 3D Delaunay triangulations for different methods (bottom) for Example 4.2.7 (d).

reports the MSE obtained when performing a (penalized) least-squares fitting with tensor product B-splines characterized by 4 levels of uniform refinement. Across all degrees, when PARGCN is used in order to perform polynomial least-squares approximation, we are able to gain on average 32% of accuracy with respect to LPSP for MSE. Moving to the B-spline setting, the MSE decreases due to the growth of degrees of freedom for all the methods, but PARGCN registers a gain in accuracy of 70% on average for each degree. Figure 4.19 shows the geometric model obtained by computing the (penalized) B-spline least-squares fitting with two (center) and four (right) levels of uniform refinement when the PARGCN parameterization model is considered. Moreover, Figure 4.20 illustrates the parameter distribution of the B-spline surface resulting from PARGCN and 4 levels of uniform refinement.

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

Table 4.8: Polynomial least-squares fitting and B-Spline fitting after 4 steps of uniform refinement, with bi-degree $\boldsymbol{d} = (d, d)$ for $d = 2, 3, 4, 5$ in Example 4.2.7 (d), with $m = 543$ points, of which $n = 458$ interiors, parameterized with UNIF, RECD, LPSP, and PARGCN parameterization weights.

4.2.8 *Beyond PARGCN*

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

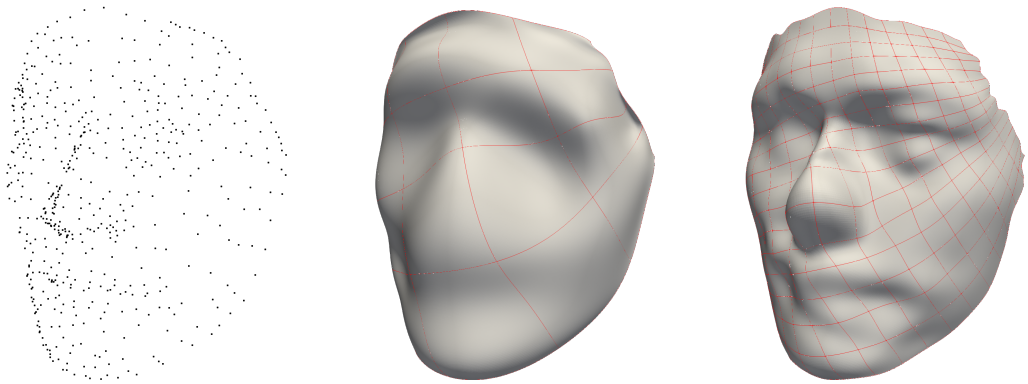


Figure 4.19: Point cloud considered in Example 4.2.7 (d) (left) and penalized B-spline least-squares fitting of bi-degree $\boldsymbol{d} = (5, 5)$ obtained with PARGCN parameterization for 2 (center) and 4 (right) levels of uniform refinement.

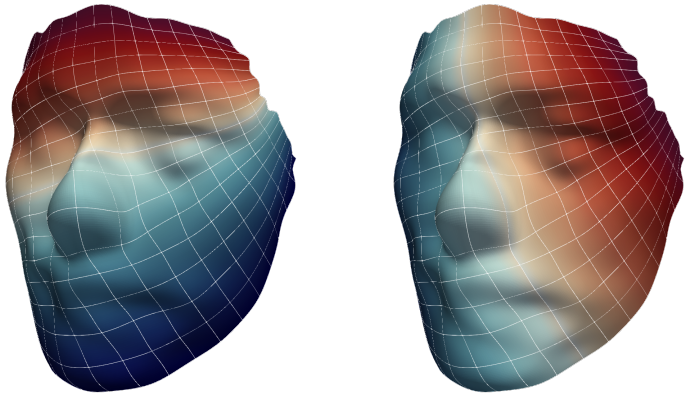


Figure 4.20: Parameter distribution along the two parametric dimensions for the tensor product B-spline model with 4 levels of uniform refinement in Example 4.2.7 (d).

A key step of our PARGCN method is the novel local feature extraction scheme by means of a directed line graph that is computed from the radius graph of an unstructured point cloud, see Section 4.2.2. This enabled us to design a new network architecture that has the ability to predict barycentric parameterization weights for each edge in the radius graph, which are then used to obtain a suitable parameterization.

On the other hand, the computation of the line-graph $L(\mathcal{G})$ as in Definition 12 represents the bottleneck of this approach in terms of computational time and memory. In particular, for point clouds with a high number of vertices ($> 10^3$), the number of edge connections can potentially be very high, and therefore,

the dimension of the dual graph drastically increases. This led us to work with datasets of the magnitude of hundreds of data points. In particular, we report the dimension of the training and validation sets in Section 4.2.6: to store 12.500 point clouds, each consisting of 264 items in $\mathbb{R}^3$, and their radius graph and line graph, about 36.2GB are needed. In addition, a crucial role is played by the choice of the radius r to define the graph $\mathcal{G}$. As highlighted in Section 4.2.1, the existence, uniqueness, and suitability of the solution of (4.12), i. e. of a parameterization for the input point cloud, depend on the choice of r. If r is *relatively* too small or too big, a suitable parameterization for the input point cloud at hand does not exist. The fine-tuning of r is therefore fundamental and unavoidable for the proposed method. Finally, post-processing steps, e. g., the solution of the linear system in (4.12), are required in order to recover the final parameterization after evaluating the network.

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

4.3 BIDGCN: PARAMETERIZATION OF SCATTERED DATA WITH BOUNDARY INFORMATION

Most current graph neural network architectures are classified as *message passing neural networks* [75], meaning that hidden states at each vertex are computed by aggregating the output of message functions applied to the features of the vertex and its neighbours connected by an edge. GCNs that are employed for processing discrete surface data have mostly been applied so far to *closed* surfaces, i. e. surfaces without boundary. This simplifies the design of suitable graph convolutional operators significantly, since all vertices of the discrete manifold or point cloud can be handled in the same way and no explicit distinction between interior and boundary vertices needs to be made. However, in many applications in geometric modelling, geometry processing, and numerical analysis, the input geometries are given as discrete surfaces *with* boundary. For a variety of problems, *boundary conditions* are imposed on the corresponding

vertices, and often the solution to a problem is only uniquely determined up to the boundary conditions. For example, this is the case for boundary value problems of elliptic partial differential equations on surfaces, as well as for the problem of scattered point cloud parameterization. In order to apply deep learning-based methods to this kind of problems, it is therefore necessary to devise a network architecture that takes into account the boundary conditions in addition to the standard vertex features. Since boundary conditions can be regarded as additional features that are defined only on the boundary vertices but not on the interior vertices, this means that such a network architecture needs to be able to handle data with varying feature dimensions.

In this Section, we propose the new model, called BIDGCN, for processing scattered point clouds. The key idea of BIDGCN is to treat boundary conditions as additional features at the boundary vertices of the point cloud. These features are propagated into the whole point cloud by a novel graph convolution operator that contains two separate trainable message functions: the first one for edges between interior and boundary vertices, and the second one for edges between interior vertices. In the subsequent hidden layers, the information stemming from the boundary conditions is further processed and used to predict the solution for the problem at hand. The two main properties of the proposed network layer are: (i) the ability to incorporate boundary conditions in its prediction (due to the boundary input layer) (ii) the dynamic prediction of the graph used for message passing (due to the dynamic edge convolution approach as in [180]).

As a use case for our new network architecture, we apply it to the problem of scattered point cloud parameterization. In particular, the new BIDGCN parameterization method overcomes the limitations of the classical methods and of the PARGCN method, discussed in Section 4.2 in terms of efficiency, robustness, and sensitivity to parameter-dependent graph connectivity.

In our approach, we also assume that the boundary is already parameterized, as in the standard methods, see Section 4.2.1. We then use a network architecture based on our new BIDGCN to predict the parameterization of the interior vertices. Using our boundary informed dynamic graph convolutional network leads to a number of advantages compared to the classical methods for scattered point cloud parameterization:

1. *Computational efficiency*. After training the network its evaluation is computationally much more efficient than applying the classical methods in [62] and the learning-based method in [71]. In particular, once the train-

ing 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., [62, 71], it is necessary to construct a graph by suitably choosing the local neighbourhoods of each interior vertex, our method automatically predicts a suitable graph without the need of free parameter selection. Note that an improper graph selection can even lead to failure of classic parameterization algorithms due to non-invertible linear systems in case of sparse neighbourhoods.

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

4.3.1 *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 [180]. We first briefly describe the original dynamic edge convolution operator, and then we present our new boundary informed dynamic convolutional layer and the additional layers of the network architecture.

Dynamic edge convolution

We assume to be given a point cloud $\mathcal{P}$ together with vertex features $\boldsymbol{x}_i \in \mathbb{R}^p$, where $p \in \mathbb{N}$ is the input dimension. The dynamic edge convolution operator defined in [180] computes new features $\boldsymbol{y}_i \in \mathbb{R}^q$ for all points in $\mathcal{P}$, where $q \in \mathbb{N}$ is the output dimension. To this end, first the k-nearest neighbour graph $\mathcal{G}$ is computed based on the input features. This is a directed graph where the existence of a directed edge (j, i) implies that j is among the k nearest neighbours of i with respect to the Euclidean distance $\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_2$ of the input

features. In the next step, edge features are computed for each directed edge in $\mathcal{G}$: for all $(j,i) \in \mathcal{G}$

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

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

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

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

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

where $\square$ computes the mean value. In a *dynamic* edge convolution network, the k-nearest neighbour graph is recomputed after each layer with respect to the new features $\boldsymbol{x}_i'$, thereby allowing the network to pass information arbitrarily fast across the point cloud. The dynamic approach enables the network to automatically predict a suitable graph for message passing instead of using a fixed one. This implies that information can travel arbitrarily far in each layer, as it is determined by the training process. In [180] 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 [180] the k-nearest-neighbour graph was used, to emphasize locality we propose to use the radius graph also in the hidden layers. In particular, instead of specifying the number of neighbours k, we specify a radius $r > 0$ and add directed edges (i,j) and (j,i) for all $i,j \in V$ such that

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

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

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 and propagates the boundary conditions into the new features of the interior points. The output of this layer consists of all *interior* points together with new vertex features. We name the resulting

neural network architecture Boundary Informed Dynamic Graph Convolutional neural Network (BIDGCN).

Assume that the input point cloud is decomposed like in Section 4.2.1 as $\mathcal{P} = \mathcal{P}_I \dot{\cup} \mathcal{P}_B$ into interior and boundary points and that each vertex $\boldsymbol{p}_i \in \mathcal{P}_I$ comes with features $\boldsymbol{x}_i \in \mathbb{R}^p$ and every vertex $\boldsymbol{p}_j \in \mathcal{P}_B$ comes with features $(\boldsymbol{x}_j, \boldsymbol{u}_j) \in \mathbb{R}^p \times \mathbb{R}^s$, were we regard $\boldsymbol{u}_j \in \mathbb{R}^s$ as boundary conditions.

We first compute two different radius graphs

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

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

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

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

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

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

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

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

where h is a feed-forward neural network

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

with output feature size q and learnable weights Θ.

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

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

where g is another feed-forward neural network

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

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

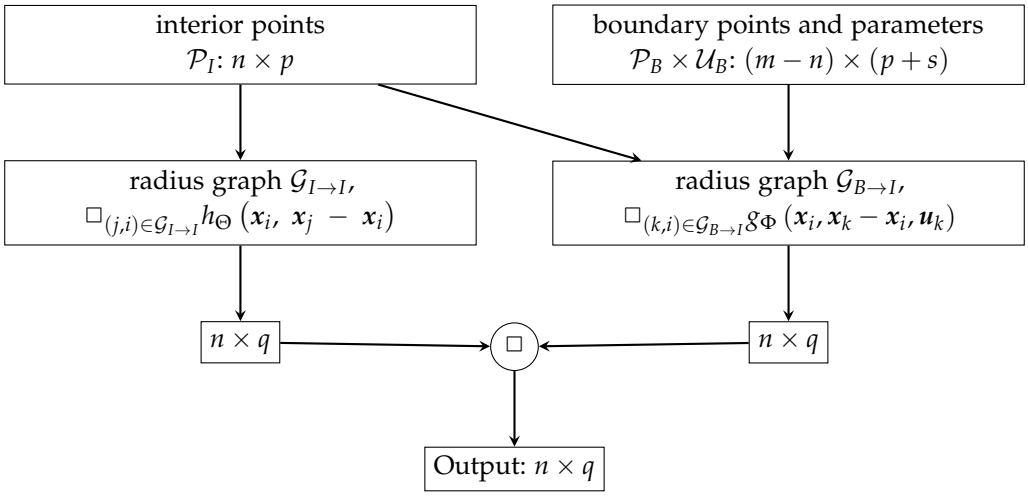


Figure 4.21: Boundary informed input layer.

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

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

where the aggregation operator $\square$ can be the sum, the mean or the component-wise maximum value. The output of the layer consists of features of dimension q for the interior point cloud $\mathcal{P}_I$. During training Θ and Φ are optimized simultaneously. The flow of the input layer is depicted in Figure 4.21 and its application to an interior vertex is illustrated in Figure 4.22.

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

Hidden layers

The novel BIDGCN layer gives as output new features in $\mathbb{R}^q$ for the interior point cloud $\mathcal{P}_I$. To further process these features, we proceed like in the original Dynamic Graph Convolutional Neural Network (DGCNN) by computing a new graph for $\mathcal{P}_I$ based on the new features $\boldsymbol{x}_i' \in \mathbb{R}^m$. As a slight modification, while in [180] the k nearest-neighbour graph was used, we propose to use the radius graph also in the hidden layers. This makes BIDGCN architecture more independent on the sampling density of the point cloud. Since the number

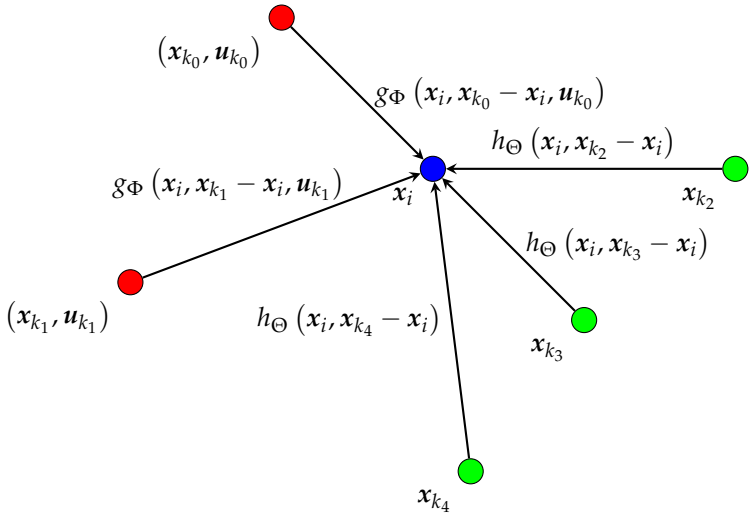


Figure 4.22: Edge contributions of the input layer for x_i (blue) computed from the features at the neighbouring boundary vertices (red) and interior vertices (green).

of neighbours of the vertices is not constant, we use the mean value as the aggregation of the edge features, also motivated by the goal of achieving independence of the sampling density.

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

4.3.2 *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. The neural network should then approximate an operator that assigns to each sufficiently large set of input features a parameterization that is optimal for fitting a surface $s : [0,1]^2 \to \mathbb{R}^3$ to the input points. Without fixing some of the parameter values a priori, this operator is not uniquely defined, making it difficult to directly train a neural network for this task. Therefore, the parameters u_j for the boundary points needs to be suitably computed. This determines a well-defined

specific parameterization operator that takes as input the positions of all points, together with the boundary parameterization, and gives as output the unique optimal parameterization of the interior points. We train our neural network to approximate this operator.

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

BIDGCN architecture

We design a neural network based on the new boundary informed input layer described in Section 4.3.1. The architecture of our BIDGCN is shown in Figure 4.23, and it has an overall number of trainable parameters of 192,130. It consists of the boundary informed input layer (BIDGC layer), four hidden dynamic edge convolution layers (DGC layers), and an MLP as output layer.

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

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

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

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

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

After each hidden layer, the ReLU activation function is applied. Finally, after the output layer, the sigmoid activation function is applied to enforce that the predicted parameters lie in $[0, 1]^2$. Since all layers in BIDGCN are convolutional

except for the last layer that is applied vertex-wise, the network can be applied to any point cloud, independent of its size.

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

4.3.3 *Loss function*

To train the neural network for the prediction of optimal fitting parameters, we follow an unsupervised strategy and use the predicted parameters for the interior points $\mathcal{U}_I$ as well as the prescribed parameters for the boundary points $\mathcal{U}_B$ to fit a bi-quadratic polynomial surface to the point cloud. More precisely, we assemble the collocation matrix, as defined in (3.4), Section 3.1, i. e.

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

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

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

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

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

$$\min_{\boldsymbol{c}\in\mathbb{R}^{9\times 3}} \|A\boldsymbol{c} - \boldsymbol{b}\|_2^2 \tag{4.15}$$

using QR decomposition and keeping track of the gradients with respect to the learnable weights of the neural network. The loss for the predicted parameterization of the interior points is the residual of (4.15).

4.3.4 *Learning boundary informed parameterization*

In order to train the network for parameterizing scattered point cloud data, we generate a data set consisting of 100,000 point clouds by following Algorithm 6.

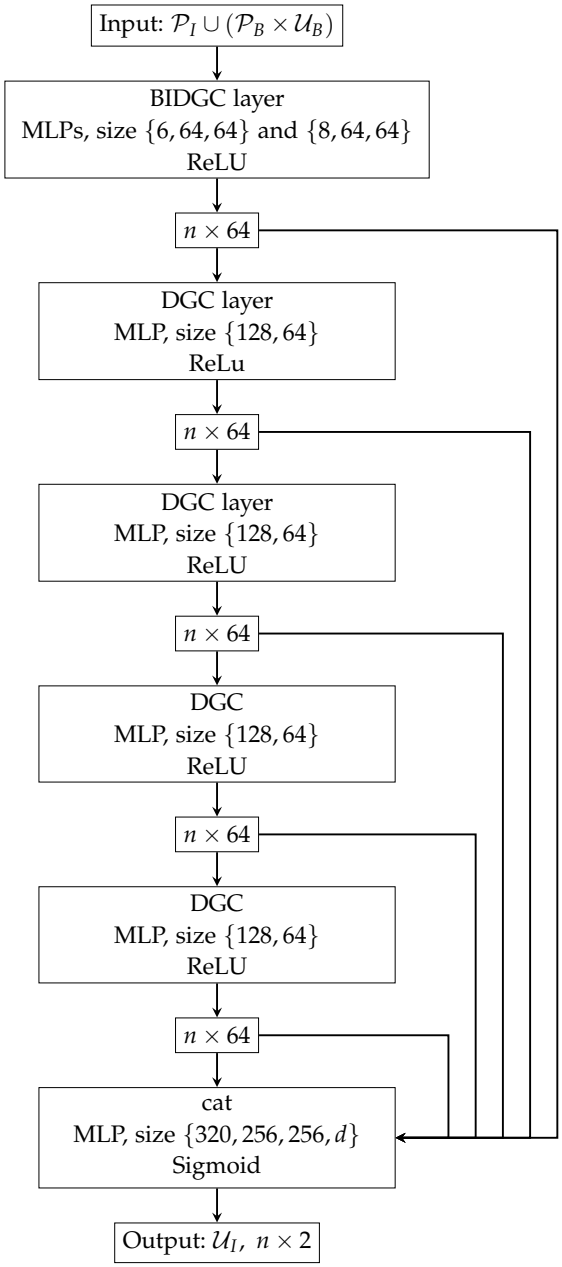


Figure 4.23: BIDGCN architecture for $p = 3$ (dimension of the point cloud) and $s = 2$ (parametric dimension).

The points are sampled from bi-quadratic tensor-product Bézier surfaces whose control points $\boldsymbol{c}_{ij}$ are randomly sampled from

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

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

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

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

4.3.5 *Numerical results for BIDGCN*

We implemented our method using the PyTorch [143] and PyG [57] libraries. The code for training and testing is available at `https://github.com/felixfeliz/BIDGCN`.

We trained the network architecture described in Section 4.3.2 and visualized in Figure 4.23 on our data set consisting of 100.000 point clouds sampled from bi-quadratic surfaces according to Section 4.3.4. Referring to Section 4.3.3, the loss function during training is the fitting error. We trained the network using stochastic gradient descent, starting with learning rate 0.1 and decreasing whenever the loss plateaus.

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

In order to test the performance of our method, we generate several test data sets as described in Section 4.3.4. Each data set consists of 100 point clouds sampled from tensor-product polynomial surfaces. We generated distinct test data sets by varying the polynomial degree, the amount of noise and the number of sample points. In particular, we used polynomial bi-degrees 2, 3, 4 and 5, Gaussian noise with standard deviation $5e-3$, $1e-2$ and $5e-2$, and

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

Table 4.9: Summary of hyperparameters.

sample sizes between 200 and 2000 inner points. The number of boundary points was always set so that the ratio between inner points and boundary points is equivalent to the one of a uniform mesh. Moreover, we also consider real-world data sets.

We start in Example 4.3.5 (a) with an ablation study of our new boundary informed layer to demonstrate its fundamental role in tackling the parameterization problem. We then show the comparison of the proposed method with the three standard approaches described in Section 4.2.1 in Example 4.3.5 (b)–4.3.5 (e).

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 4.9.

Example 4.3.5 (a)

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

In our first experiment, we compare the performance of BIDGCN with two other learning methods based on graph neural networks. The first method we compare with is the standard dynamic graph convolutional neural network proposed in [180]. In order to ensure a fair comparison, we trained a network with the same architecture as our BIDGCN shown in Figure 4.23, 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 4.3.4. In our plots, we will denote this method by DGCNN. The second learning-based method that we compare with is the PARGCN method proposed in [71] and described in Section 4.2. We recall that PARGCN is based on the standard parameterization methods but it uses a graph convolutional neural network to predict parameterization weights instead of determining them heuristically. Since the parameterization weights are defined edge-wise, PARGCN generates a *line graph* of the initial radius graph, which can be costly. Moreover, it applies a post-processing step called *shape-preserving correction*, which is similar to the shape-preserving parameterization described in Section 4.2.1. PARGCN was trained on a dataset consisting of point clouds of size 200, sampled from randomly generated bi-quadratic surfaces.

We first apply the three methods to a test data set consisting of point clouds sampled from bi-quadratic surfaces with sample size varying between 200 and 2000. Note that while BIDGCN was trained only on point clouds of size 1000, it is important to ensure that it gives good results for point clouds of any size. We use each predicted parameterization to fit the input point cloud with a bi-quadratic surface and compute the MSE of the approximation. The computation time as well as the fitting MSE is shown in the left column of Figure 4.24. All the plots are semi-log plots with a logarithmic scale used for the time and error axes.

Our first observation is that the MSE resulting from DGCNN is prohibitively large while the network based on our novel BIDGCN input layer results in a very good approximation. This is expected since, as described earlier, the parameterization problem cannot be solved without taking into account the parameterization of the boundary curves and a boundary-informed input layer is therefore necessary. This ablation study demonstrates this fact and moreover shows that our novel BIDGCN layer is indeed able to correctly process the given boundary information. In order to better visualize the difference of the predicted parameterizations of BIDGCN and DGCNN, we plot two examples from our test data set in Figure 4.25. We observe that the parameterization predicted by DGCNN contains large voids and is far from the original. This is expected, since the optimal parameterization is not uniquely defined by the positional vertex features only. On the other hand, BIDGCN correctly takes the boundary conditions into account and predicts parameters that are very close to the original ones, leading to a much better approximation with a bi-quadratic surface.

A second observation from Figure 4.24 is that the performance of BIDGCN does not depend significantly on the size of the scattered data set, even if the network was trained exclusively on data with 1000 points per point cloud.

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

Measured real-world data is always subjected to noise. For this reason, we study the behaviour of BIDGCN, DGCNN, and PARGCN when applied to noisy data. In particular, we evaluate all methods on 100 point clouds sampled from bi-quadratic surfaces with added Gaussian noise of standard deviation $5e-3$ and $1e-2$. In the middle and right columns of Figure 4.24 we show the MSE and computation time when applying the methods to noisy data with Gaussian noise of standard deviation $5e-3$ and $1e-2$, respectively. As in the non-noisy case, the plain DGCNN does not result in acceptable fitting errors in the presence of noise. We observe that BIDGCN is more robust than PARGCN with respect to noise and results in better approximations for larger point clouds while needing much smaller amount of computation time. In particular, this means that BIDGCN is able to predict good parameterizations even for data that it is not of the same class as the data that it was trained on.

Example 4.3.5 (b)

Comparison with standard methods on exact and noisy data

In this example, we study the performance of our neural network when applied to data from the same class as the training data, i. e. data sampled from bi-quadratic surfaces. As a benchmark, we use the three parameterization methods presented in Section 4.2.1: parameterization with UNIF, RECD, and LPSP weights. For these methods, we choose the radius for defining the local neighbourhoods adaptively depending on the number of points per point cloud as

$$r = \frac{3}{\sqrt{m}}. \tag{4.16}$$

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

In order to compare the methods, we evaluate them on 100 point clouds and report the MSE as well as the total computational time needed to parameterize

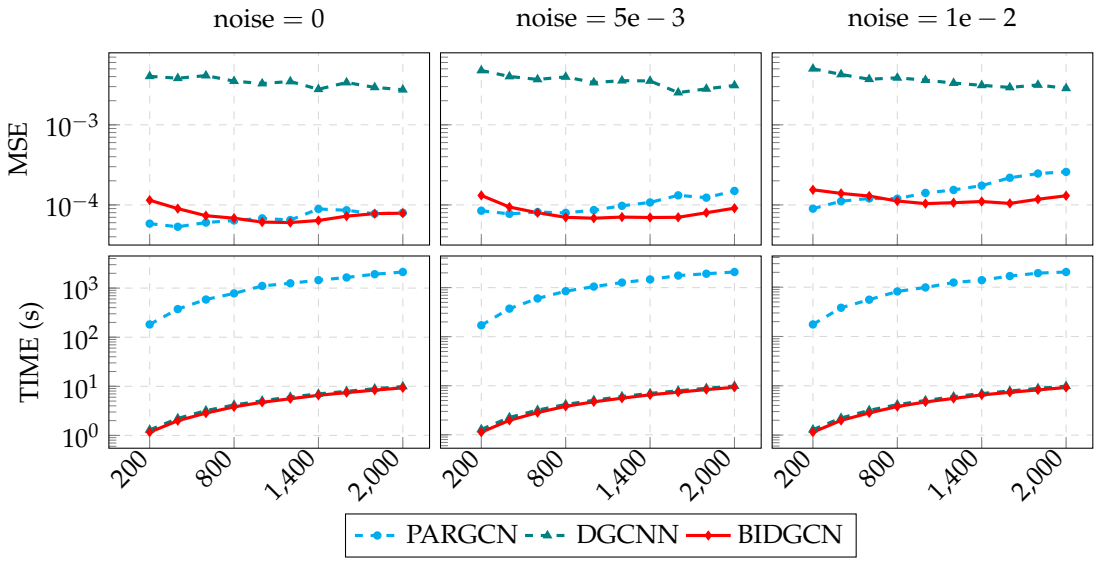


Figure 4.24: Comparison between learning parameterization methods of Example 4.3.5 (a). MSE (top) and computation time (bottom) when applying the three learning-base methods BIDGCN, DGCNN, and PARGCN to point clouds of different sizes sampled from 100 bi-quadratic surfaces without noise (left), with Gaussian noise of standard deviation 5e − 3 (middle) and with Gaussian noise of standard deviation 1e − 2 (right).

and approximate all 100 point clouds. In particular, we study the dependence of accuracy and computation time on the number of points in each point cloud. Note that while BIDGCN was trained only on point clouds of size 1000, it is important to ensure that it gives good results for point clouds of any size. The left column of Figure 4.26 shows the comparison of the methods on point clouds sampled from surfaces that were generated in the same way as the training data set for our neural network, see Section 4.3.4. All the plots are semi-log plots, with a logarithmic scale used for the time and error axes. We observe that the MSEs produced by our method are much smaller than the ones resulting from the UNIF and RECD parameterizations. On the other hand, while the accuracy of our method on this synthetic data is similar to that of LPSP parameterization, the computational time of the neural network-based method is much lower than the time needed for all three other methods. In particular, LPSP is very costly, with a computational time that is about two orders of magnitude higher than the one of BIDGCN.

We perform the comparison of BIDGCN with the previously considered standard methods on noisy data. The middle column of Figure 4.26 shows the

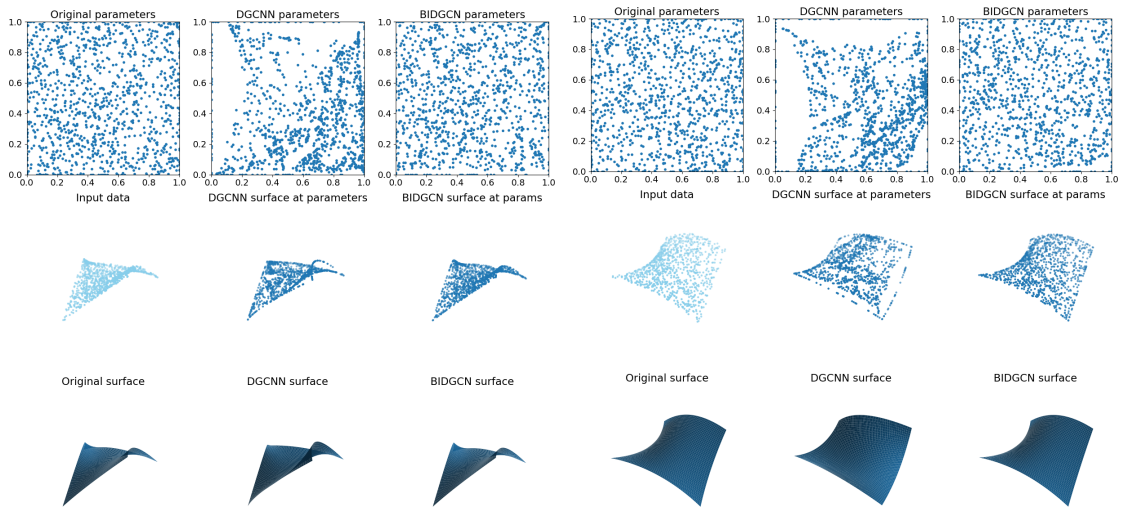


Figure 4.25: Comparison of the BIDGCN network with a pure DGCNN trained for the point parameterization problem considered in Example 4.3.5 (a). Top row: Original parameters and parameters predicted by the networks. Middle row: Input data and the evaluation of the fitted surfaces at the predicted parameters. Bottom row: The original surface and the fitted bi-quadratic surfaces.

behaviour of all four methods when applied to data with Gaussian noise of standard deviation $5e-3$, while the right column of Figure 4.26 shows their behaviour on surfaces with Gaussian noise of standard deviation $1e-2$. We observe that when applied to noisy data, BIDGCN performs much better than the three other methods.

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

A further advantage of BIDGCN is that it is very robust with respect to the presence of noise, even if the noise becomes very large. For the standard methods suitably choosing the radius that determines the local neighbourhood becomes near impossible when the data is non-regularly distributed. For a fixed choice of radius or our simple adaptive choice (4.16), this means that these methods often fail due to neighbourhoods that are too small. In particular, LPSP needs enough points in each neighbourhood to generate a Delaunay triangulation. Figure 4.27 (left) shows how many surfaces out of 100 surfaces with Gaussian noise of standard deviation $5e-2$ could be successfully parameterized by the four methods. We observe that BIDGCN was always successful, while the number

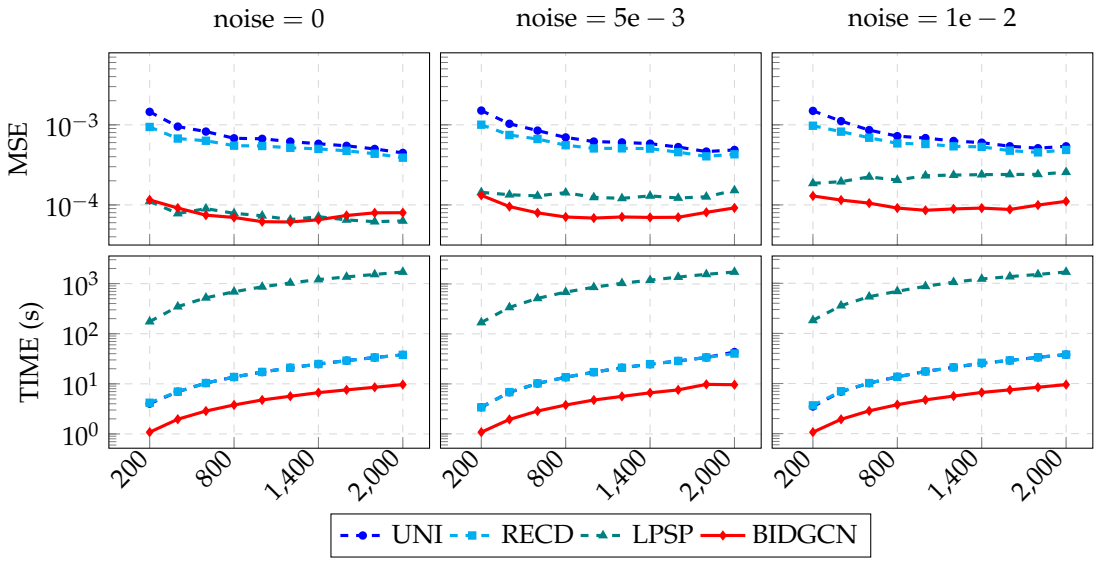


Figure 4.26: Comparison between BIDGCN and standard meshless parameterization methods of Example 4.3.5 (b). MSE (top) and computation time (bottom) when applying the four methods to point clouds of different sizes sampled from 100 bi-quadratic surfaces from the same class as the training data set without noise (left), with Gaussian noise of standard deviation 5e − 3 (middle) and with Gaussian noise of standard deviation 1e − 2 (right).

of surfaces that could be parameterized using LPSP strongly decreases with the number of points per point cloud. One possible way to tackle this problem could be by suitably choosing a different radius r_i for each point $i \in \mathcal{P}$; however, there is no obvious way how to realize such an adaptive local choice in a robust way. Moreover, a local choice, if one exists, would further increase the computational complexity and overall efficiency risks to deteriorate even further.

Finally, we report the MSEs of the surfaces that were successfully parameterized in Figure 4.27 (right). Also in this case the neural network results in significantly smaller errors.

Example 4.3.5 (c)

Generalization to higher degrees

In order to test the generalization properties of the neural network in terms of different degrees, we apply it to point clouds sampled from surfaces of higher bi-degree, namely $\boldsymbol{d} = (d, d)$ with $d = 3, 4, 5$, with and without Gaussian noise and we use tensor-product Bézier surfaces of the same degree for fitting the parameterized point clouds. Figure 4.28 shows the MSEs as well as the timings

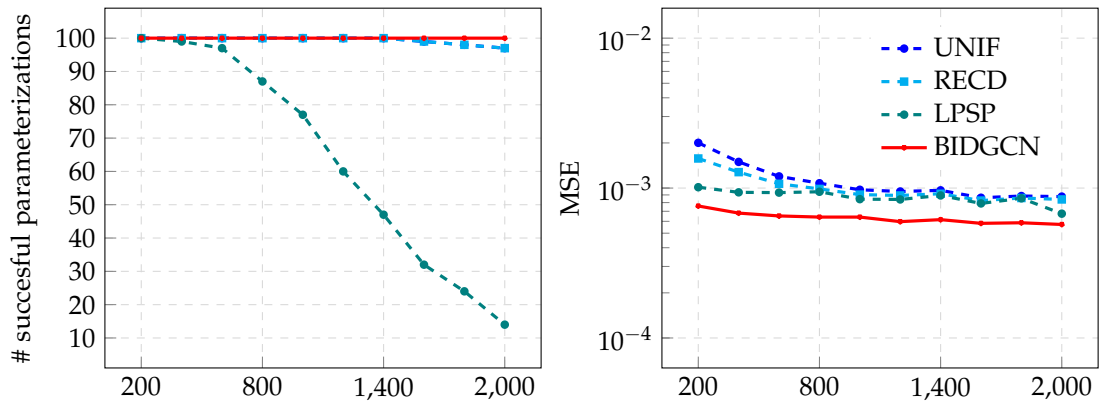


Figure 4.27: Number of successfully parameterized surfaces out of 100 bi-quadratic surfaces of magnitude m with added Gaussian noise of size 5e − 2 analyzed in Example 4.3.5 (b) (left). MSE when applying BIDGCN and the three standard methods to point clouds of different sizes sampled from 100 bi-quadratic surfaces with added Gaussian noise of standard deviation 5e − 2 analyzed in Example 4.3.5 (b) (right); all cases where a method did not succeed were removed from the computation of the MSE.

for point clouds without noise. We observe that BIDGCN results in similar MSEs compared to the LPSP parameterization. However, the computation time for BIDGCN is more than one order of magnitude smaller with respect to LPSP.

Figure 4.29 shows the results on point clouds with added Gaussian noise of standard deviation 1e − 2. We observe that in this case, BIDGCN results in improved MSEs compared to the LPSP parameterization, while the computation time is still more than one order of magnitude smaller.

Example 4.3.5 (d)

Visual comparison of the methods

While the error in the previous examples was averaged over a large number of point clouds, we will now present particular examples that visually demonstrate the performance of the neural network in terms of quality of the reconstructed model. We applied BIDGCN as well as LPSP to the first two point clouds of size 1000 in our test data set, with and without noise, sampled from bi-quadratic surfaces. For LPSP, we compare two choices of the radius: $r = 0.09 \approx 3/\sqrt{m}$ and $r = 0.2$. The parameters, the fitted surface for both methods, the evaluation of the resultant surface at the parameters, and the input data are displayed in Figure 4.30. The point cloud displayed in Figure 4.30 (a) is noise free, whereas

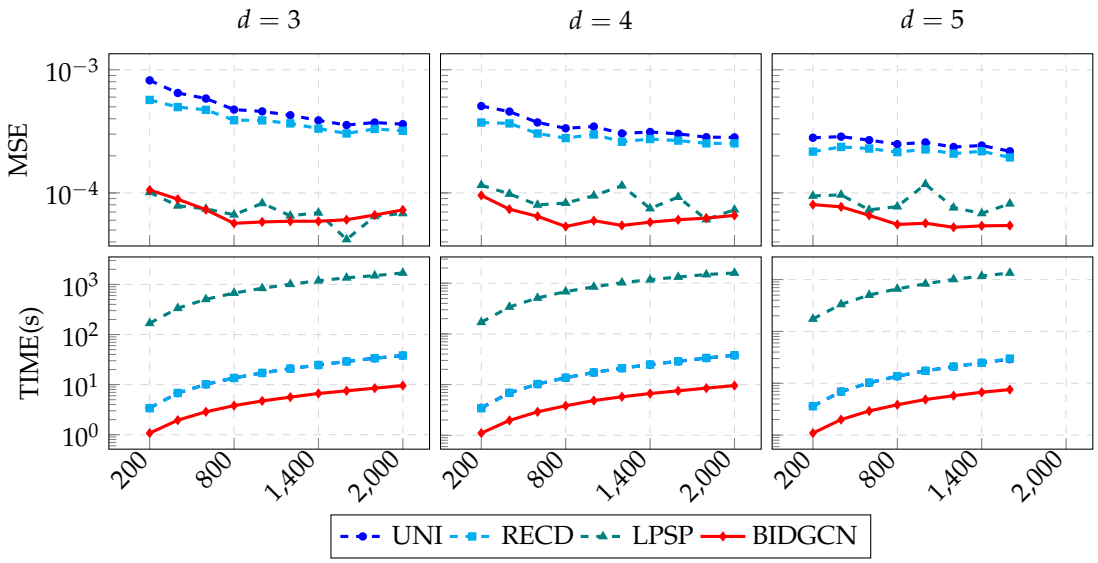


Figure 4.28: MSE (top) and computation time (bottom) when applying the four methods to point clouds of different sizes sampled from 100 surfaces of bi-degree $\boldsymbol{d} = (d, d)$ with $d = 3$ (left), $d = 4$ (middle) and $d = 5$ (right) considered in Example 4.3.5 (c).

we added Gaussian noise with standard deviation $1e-2$ to the point cloud displayed in Figure 4.30 (b). 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 cases. On the other hand, the behaviour of LPSP largely depends on the choice of the radius r. For $r = 0.09$, LPSP parameterization performs well but appears to result in larger deviations from the original parameters compared to BIDGCN. For the choice $r = 0.2$, the parameterization contains large gaps in the interior of the parameter domain. This shows that for using LPSP effectively, one needs to carefully choose the radius r, while BIDGCN is able to predict the optimal graph in the first layers of the network architecture and therefore does not require any fine-tuning.

Example 4.3.5 (e)

Real-world data

In this experiment, we process real world point clouds of different sizes, representing a Nefertiti face model, and we lead a comparison between the standard parameterization methods and the proposed BIDGCN, in terms of accuracy and computational time. As concerns the standard parameterization methods, a suit-

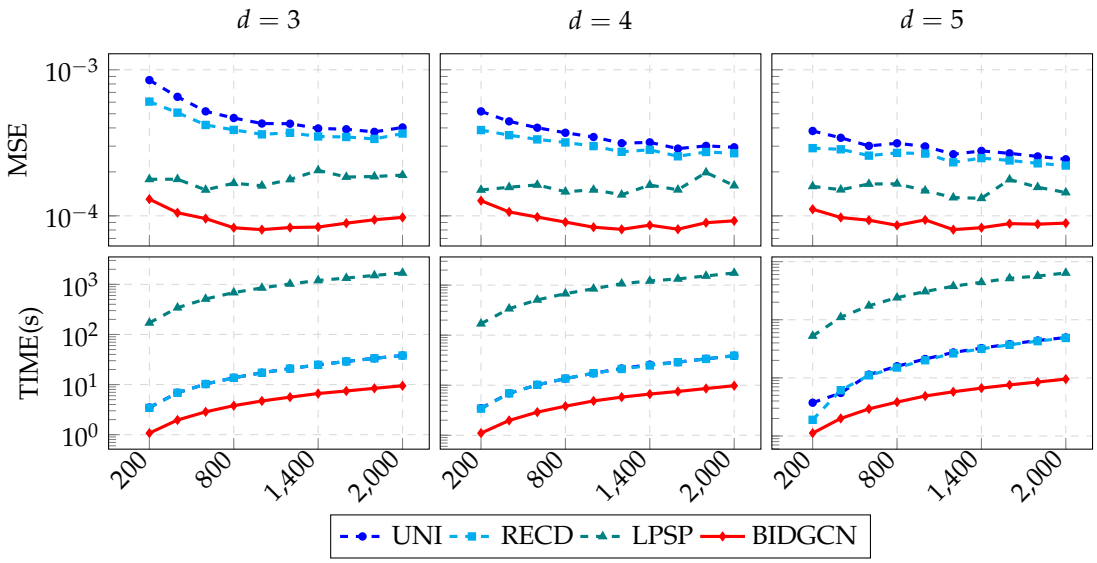
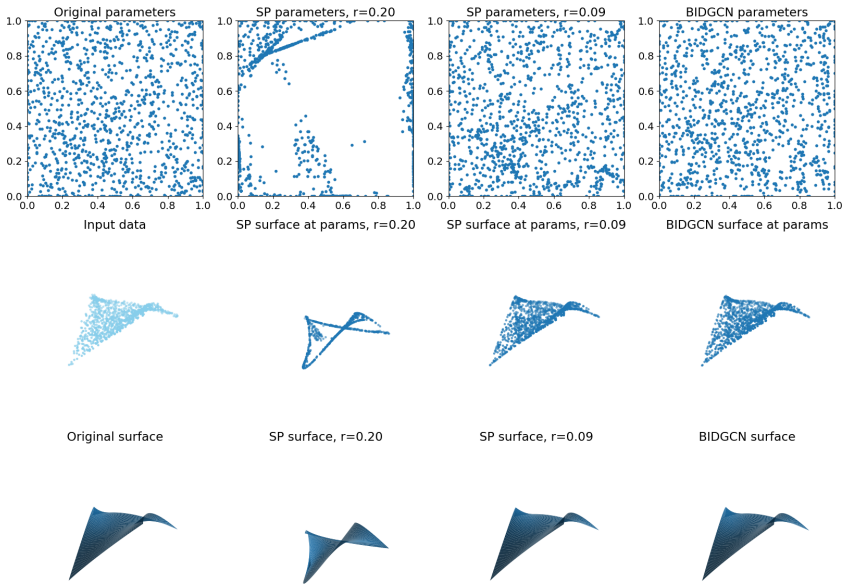
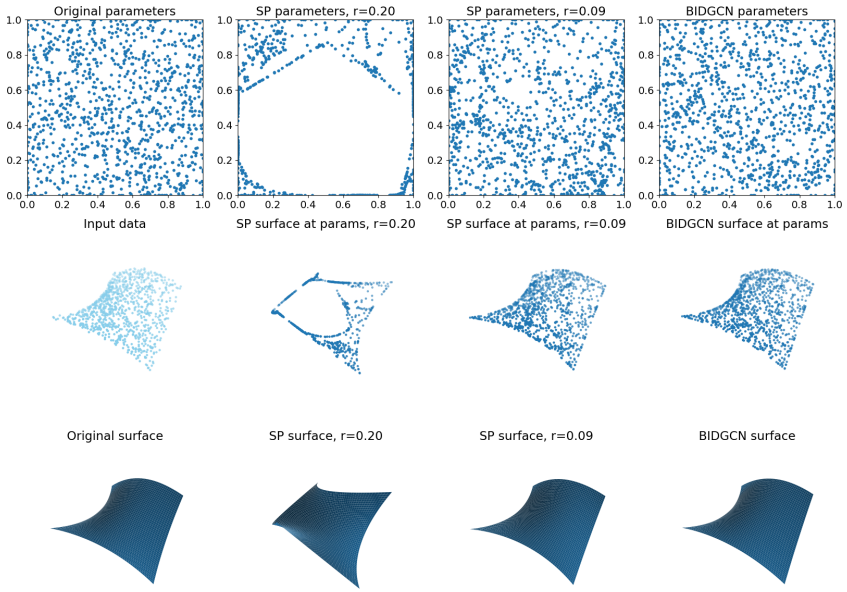


Figure 4.29: MSE (top) and computation time (bottom) when applying the four methods to point clouds of different sizes sampled from 100 of bi-degree $\boldsymbol{d} = (d, d)$, with $d = 3$ (left), $d = 4$ (middle) and $d = 5$ (right), with added Gaussian noise of standard deviation 1e $-$ 2 analyzed in Example 4.3.5 (c).

able choice of the radius r needs to be selected. To produce fair comparisons and avoid unreasonable parameter value distributions, as illustrated in Figure 4.30, we execute a heuristic search for $r = \frac{3}{\sqrt{m}}, 0.05, 0.075, 0.1, \ldots, 0.25, 0.275, 0.3$ on each dataset by computing the polynomial approximation of bi-degree $\boldsymbol{d} = (2, 2)$, and selecting the radius r giving the best MSE. By analyzing the value of the error with respect to r, we decide to fix the radius as defined in (4.16) since it leads to the best approximation results for almost all the real data point clouds. Subsequently, we collect 6 data sets of $m = 532, 1043, 2062, 3253, 6024, 8818$ points, acquired from the same Nefertiti face model. We then compute the parametric values for each Nefertiti point cloud with the standard and the proposed BIDGCN methods to construct a polynomial bi-quadratic least-squares approximation. For each method, we report the MSE associated with the reconstructed polynomial surface and the total computational time needed to parameterize and approximate each point cloud in Figure 4.31. In line with the trends observed on the synthetic data investigated in the previous sections, the uniform and inverse distance parameterizations lead to MSEs that are, in general, much higher than LPSP and BIDGCN. On the other hand, the error values obtained with BIDGCN and LPSP are similar, but our method scales favourably



(a)



(b)

Figure 4.30: Visual comparison described in Example 4.3.5 (d) of the BIDGCN and LPSP for radius $r = 0.2$ and $r = 0.09$ on a point cloud of size 1000 sampled from a bi-quadratic surface without noise (a) and with Gaussian noise of standard deviation $1e-2$ (b).

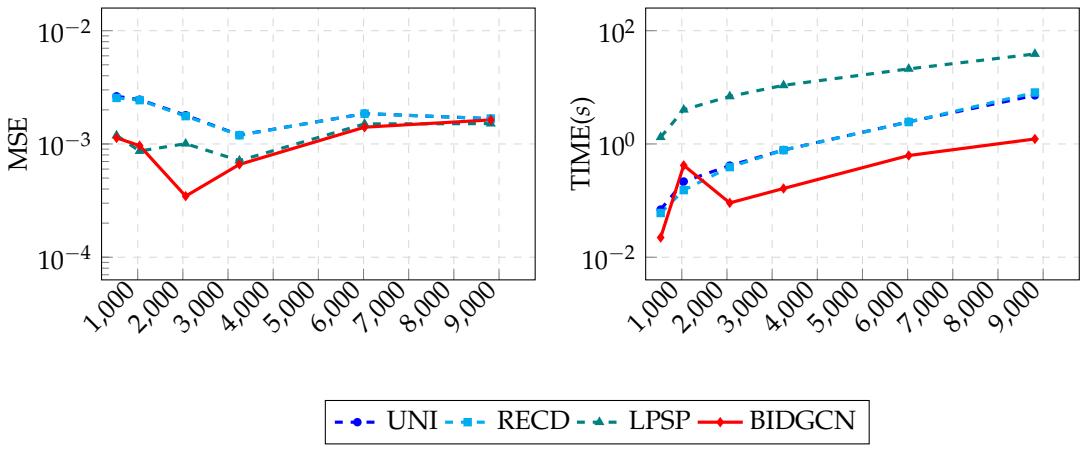


Figure 4.31: Quantitative comparison described in Example 4.3.5 (e) of MSE (left) and computational time (s) (right) of the approximating bi-quadratic surface when parameterizing the point clouds of different size m sampled from the Nefertiti bust model using BIDGCN and the standard methods.

with the dimension of the point cloud always having the lowest computational time.

5

ADAPTIVE THB-SPLINE FITTING WITH MOVING PARAMETERIZATION

In this Chapter, we present adaptive spline model reconstruction schemes with moving parameterization, which consist of a suitable combination of parameter locations and the control points optimization with adaptive refinement routines, using THB-splines. In particular, we propose adaptive alternating and joint optimization methods to optimize the parameter locations and the control points of the (hierarchical) spline geometric model.

We start in Section 5.1 by analyzing existing alternating fitting schemes for tensor-product B-splines. These alternating methods optimize the parameters separately from the control point computation by iteratively inferring the foot-points of the point cloud on the current surface and subsequently updating the control points. The method involved in this second step, e. g., Point Distance Minimization (PDM), Tangent Distance Minimization (TDM) [179], Hybrid Distance Minimization (HDM) [10], or QI, gives the name to the whole fitting procedure.

In Section 5.2, we extend the alternating fitting scheme to the adaptive spline constructions and propose novel adaptive fitting schemes with THB-splines based on different error metrics. We compare the behaviour of different optimization settings for the critical task of distance minimization by also relating the effectiveness of the correction step to the quality of the initial parameterization.

Finally, in Section 5.3 we extend to adaptive spline scenarios the Joint Point Distance Minimization (J-PDM) method, based on the simultaneous joint optimization of control points and parameters, hence avoiding the solution of a linear system of equations and the computation of foot-point projection at every refinement iteration.

We apply the proposed approaches to the reconstruction of real-world data sets, also consisting of aircraft engine components from scanned point data, see Section 5.4. Our study reveals that using a moving parameterization instead of a fixed one, when suitably combined with the adaptive spline loop, can significantly improve the fitting results while also reducing the number of degrees of freedom required to achieve a certain accuracy. More specifically, it can lead to earlier termination of the adaptive process, thus providing more

compact models with less refinement levels and outperforming state-of-the-art hierarchical spline model reconstruction schemes. This Chapter is mostly based on [68] and [69].

5.1 ALTERNATING FITTING SCHEMES: A-PDM, A-TDM, A-HDM, A-QI

In this Section, we consider the surface fitting problem (5.1) addressed at the beginning of Chapter 4, in case of B-spline constructions. Given a set of scattered points as in (1.1), so that $\mathcal{P} = \mathcal{P}_I \dot{\cup} \mathcal{P}_B$, i. e. disjoint union between interior and boundary points, find a spline model $\boldsymbol{s} : \Omega \subset \mathbb{R}^D \to \mathbb{R}^N$,

$$\boldsymbol{s}(\boldsymbol{x}) = \sum_{j=0}^{n} \boldsymbol{c}_j \beta_j(\boldsymbol{x}), \text{ for } \boldsymbol{x} \in \Omega$$

with B-spline basis $\{\beta_0, \dots, \beta_n\}$, coefficients $\boldsymbol{c}_j \in \mathbb{R}^N$ for $j = 0, \dots, n$, collected in the matrix C, and point parametric values $\mathcal{U}$ (1.3), collected in the matrix U, so that it satisfies

$$\underset{U,\, C}{\arg\min} \frac{1}{2} \sum_{i=1}^{m} \left\| \sum_{j=0}^{n} \boldsymbol{c}_j \beta_j(\boldsymbol{u}_i) - \boldsymbol{p}_i \right\|_2^2 + \lambda J(\mathcal{C}), \tag{5.1}$$

where J is a fairing term, which we choose as in (3.16).

Due to the complexity of this non-linear problem, standard methods usually decouple it into two smaller sub-problems, treated separately, i. e. the computation of the parameters $\mathcal{U}$ and the computation of the control points. Moreover, a *fixed* parameterization $\mathcal{U}$ is initially constructed and is often kept fixed throughout the (adaptive) fitting algorithm.

The idea that the parameterization must also be adjusted to the fitted geometry was introduced in [90], in the context of B-spline curve approximation. The so-called intrinsic parametrization, or Parameter Correction (PC) method, is (iteratively) computed starting from a certain initial approximation. At each iteration, the fitted points $\mathcal{P}$ are projected onto the current geometry, and the projected footpoints take the place of the previous parameter values. The control points of the B-spline geometry are updated by re-fitting the geometry with the updated parameter values. Note that each re-fit can then significantly improve the reconstructed geometric model. The core of the method relies on efficient footpoint projection, which is reduced to a non-linear minimization problem. The iterative approach of [90] for finding the footpoint (i. e. the closest point on the geometry) has been revisited in [159], where a Newton-like method

is proposed for the problem. As in all non-linear minimization schemes, the quality of the initial points, the nature of the objective function, as well as the different stepping strategies play a crucial role in the computation.

For a fixed spline space, tackling the problem of foot-points and control points separately, brought to the development of alternating fitting methods, based on the iterative execution of the two following steps:

1. footpoint projection and parameter update;
2. control points computation.

5.1.1 *Foot-point projection and parameter update*

This method consists of locating the points on the geometric model that are the closest to the data points in terms of Euclidean distance. Given a point cloud $\mathcal{P}$, its parameterization $\mathcal{U}$, and a surface $\boldsymbol{s} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^3$, the surface closest point problem consists in solving the following minimization problem,

$$\min_{\boldsymbol{u}_i \in \Omega} \frac{1}{2} \left\| \boldsymbol{s}\left(\boldsymbol{u}_i\right) - \boldsymbol{p}_i \right\|_2^2, \quad \text{for each } i = 1, \ldots, m. \tag{5.2}$$

This two-dimensional nonlinear problem can be explicitly formulated as

$$\left(\boldsymbol{s}\left(\boldsymbol{u}_i\right) - \boldsymbol{p}_i\right)^T \nabla \boldsymbol{s}\left(\boldsymbol{u}_i\right) = 0, \text{ for each } i = 1, \ldots, m, \tag{5.3}$$

where $\nabla \boldsymbol{s}$ indicates the gradient of the surface $\boldsymbol{s}$, and solved by employing a suitable optimizer. In view of (5.3), the vector connecting the data point $\boldsymbol{p}_i$ to the surface point $\boldsymbol{s}(\boldsymbol{u}_i)$ has to be orthogonal to the tangent plane of the surface, and $\boldsymbol{s}\left(\boldsymbol{u}_i\right)$ is then usually called the *footpoint* of $\boldsymbol{p}_i$ over $\boldsymbol{s}$ for each $i = 1, \ldots, m$. The updated parameterization $\bar{\boldsymbol{u}}_i$ is defined as the solution of (5.3), for $i = 1, \ldots, m$. Figure 5.1 illustrates the projection of a point $\boldsymbol{p}_i$ and its footpoint over a surface $\boldsymbol{s}$. The connection between the data point and the surface point is represented by a dotted line, whereas the connection between the data point and its footpoint over the surface is represented by a dashed line. Finally, the updated parameter $\bar{\boldsymbol{u}}_i$ to be associated with $\boldsymbol{p}_i$ is highlighted in red.

Remark 22. *Since we assume the point cloud $\mathcal{P}$ to be a disjoint union between interior and boundary points, i. e. $\mathcal{P} = \mathcal{P}_I \dot{\cup} \mathcal{P}_B$, we decouple the problem in (5.2) accordingly. In particular, if $\boldsymbol{p}_i \in \mathcal{P}_I$, i. e. for interior points, we solve the problem in (5.2). On the other hand, if if $\boldsymbol{p}_i \in \mathcal{P}_B$, i. e. for boundary points, we solve*

$$\min_{\boldsymbol{u}_i \in \partial\Omega} \frac{1}{2} \left\| \boldsymbol{s}\left(\boldsymbol{u}_i\right) - \boldsymbol{p}_i \right\|_2^2, \textit{ for each } i = n+1, \ldots, m,$$

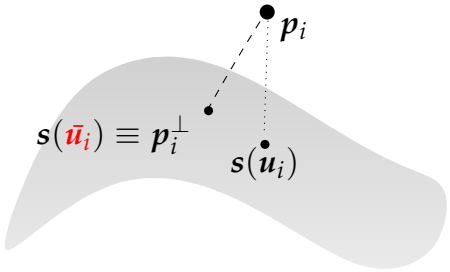


Figure 5.1: Foot-point projection.

where $\partial\Omega$ *indicates the boundary of the parametric domain* $\Omega \subset \mathbb{R}^D$*. Moreover, if the boundary of the domain is composed of more than* 1 *curve, we further decouple the optimization problem by constraining each boundary point to be projected only on the boundary curve it belongs to.*

5.1.2 *Control points computation*

Depending on the strategy employed to compute the control points, different methods have been developed.

A-PDM

Solving at each iteration

$$\min_{C} \frac{1}{2} \sum_{i=1}^{m} \left\| \sum_{j=0}^{n} \boldsymbol{c}_j \beta_j(\boldsymbol{u}_i) - \boldsymbol{p}_i \right\|_2^2 + \lambda J(\mathcal{C}), \tag{5.4}$$

with respect to the coefficients $C \in \mathbb{R}^{(n+1)\times N}$, leads to the formulation of the Alternating Point Distance Minimization (A-PDM) method. This method is wildly used because of its simplicity, see e. g., [65, 89, 90, 144, 146, 159]. As observed in [8], from an optimization view-point, A-PDM exhibits linear convergence.

Remark 23. *Note that the coefficients update of A-PDM, correspond to the weighted LS problem discussed in Section 3.1 (with* $\lambda = 0$*) see* (3.15)*, with constant weights equal to* 1*.*

Consequently, the linear system of normal equations to solve at each step of A-PDM is

$$\left(B^\top B + \lambda G\right) \boldsymbol{c} = B^\top \boldsymbol{p}, \tag{5.5}$$

where B is the collocation matrix as in (3.4), G is the matrix representing the contribution of the functional J, i. e.

$$G_{i,j} = \int_\Omega \left(\frac{\partial^2 \beta_i}{\partial x \partial x} \frac{\partial^2 \beta_j}{\partial x \partial x} + 2 \frac{\partial^2 \beta_i}{\partial x \partial y} \frac{\partial^2 \beta_j}{\partial x \partial y} + \frac{\partial^2 \beta_i}{\partial y \partial y} \frac{\partial^2 \beta_j}{\partial y \partial y} \right) \mathrm{d}x\mathrm{d}y, \tag{5.6}$$

for $i, j = 0, \ldots, n$, $\boldsymbol{p} \in \mathbb{R}^{m \times N}$ is the matrix containing the points of $\mathcal{P}$, and $\boldsymbol{c} \in \mathbb{R}^{(n+1) \times N}$ is the matrix containing the unknown coefficients.

We now introduce a different arrangement of the linear system of equations in (5.5), which will be useful for the following discussions. Let $\widetilde{B} \in \mathbb{R}^{Nm \times N(n+1)}$ be the matrix, where the main-diagonal blocks B consists of the collocation matrix (3.4),

$$\widetilde{B} = \begin{pmatrix} B & 0 & \ldots & 0 \\ 0 & B & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & B \end{pmatrix}. \tag{5.7}$$

Similarly, let $\widetilde{G} \in \mathbb{R}^{N(n+1) \times N(n+1)}$ be of the form

$$\widetilde{G} = \begin{pmatrix} G & 0 & \ldots & 0 \\ 0 & G & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & G \end{pmatrix}, \tag{5.8}$$

where the main-diagonal blocks correspond to the matrix G as in (5.6), representing the functional J. Finally, let $\widetilde{\boldsymbol{p}} \in \mathbb{R}^{mN}$ be the vector containing the data points ordered as follows,

$$\widetilde{\boldsymbol{p}} = \left(p_1^{(1)}, \ldots, p_m^{(1)}, \ldots, p_1^{(N)}, \ldots, p_m^{(N)} \right)^\top \in \mathbb{R}^{mN}. \tag{5.9}$$

Hence, the problem in (5.5) can be rewritten as

$$\left(\widetilde{A}_{\mathrm{pdm}} + \lambda \widetilde{G} \right) \boldsymbol{c} = \boldsymbol{b}_{\mathrm{pdm}}, \tag{5.10}$$

where $\widetilde{A}_{\mathrm{pdm}} \in \mathbb{R}^{N(n+1) \times N(n+1)}$ and $\boldsymbol{b}_{\mathrm{pdm}} \in \mathbb{R}^{N(n+1)}$, so that

$$\widetilde{A}_{\mathrm{pdm}} = \widetilde{B}^\top \widetilde{B}, \quad \boldsymbol{b}_{\mathrm{pdm}} = \widetilde{B}^\top \widetilde{\boldsymbol{p}},$$

and $\boldsymbol{c} \in \mathbb{R}^{N(n+1)}$ is the vector of unknowns, i. e.

$$\boldsymbol{c} = \left(c_0^{(1)}, \ldots, c_n^{(1)}, \ldots, c_0^{(N)}, \ldots, c_n^{(N)} \right)^\top. \tag{5.11}$$

A-TDM

The A-TDM method [9, 123, 179] is characterized by the following update rule for the control points,

$$\min_{\mathbf{C}} \frac{1}{2} \sum_{i=1}^{m} \left[\left(\sum_{j=0}^{n} \boldsymbol{c}_j \beta_j(\boldsymbol{u}_i) - \boldsymbol{p}_i \right)^T \cdot \boldsymbol{n}_i \right]^2 + \lambda \boldsymbol{J}(\mathcal{C}), \tag{5.12}$$

where

$$\boldsymbol{n}_i = \left(n_i^{(1)}, \ldots, n_i^{(N)} \right)^\top \in \mathbb{R}^N$$

is the unit normal vector at the surface point $\boldsymbol{s}(\boldsymbol{u}_i)$, for each $i = 1, \ldots, m$. For $N = 2, 3$, and for each $i = 1, \ldots, m$ the term

$$\left[\left(\sum_{j=0}^{n} \boldsymbol{c}_j \beta_j(\boldsymbol{u}_i) - \boldsymbol{p}_i \right)^T \cdot \boldsymbol{n}_i \right]^2 = \left[\left(\boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i \right) \cdot \boldsymbol{n}_i \right]^2,$$

defines the distance between the data point $\boldsymbol{p}_i$ and the tangent line (if $D = 1$) or plane (if $D = 2$) at $\boldsymbol{s}(\boldsymbol{u}_i)$.

In order to recover the linear system of equation to be solved at each step of the A-TDM method, for each $k = 1, \ldots, N$, let $N_k \in \mathbb{R}^{m \times m}$ be the diagonal matrix containing the normal components at each point along the k-th physical direction, i. e.

$$N_k = \begin{pmatrix} n_1^{(k)} & 0 & \ldots & 0 \\ 0 & n_2^{(k)} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & n_m^{(k)} \end{pmatrix},$$

and let $\widetilde{N} \in \mathbb{R}^{Nm \times m}$ the block matrix, so that its blocks correspond to N_k, i. e.

$$\widetilde{N} = \begin{pmatrix} N_1 \\ N_2 \\ \vdots \\ N_N \end{pmatrix} \in \mathbb{R}^{Nm \times m}.$$

The problem in (5.12) can be rewritten in matrix form as

$$\min_{\boldsymbol{c}} \frac{1}{2} \left[\left(\widetilde{B}\boldsymbol{c} - \widetilde{\boldsymbol{p}} \right)^{\top} \widetilde{N} \right] \left[\left(\widetilde{B}\boldsymbol{c} - \widetilde{\boldsymbol{p}} \right)^{\top} \widetilde{N} \right]^{\top} + \lambda \widetilde{G},$$

with $\widetilde{B}$, $\widetilde{G}$, $\widetilde{\boldsymbol{p}}$ and $\boldsymbol{c}$ as defined in (5.7), (5.8), (5.9) and (5.11), respectively. Therefore, the normal equations with respect to the unknowns $\boldsymbol{c}$ are

$$\left(\widetilde{A}_{\text{tdm}} + \lambda \widetilde{G} \right) \boldsymbol{c} = \boldsymbol{b}_{\text{tdm}}, \tag{5.13}$$

where $\widetilde{A}_{\text{tdm}} \in \mathbb{R}^{N(n+1) \times N(n+1)}$ and $\boldsymbol{b}_{\text{tdm}} \in \mathbb{R}^{N(n+1)}$ so that

$$\widetilde{A}_{\text{tdm}} = \widetilde{B}^{\top} \widetilde{N} \widetilde{N}^{\top} \widetilde{B}, \quad \boldsymbol{b}_{\text{tdm}} = \widetilde{B}^{\top} \widetilde{N} \widetilde{N}^{\top} \widetilde{\boldsymbol{p}}.$$

It is well known that A-TDM does not show a stable performance near high curvature regions. Moreover, it has been proven to be a Gauss-Newton minimization without a step-size control, see [179], and a regularization term needs to be added in order to improve the stability of the method. In particular, it is common to apply the Levenberg–Marquardt regularization, see [179, 189], by modifying (5.13) as

$$\left(\widetilde{A}_{\text{tdm}} + \gamma I + \lambda \widetilde{G} \right) \boldsymbol{c} = \boldsymbol{b}_{\text{tdm}}, \tag{5.14}$$

to add the term γI, where $I \in \mathbb{R}^{N(n+1) \times N(n+1)}$ is the identity matrix and $\gamma \in \mathbb{R}, \gamma \geq 0$. This A-TDM variant is usually addressed in the literature as Alternating Tangent Distance Minimization Levenberg-Marquardt (A-TDMLM).

A-HDM

By combining the A-PDM and A-TDM methods, the Alternating Hybrid Distance Minimization (A-HDM) method can be defined, see, e. g., [10, 130]. In this case, the distance metric at a sample $\boldsymbol{s}(\boldsymbol{u}_i)$ for each $i = 1, \ldots, m$ takes into account both PDM and TDM metrics, i. e.

$$\gamma_i \left\| \boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i \right\|_2^2 + \delta_i \left[\left(\boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i \right)^{\top} \cdot \boldsymbol{n}_i \right]^2, \tag{5.15}$$

with $\gamma_i, \delta_i \geq 0$ and $\gamma_i + \delta_i = 1$. The objective function for the control point update to minimize is then

$$\min_{C} \frac{1}{2} \sum_{i=1}^{m} \left\{ \gamma_i \left\| \sum_{j=0}^{n} \boldsymbol{c}_j \beta_j(\boldsymbol{u}_i) - \boldsymbol{p}_i \right\|_2^2 + \delta_i \left[\left(\sum_{j=0}^{n} \boldsymbol{c}_j \beta_j(\boldsymbol{u}_i) - \boldsymbol{p}_i \right)^{T} \cdot \boldsymbol{n}_i \right]^2 \right\} + \lambda \boldsymbol{J}(\mathcal{C}).$$

Remark 24. *If $\gamma_i = 1$ and $\delta_i = 0$ for each $i = 1, \dots, m$, we recover the A-PDM method. Similarly, if $\gamma_i = 0$ and $\delta_i = 1$, then we fall back to the A-TDM method.*

The authors in [130] suggest the following choice of the blending weights,

$$\gamma_i = \frac{\mathrm{d}_i}{2 \max_{j=1,\dots,m} \mathrm{d}_j} \text{ and } \delta_i = (1 - \gamma_i), \text{ for } i = 1, \dots, m,$$

for which (5.15) results to be a weighted combination of the A-PDM and A-TDM error terms, depending on the point-wise distance

$$\mathrm{d}_i = \| \boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i \|_2 .$$

In order to exploit the curvature information from the point cloud $\mathcal{P}$, the authors in [10], propose as blending weights

$$\gamma_i = \frac{\mathrm{d}_i}{\mathrm{d}_i + \rho_i} \quad \text{and} \quad \delta_i = (1 - \gamma_i) = \frac{\rho_i}{\mathrm{d}_i + \rho_i},$$

with,

$$\rho_i = \frac{1}{\max\{|c_{i,1}|, |c_{i,2}|\}},$$

where $c_{i,1}, c_{i,2}$ are the principal curvature values at every data points $\boldsymbol{p}_i \in \mathcal{P}$, for $i = 1, \dots, m$. Hence, (5.15) becomes a weighted combination of the A-PDM and A-TDM error terms, with weights depending also on the surface discrete curvature. In the numerical investigation presented in this Thesis, we also consider a more simple choice of the blending weights as constant non-negative values, i. e.

$$\gamma_i = \gamma \geq 0, \quad \text{and} \quad \delta_i = \delta = (1 - \gamma), \text{ for } i = 1, \dots, m,$$

with $\gamma + \delta = 1$.

Independently from the specific choice of the blending weights, the linear system of equations to be solved at each iteration of the A-HDM method has the following form,

$$\left(\widetilde{A}_{\mathrm{pdm}} + \widetilde{A}_{\mathrm{tdm}} + \lambda \widehat{G} \right) \boldsymbol{c} = \left(\boldsymbol{b}_{\mathrm{pdm}} + \boldsymbol{b}_{\mathrm{tdm}} \right), \tag{5.16}$$

where the matrices and right-hand-sides defined in (5.5) and (5.13) include the blending weight matrices

$$\begin{aligned}
&\Gamma = \operatorname{diag}\{\gamma_1, \dots, \gamma_m\} \in \mathbb{R}^{m\times m},\\
&\widetilde{\Gamma} = \begin{pmatrix} \Gamma & 0 & \dots & 0 \\ 0 & \Gamma & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \Gamma \end{pmatrix} \in \mathbb{R}^{Nm\times Nm},\\
&D = \operatorname{diag}\{\delta_1, \dots, \delta_m\} \in \mathbb{R}^{m\times m},
\end{aligned} \tag{5.17}$$

so that

$$\widetilde{A}_{\text{pdm}} = \widetilde{B}^\top \widetilde{\Gamma} \widetilde{B}, \text{ and } \widetilde{A}_{\text{tdm}} = \widetilde{B}^\top \widetilde{N} D \widetilde{N}^\top \widetilde{B},$$

and

$$\boldsymbol{b}_{\text{pdm}} = \widetilde{B}^\top \widetilde{\Gamma} \widetilde{\boldsymbol{p}}, \text{ and } \boldsymbol{b}_{\text{tdm}} = \widetilde{B}^\top \widetilde{N} D \widetilde{N}^\top \widetilde{\boldsymbol{p}}.$$

Remark 25. *When dealing with open surfaces, to reconstruct point clouds $\mathcal{P} = \mathcal{P}_I \dot\cup \mathcal{P}_B$, instability might arise from the normals computed at the boundary points $\mathcal{P}_B$. To reduce this unstable behaviour, we modify the A-TDM method by decoupling the computation of the unknowns related to the interior of the surface from the computation of the unknowns for the boundary curves. In particular, $\boldsymbol{c} = \boldsymbol{c}_{\text{int}} \dot\cup \boldsymbol{c}_{\text{bdr}}$, where $\boldsymbol{c}_{\text{int}}$ is the solution of (5.16) built on the interior parameters $\mathcal{U}_I$ and points $\mathcal{P}_I$, and similarly, $\boldsymbol{c}_{\text{bdr}}$ is the solution of (5.5) built on the boundary parameters $\mathcal{U}_B$ and points $\mathcal{P}_B$.*

Remark 26. *A-HDM can be considered a regularized version of A-TDM. While for A-TDMLM the regularization term is a diagonal matrix γI as in (5.14), in this case the regularization term is played by the PDM matrix $\tilde{A}_{\text{pdm}}$ as in (5.16). Moreover, from all the numerical results of [10], we can infer that A-HDM with curvature-based weights and A-TDMLM can achieve the same accuracy. We also experienced that, independently of the choice of the weights, the A-HDM system matrix results more numerically stable than the A-TDMLM, hence better suited for numerical computations, in particular for complex, e. g., adaptive, constructions. For this reason, we prefer A-HDM over A-TDMLM to conduct the analysis developed in the following sections. Note, however, that A-PDM is nevertheless the method whose system matrix has lower condition numbers, being numerically more stable also of A-HDM.*

For completeness, we recall that the A-HDM method is a variant of the Alternating Squared Distance Minimization (A-SDM) method, based on the

local quadratic approximants of the squared distance function of a surface to a point. A-SDM has been developed in [147, 149] for active contour models to fit a surface to a target shape, applied to subdivision fitting schemes in [29], and subsequently adjusted in [179] for the problem of B-spline curve fitting to points clouds and in [122] for the problem of surface fitting and registration. Further discussion on the local quadratic approximants for the squared distance function can be found in [148].

A-QI

All the presented methods are characterized by the solution of a minimization problem for the computation of the control points of the spline geometry. However, there are no constraints on the method to be used for their definition. In particular, we consider an additional alternating method, i. e. the Alternating Quasi-Interpolation (A-QI), where the second step of the alternating scheme, see Section 5.1, consists in the solution of a QI problem, as the one described in Section 3.2.1.

5.1.3 *Comparison of different weight choice for A-HDM*

In this Section we compare the behaviour of the A-PDM and A-HDM methods for different choices of blending weights. More precisely, for A-HDM, we consider the blending weights in [130], [10], together with a suitable constant choice. We consider a gridded point cloud, shown in the middle of the bottom line of Figure 4.7, consisting of 3025 data sampled from a ship hull geometry. Subsequently, we parameterize the data with CEN and CHL parameterization methods [145]. We then compute a first initial approximation by performing a penalized LS fit, i. e. PDM, with tensor-product B-splines of bi-degree $\boldsymbol{d} = (2, 2)$, a 5×2 tensor-product mesh, and penalization weight $\lambda = 1\mathrm{e} - 7$. We then consider this initial approximation as starting point for the A-PDM and A-HDM schemes.

As concerns the A-HDM method with constant weights, for the shiphull point cloud parameterized with CEN, we perform a fine-tuning search on the values of γ, with $\delta = (1 - \gamma)$. More precisely, we execute the A-HDM method for $\gamma = 1\mathrm{e} - 3, 1\mathrm{e} - 2, 1\mathrm{e} - 1, 2.5\mathrm{e} - 1, 5\mathrm{e} - 1, 7.5\mathrm{e} - 1$ and show the results in Figure 5.2 in terms of MSE evolution over the number of alternating iterations (10 on the left and 30 on the right). From this fine-tuning, we choose to set $\gamma = 1\mathrm{e} - 2$ (the violet line in the plots) as constant choice in view of the higher approximation accuracy achieved in this case.

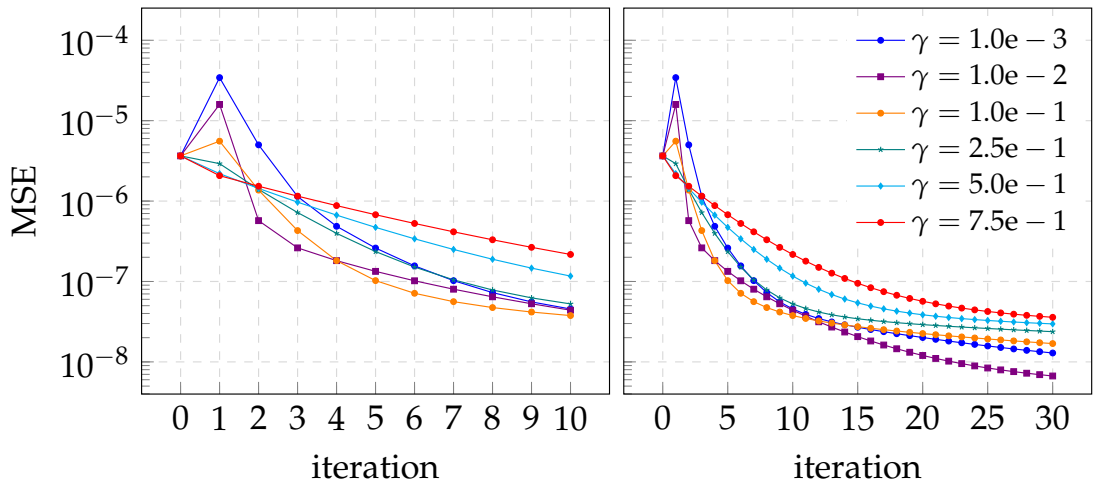


Figure 5.2: Shiphull point cloud with 3025 gridded points initially parametrized with standard CEN method. Alternating spline approximation for bi-degree $\boldsymbol{d} = (2,2)$, on a 5×2 tensor-product mesh. Comparison of MSE for different choice of constant weights in the A-HDM method for 10 (left) and 30 (right) iterations.

Figure 5.3 shows the comparison of different weight choices for the A-HDM with A-PDM, performed with 10 (left) and 30 (right) iterations, for the approximation of the point cloud with initial CEN parameterization. We can observe that the A-HDM method, for any choice of blending weights, improves the approximation accuracy in terms of MSE with respect to the A-PDM method by about 1 order of magnitude. Note also that for the first 10 iterations, the curvature-based blending weights (teal line in the plots) show the worst approximation behaviour, among the other weight choices. Nevertheless, if we let the algorithm perform a sufficient number of iterations, e. g., 30, then they show better behaviour with respect to the error-driven weights of [130] (olive line in the plots) and achieve the same accuracy as the constant fine-tuned weights (violet line in the plots).

Analogous results can be observed in Figure 5.4 for the approximation of the ship hull point cloud initially parameterized with CHL. In particular, the curvature error-based methods show better performance if we let the alternating fitting algorithm run for a sufficient number of iterations.

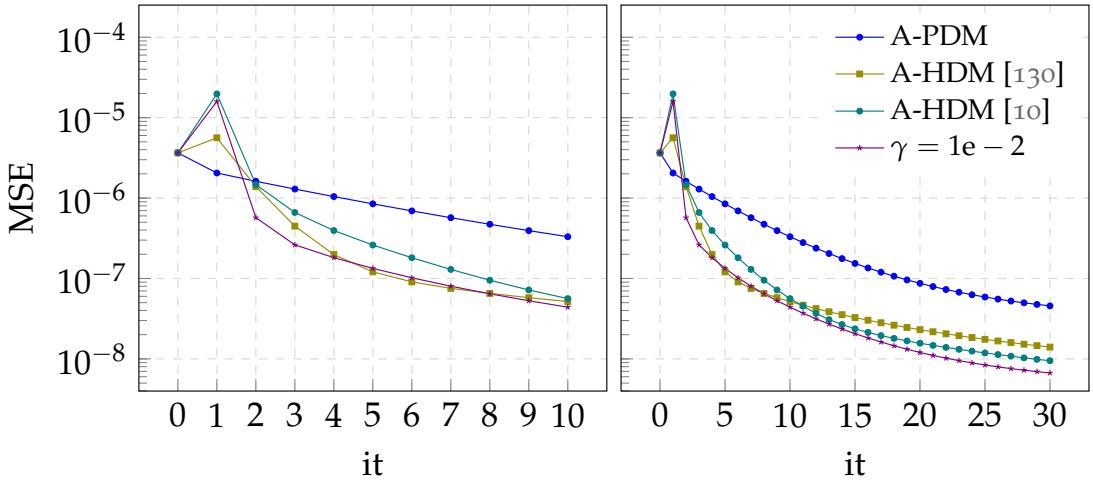


Figure 5.3: Alternating B-spline fitting of the shiphull point cloud with 3025 gridded data points for bi-degree $\boldsymbol{d} = (2, 2)$, on a 5×2 tensor-product mesh, with initial CEN parameterization. The comparison of MSE between A-PDM, A-HDM with constant choice of the weights, A-HDM with the weights of [130] and A-HDM with the weights of [10] is shown for 10 (left) and 30 (right) iterations.

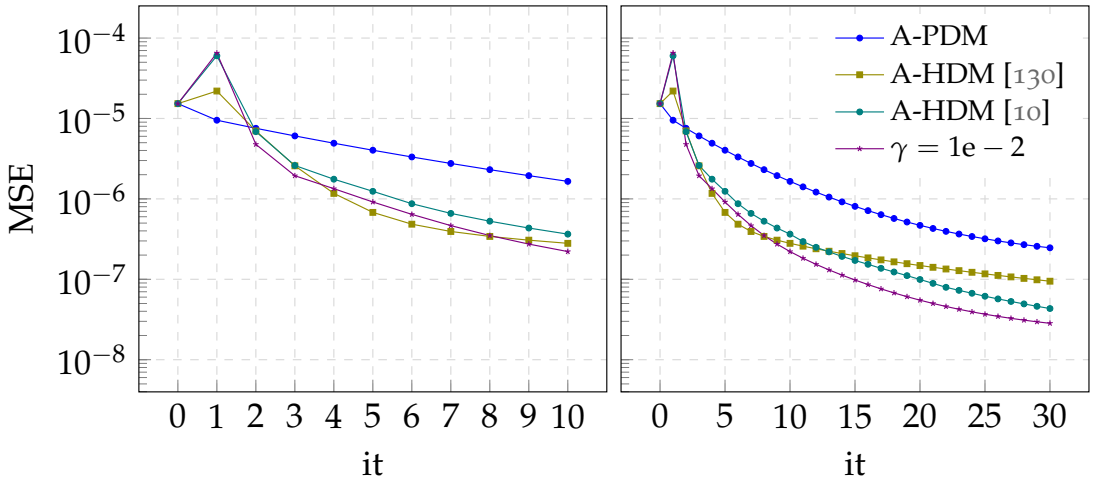


Figure 5.4: Alternating B-spline fitting of the shiphull point cloud with 3025 gridded data points for bi-degree $\boldsymbol{d} = (2, 2)$, on a 5×2 tensor-product mesh, with initial CHL parameterization. The comparison of MSE between A-PDM, A-HDM with constant choice of the weights, A-HDM with the weights of [130] and A-HDM with the weights of [10] is shown for 10 (left) and 30 (right) iterations.

5.2 ADAPTIVE ALTERNATING FITTING SCHEMES

In this Section, we extend the presented methods to adaptive THB-spline fitting frameworks, motivated by the idea that, as the refinement proceeds in an adaptive setting, not only the geometric hierarchical model but also the parameter values of each data point should be optimized. In fact, when adaptively fitting a parametric geometric model, there is no evidence that the initial (possible) optimality of the parameterization is maintained when the approximation space is iteratively refined. Note, however, that the optimality of the initial parameterization is fundamental for the success of the proposed method, as discussed in Example 5.2.2 (d).

In the adaptive THB-spline fitting scenario, once an initial parameterization and mesh configuration are chosen, the adaptive fitting technique consists of four fundamental steps described in (1.4) that are successively repeated: 1. SOLVE: fitting the THB-spline approximation on the current (hierarchical) mesh; 2. ESTIMATE: error estimation; 3. MARK: mesh marking strategy; 4. REFINE: mesh refinement strategy to suitably identify the new hierarchical mesh to be used in the next iteration, which starts again from 1.

To address the parameter update within the adaptive fitting loop, we propose exploiting the alternating fitting methods covered in Section 5.1 in the first step of the adaptive scheme, i. e. SOLVE. This choice enables the definition of adaptive THB-spline fitting with moving parameters.

At the beginning of any adaptive iteration, the SOLVE routine is performed as follows. Compute the control points according to the chosen method, e. g., PDM, TDM, HDM, or QI; perform a certain number of PC steps, i. e.

1. solve (5.3) for $\boldsymbol{s}$ belonging to the current hierarchical space, hence projecting the points of $\mathcal{P}$ on the THB-spline surface $\boldsymbol{s}$, and considering their foot-point as the new parameters $\mathcal{U}$;

2. update the geometric THB-spline model by fitting again the surface $\boldsymbol{s}$ to the points $\mathcal{P}$ with the corrected parameters.

After the conclusion of the PC steps, the new parameters and updated THB-spline geometry are considered for the next iteration of the adaptive loop. Note that the a certain number of PC-like is applied globally to the entire approximant. In particular, in the A-QI scheme, for each local problem the parameters $\mathcal{U}$ are kept fixed.

As for standard adaptive fitting strategies, the described loop is performed until the maximum point-wise error is within an input tolerance ϵ or a maximum

number of hierarchical levels L is reached. Note that PC steps should naturally be embedded in any adaptive scheme, i. e. not limited to fitting problems, since even if the starting parameterization benefits from optimality within the initial space, there is no evidence that optimality is maintained when moving to wider successively refined approximation spaces.

The pseudocode for the adaptive A-HDM fitting scheme is reported in Algorithm 7. The algorithm for the adaptive A-PDM and A-TDM methods can be recovered by choosing δ_i and γ_i for $i = 1, \dots, m$ on line 3 as in Remark 24. Finally, the algorithm for the adaptive A-QI can be inferred by suitably adding the PC routine (lines 4–6 of Algorithm 7) after line 3 of Algorithm 4.

5.2.1 *Interaction of foot-point projection with adaptive spline configurations*

Finding successfully the foot-point projection of a point on a surface is a challenging open problem; see, e. g., [102] and references therein. As far as parameter correction is concerned, the foot-point projection in [90] is performed iteratively by linearizing the problem in (5.3). Subsequently, in [159], the iterative procedure has been replaced by the application of a Newton-like approach. Due to the locality of Newton-like gradient methods, to properly correct the parameters, it is of fundamental importance to start with a good initial parameterization guess as well as with a reasonably good geometric model.

Figure 5.5 shows the consequences of incorporating the PC routine at a too-early stage of an adaptive fitting scheme. More specifically, we consider the point cloud of Section 3.3.1, consisting of 9281 scattered data acquired from an industrial *tensile* part, which is approximately 2.5e − 2 m long. We fit this data with the adaptive LS scheme developed in [101], equipped with the PC routine. Note that combining the PC with the adaptive LS fitting scheme results in the adaptive extension of the A-PDM method.

An initial tensor-product B-spline approximation of bi-degree $\boldsymbol{d} = (2, 2)$ is built on a *coarse* 4×4 tensor-product mesh and illustrated in Figure 5.5 (a). This initial coarse mesh is then adaptively refined using a criterion of error threshold with $\epsilon = 5\text{e} - 5$ m, leading to a final THB-spline model with 1136 DOFs that registers a MAX error of $8.33\text{e} - 5$ m. The THB-spline approximation obtained by introducing 1 PC step at each iteration of the adaptive loop is shown in Figure 5.5 (b) and is characterized by 969 DOFs, MAX error of $3.059\text{e} - 4$ m. Consequently, we may note that using the inaccurate approximation shown in Figure 5.5 (a) to compute the foot-point projections and correct the parameters leads to a final fitting result that is not suitable for further processing

Algorithm 7: Adaptive alternating fitting scheme.

Input: Point cloud and initial parameterization $\{\boldsymbol{u}_i, \boldsymbol{p}_i\}_{i=1}^m$, 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 PC steps $c_{\max}$.

Compute the initial tensor-product B-spline approximation $\boldsymbol{s} \in V^0$ and the point-wise errors $e_i = \|\boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i\|_2$, for each $i = 1, \dots, m$ and set loop $= 0$

while $\max_i e_i > \epsilon$ *and* loop $<$ L **do**

  Solve the least squares problem

$$\boldsymbol{s} = \arg\min_{v \in V} \frac{1}{2} \sum_{i=1}^{m} \delta_i \left\| \boldsymbol{s}\left(\boldsymbol{u}_i\right) - \boldsymbol{p}_i \right\|_2^2 + \gamma_i \left[\left(\boldsymbol{s}\left(\boldsymbol{u}_i\right) - \boldsymbol{p}_i\right)^\top \cdot \boldsymbol{n}_i \right].$$

  for step $= 1, \dots, c_{\max}$ **do**

    Compute the foot-point projections

$$\min_{\boldsymbol{u}_i \in \Omega} \frac{1}{2} \left\| \boldsymbol{s}\left(\boldsymbol{u}_i\right) - \boldsymbol{p}_i \right\|_2^2, \text{ for each } i = 1, \dots, m$$

    and update the parameters and control points of the spline geometry.

  end

  Compute the errors $e_i = \|\boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i\|_2$, for $i = 1, \dots, m$.

  Mark the domain elements where $e_i > \epsilon$.

  Refine the marked cells and the two surrounding rings of cells.

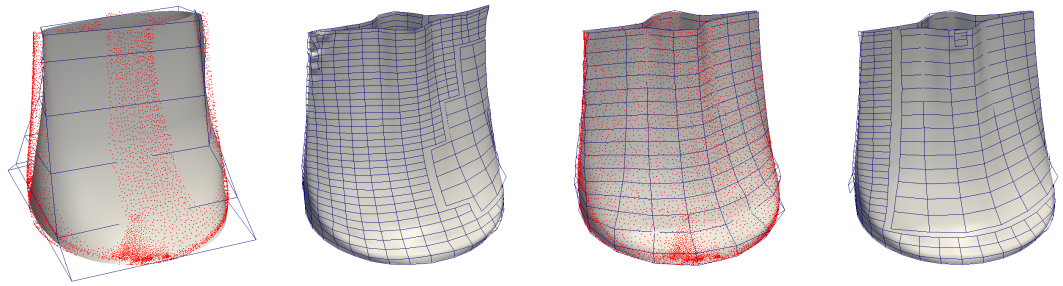
  Update the hierarchical mesh $\mathcal{M}$ and hierarchical space V.

  Set loop $=$ loop $+ 1$.

end

Output: $\boldsymbol{s} \in V$, the THB-spline approximant.

due to the self-intersections visible in Figure 5.5 (b). We then consider as an initial spline configuration the tensor-product B-spline approximation built on a tensor-product mesh with two additional dyadic refinements with respect to Figure 5.5 (a), i. e. with a 16×16 mesh, as shown in Figure 5.5 (c). These settings lead to stable foot-point projection due to the quality of the initial



(a) coarse initial B-spline model (b) THB-spline model (969 DOFs) (c) refined initial B-spline model (d) THB-spline model (607 DOFs)

Figure 5.5: Adaptive A-PDM with THB-splines: effect of the initial tensor-product B-spline approximation. The THB-spline model (b) constructed from the initial configuration (a) defined on a 4×4 tensor-product mesh is affected by self-intersections due to the too coarse initial mesh. The THB-spline model (d) constructed from the initial configuration (c) defined on a 16×16 tensor-product mesh is successfully computed.

approximation, and the LS scheme with moving parameters results in a model with 607 DOFs and MAX error $8.467\text{e}-5$ m, shown in Figure 5.5 (d).

The choice of the optimizer and its settings are fundamental for the successful computation of the foot-point projections. Here, we consider the LS adaptive fitting method with moving parameters and compare the results with respect to the use of different optimizers and optimization settings. In particular, we start with the configuration of Figure 5.5 (c), i. e. bi-degree $\boldsymbol{d} = (2, 2)$ and a 16×16 tensor-product mesh, and apply 1 PC step at each adaptive iteration. For better comparison, we also fix the total number of adaptive refinement steps. The results are shown in Table 5.1. As a reference, we consider the commercial library Parasolid [164]. We can see that taking a too-big minimum step s in the Gradient Descent (GD) method leads to a final result that can be *worse* than not moving the parameters at all, with respect to the number of DOFs of the final models and its accuracy in terms of MAX and MSE. However, if carefully fine-tuned, the results obtained by Parasolid in terms of DOFs, MAX, and MSE can be reached for $s = 1\text{e}-12$. In the following numerical examples, we employed the Hybrid Limited memory Broyden-Fletcher-Goldfarb-Shanno (HLBFGS) method [124] with minimum step length $s = 1\text{e}-9$, since it has always shown to provide extremely similar results to Parasolid in all the considered examples. The results obtained employing HLBFGS for the above-mentioned configuration are also shown in Table 5.1.

method	pts $< \epsilon$	MAX (m)	MSE (m^2)	DOFs
no PC	92.77%	$1.97e-4$	$7.30e-10$	751
GD, $s = 1e-9$	91.91%	$2.92e-4$	$1.06473e-9$	748
GD, $s = 1e-10$	96.88%	$1.62e-4$	$4.43e-10$	690
GD, $s = 1e-11$	99.29%	$8.47e-5$	$2.41e-10$	607
GD, $s = 1e-12$	99.31%	$8.47e-5$	$2.37e-10$	607
HLBFGS	99.31%	$8.47e-5$	$2.37e-10$	607
Parasolid	99.31%	$8.47e-5$	$2.37e-10$	607

Table 5.1: Influence of optimizers used for the foot-point computation loop starting from configuration of Figure 5.5 (c).

5.2.2 *Numerical examples for adaptive alternating methods*

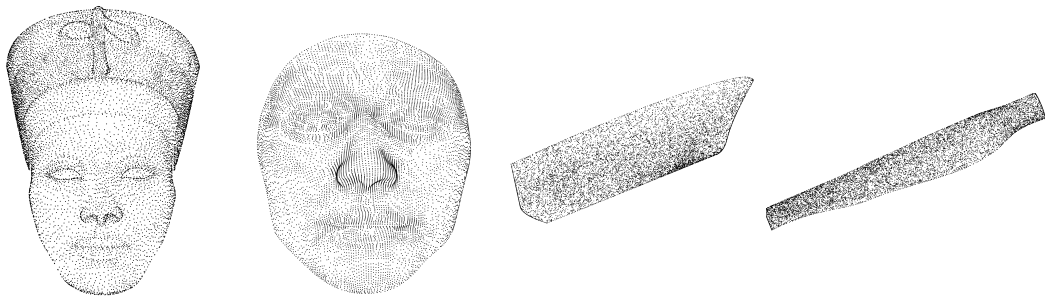
In this Section, we present a selection of numerical examples to assess the benefits of the moving parameterization approach in combination with the adaptive PDM fitting scheme. In particular, we numerically prove that including the PC routine on a proper initial guess and choosing suitable settings for the optimizer leads to outperforming the state-of-the-art THB-spline fitting schemes, see Example 5.2.2 (a). In addition, we combine the presented adaptive A-PDM fitting methods with the parameterization methods presented in Chapter 4 to further improve the accuracy of the final reconstructed geometry, see Example 5.2.2 (b) and Example 5.2.2 (c). Besides the strength of moving the parameters accordingly to the designed space, we show that finding an optimal parameterization for the initial settings plays a fundamental role, see Example 5.2.2 (d). In Example 5.2.2 (e), we combine the moving parameterization technique with the parameterization method in Section 4.3 and the (reweighted) fitting scheme proposed in Section 3.1.8. Finally, in Example 5.2.2 (f) we conduct a comparison between the adaptive A-PDM and A-HDM.

Example 5.2.2 (a)

Adaptive A-PDM

In the following example, we analyze the effects of enriching the adaptive LS fitting scheme with the moving parameterization method. In particular, we approximate a point cloud of 9636 scattered data in $\mathbb{R}^3$, illustrated in Figure 5.6 (c), obtained by sampling a ship hull B-spline geometry according to the uniform random distribution.

In this example, we also store the exact sampling parametric values in $[0, 1]^2$, in order to exploit the properties of the method and reduce the influence of the



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

Figure 5.6: Nefertiti (a), face (b), ship hull (c) and wind turbine (d) scattered point clouds characterized by 8818 (a), 9283 (b), 9636 (c-d) points.

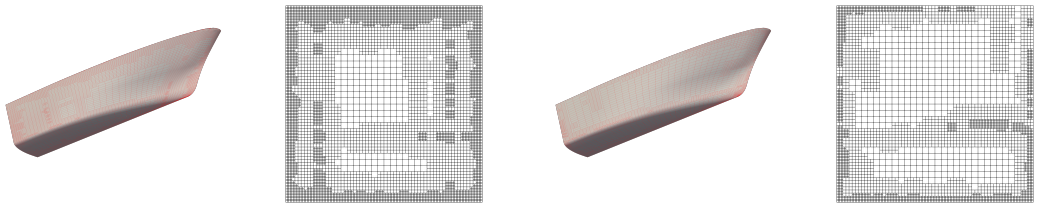
initial parametrization. On the contrary, in all the following numerical results, we assume the parametric values to be unknown, and we will consequently compute them, showing in addition how a good initial parameterization is fundamental.

By performing the adaptive LS fitting algorithm proposed in [101] with bi-degree $\boldsymbol{d} = (2, 2)$, the final reconstructed geometry has 7173 DOFs, i. e. 74% of the points, and the THB-spline model accuracy is characterized by a MSE equal to $5.54\text{e} - 5$. By enriching the previous method with the moving parameter schemes, i. e. performing the adaptive A-PDM method, the final geometry is characterized by 4277 degrees of freedom, i. e. 44% of the points, and a MSE of $4.25\text{e} - 6$. Thus, introducing the PC scheme inside the adaptive loop results in a considerably smaller and more accurate THB-spline model, with about 40% of DOFs less. The final THB-spline models and the relative hierarchical meshes are illustrated in Figure 5.7.

Example 5.2.2 (b)

Adaptive A-PDM with PARCNN

In this experiment, we exploit the performance of the PARCNN parameterization model, introduced in Section 4.1, with the adaptive A-PDM fitting scheme. We consider the gridded point cloud consisting of 3025 items, shown in Figure 4.7 (bottom line, center) and we compute its initial parameterization with the PARCNN model. Subsequently, we perform the A-PDM fitting scheme. The resulting hierarchical mesh and the THB-spline approximation are shown in Figure 5.8. In particular, the final geometry has 1349 DOFs and an accuracy



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

Figure 5.7: The hierarchical THB-spline models (a-c) and the corresponding hierarchical meshes (b-d) for the ship hull point cloud of dimension 9636 obtained with adaptive LS (a-b) and adaptive A-PDM (c-d) in Example 5.2.2 (a).

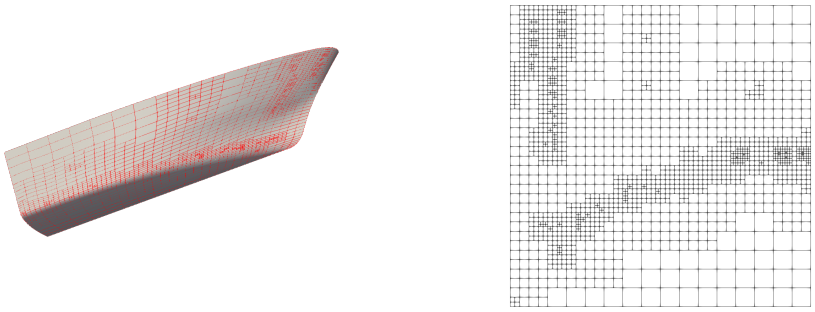


Figure 5.8: Reconstructed geometric spline model (left) and hierarchical mesh (right) of the ship hull point cloud of dimension 3025 via the adaptive scheme with THB-splines in Example 5.2.2 (b).

characterized by a MSE equals to $2.02e-11$. Note that the degrees of freedom are properly distributed along the parametric domain Ω, by following the sharp feature and high curvature regions of the geometry, as shown in Figure 5.8.

Example 5.2.2 (c)

Adaptive A-PDM with BIDGCN

In this example, we investigate the generalization capabilities of the BIDGCN parameterization, proposed in Section 4.3, to spline configurations of different bi-degrees for the reconstruction of different point clouds.

We start by performing the adaptive A-PDM fitting scheme with THB-splines of a human face model of 9283 points, shown in Figure 5.6 (b).

In particular, we reconstruct the THB-spline models of the input point cloud by performing the adaptive A-PDM fitting scheme both for bidegree $(3,3)$ and $(4,4)$. In the first case, we start from a tensor-product space with bi-degree $\boldsymbol{d} = (3,3)$ and perfom 8 adaptive iterations, each of them characterized by 1 step of PC. The smoothig weight λ is set to $1e{-}6$ and the refinement threshold is $\epsilon = 8e{-}4$. 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 approximate the input point cloud registering a MSE of $1.45e{-}8$, a MAX error of $1.03e{-}3$, and the DHD from the point cloud to the surface is $3.43e{-}2$. In the second case, the algorithm settings are the same of the previous configuration, except for the bi-degree, which is not set to $\boldsymbol{d} = (4,4)$, and the refinement threshold, which is lowered to $6.5e{-}4$. 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 MSE of $1.44e{-}8$, a maximum error of $1.07e{-}3$, and the DHD from the point cloud to the surface is $3.23e{-}2$. 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 Figure 5.9.

We conclude this example by considering two additional point clouds of 9636 items collected from a ship hull and a wind turbine, shown in Figure 5.6 (c) and (d). In particular, we approximate the two scattered data sets by performing 8 iterations of the adaptive A-PDM fitting algorithm, starting from a tensor-product polynomial space with bi-degree $\boldsymbol{d} = (5,2)$. The point clouds, the reconstructed THB-spline model, the point wise error distribution on the scattered data and the hierarchical meshes are displayed in Figure 5.10 for both the ship hull (top) and the wind turbine (bottom).

As concerns the ship hull data, we set the smoothig weight $\lambda = 1e{-}6$ and the refinement threshold is $\epsilon = 1e{-}5$. 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 MSE equal to $1.30e{-}10$, a MAX error equal to $2.27e{-}4$, and registers a DHD from the points to the surface equals to $8.45e{-}2$.

Finally, for the wind turbine point cloud we set the smoothig weight as $\lambda = 1e{-}6$ and the refinement threshold as $\epsilon = 5e{-}5$. The final THB-spline model has 5 uniformly refined and 3 locally refined hierarchical levels, 2449 degrees of freedom, equivalent in magnitude to 25% of the number of input data, a MSE of $2.13e{-}10$, a MAX error of $4.57e{-}4$, and DHD from the points to the surface of $1.66e{-}1$.

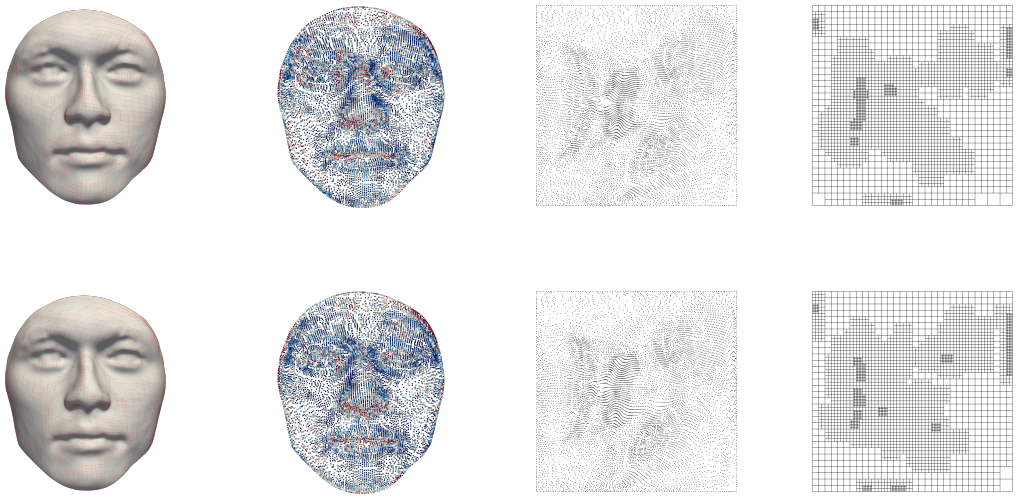


Figure 5.9: Hierarchical spline model reconstruction of the face point cloud shown in Figure 5.6 (b) for bidegree $\boldsymbol{d} = (3,3)$ (top) and $\boldsymbol{d} = (4,4)$ (bottom) using the BIDGCN parameterization. The reconstructed THB-spline models are shown together with the scaled error distributions on the point cloud, the computed parametric values, and the corresponding hierarchical meshes (from left to right) for Example 5.2.2 (c).

Example 5.2.2 (d)

Comparison between adaptive A-PDM with LPSP and BIDGCN

In this example, we process the point cloud representing a Nefertiti face model, of size 8818, shown in Figure 5.6 (a) to perform a comparison between the standard parameterization methods described in Section 4.2.1 and the learning BIDGCN parameterization method proposed in Section 4.3. 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 already illustrated in Figure 4.30, we execute a heuristic search for $r = \frac{3}{\sqrt{m}}, 0.05, 0.075, 0.1, \ldots, 0.25, 0.275, 0.3$, by computing the polynomial approximation of bi-degree $\boldsymbol{d} = (2,2)$, and select the radius r giving the best MSE. By analyzing the value of the error with respect to r, we decide to fix the radius as defined in (4.16), i. e. $r = \frac{3}{\sqrt{8818}}$, since it leads to the best approximation result. We then parameterize the input scattered point cloud with the standard LPSP method as well as the learning BIDGCN method and subsequently reconstruct hierarchical spline model by performing the adaptive A-PDM fitting scheme for

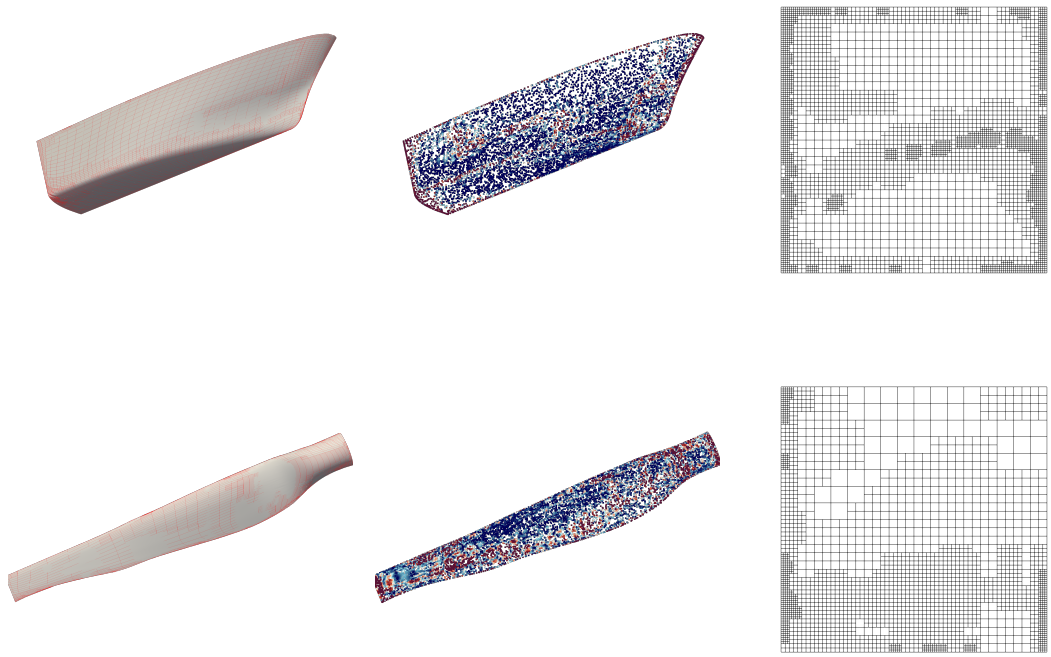


Figure 5.10: The hierarchical spline model reconstruction, the scaled error distribution and the hierarchical mesh for a ship hull (top) and a wind turbine (bottom) using the BIDGCN parameterization with bi-degree $\boldsymbol{d} = (5,2)$ in Example 5.2.2 (c).

bi-degree $\boldsymbol{d} = (2,2)$. Note that among all the listed values for the radius r of the LPSP method that we tested on this point cloud, the choice $r = \frac{3}{\sqrt{m}}$ gives the best results in terms of biquadratic polynomial approximation. We start from a tensor-product space with bi-degree $\boldsymbol{d} = (2,2)$ and perform 8 adaptive iterations, each of them characterized by 1 step of PC. To properly compare the results, the refinement tolerances are chosen such that the final THB-spline models have (almost) the same number of degrees of freedom. Both BIDGCN and LPSP are characterized by 4 uniformly refined and 4 locally refined hierarchical levels. Moreover, when the LPSP 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 MSE of 5.50e−6, a MAX error of 3.34e−2, and DHD 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 MSE of 1.75e−6,

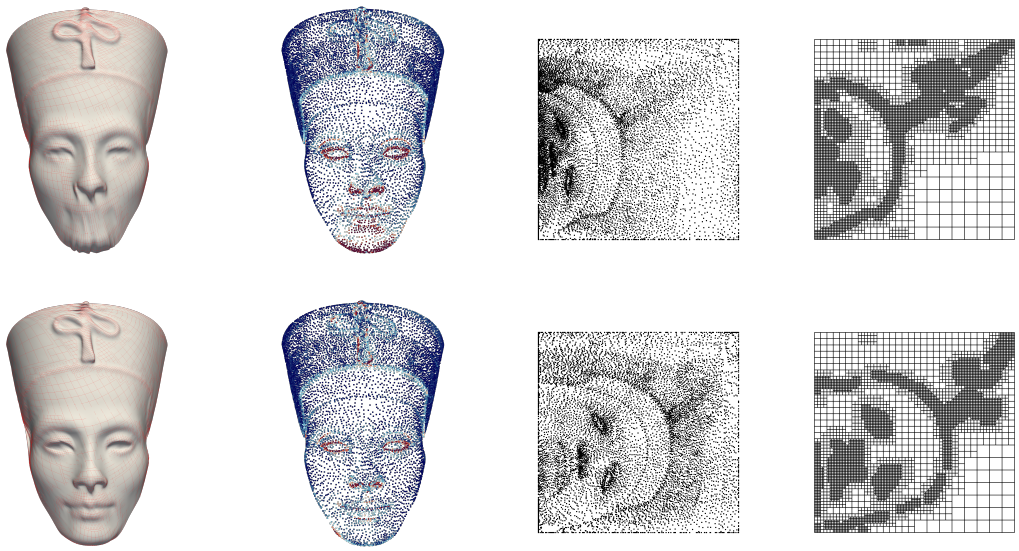


Figure 5.11: Hierarchical spline model reconstruction of the Nefertiti point cloud shown on left of Figure 5.6 for LPSP parameterization with 5492 degrees of freedom (top), as well al for the BIDGCN parameterization with 5288 degrees of freedom (bottom) with bi-degree $\boldsymbol{d} = (2, 2)$. The reconstructed 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).

MAX error of 1.41e−2, and DHD from the data points to the surface of 3.46e−1. The THB-spline approximations, scaled error distributions on the point cloud, parameter values, and hierarchical meshes obtained with LPSP and BIDGCN parameterizations are shown on the top and bottom of Figure 5.11, 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.

Example 5.2.2 (e)

Adaptive A-PDM with type II markers and BIDGCN

In this example, we perform the reweighted adaptive spline least squares fitting scheme proposed in Section 3.1.8, enriched by the moving parameterization technique, to approximate a scattered point cloud of 9000 points representing

a Nefertiti bust, illustrated in Figure 5.12 (top-left). Firstly, we parameterize the input data with the BIDGCN method proposed in Section 4.3. In addition, to tackle the presence of corrupted data, we exploit the use of the weights associated to the data. More precisely, together with the points, we are given a set of markers of the second type K_{II}, highlighted in black in Figure 5.12 (bottom-left) which we use to perform the reweighted adaptive scheme. Due to the complexity of the problem, we decide not to update the markers K_{II} but to keep them fixed over all the iterative scheme, by setting tol_{II} sufficiently small. We then compare the results obtained by the proposed rWLS with the ordinary adaptive least squares scheme LS, where all the weights set to one. Note that both schemes are enriched by the moving parametrization technique, included within the refinement procedure. Such a comparison is illustrated in Figure 5.12, where we show on the top the hierarchical approximation obtained by LS and on the bottom the one obtained by rWLS. In particular, the region corresponding to the corrupted data is better approximated by the rWLS method, which results free of self-intersections and it is smoother than the one obtained with LS. Finally, the MAX error registered by LS is $3.28\text{e}-2$, whereas the MAX of rWLS is $1.57\text{e}-2$.

Example 5.2.2 (f)

Comparison of A-PDM, A-HDM

In this last example, we analyze the behaviour of the adaptive A-PDM and adaptive A-HDM with two different choices of blending weights, the weights proposed in [10], and the constant choice. We adaptively approximate the ship hull point cloud already considered in Section 5.1.3, consisting of 3025 gridded data points and initial CEN parameterization.

As initial settings, we choose the polynomial bi-degree $\boldsymbol{d} = (2, 2)$, the initial 5×2 tensor-product mesh, smoothing weight $\lambda = 1\text{e}-7$, and we set the refinement threshold as $\epsilon = 1\text{e}-4$. Moreover, based on the weights investigations conducted in Section 5.1.3, we select the constant weight choice $\gamma = 1\text{e}-2$. We run the adaptive alternating fitting methods for 1 and 5 PC steps performed at each adaptive loop. The results in terms of MSE are shown in Figure 5.13.

We can observe that the A-HDM method with constant weight choice (orange line) shows better approximation performance with respect to A-PDM, whereas this is not the case for the A-HDM method with curvature-based weights as in [10], for 1 PC step. This is in line with the results presented in Section 5.1.3, specifically in Figure 5.3. After 1 PC step, the A-HDM methods tend not to be in a convergence regime, hence initiating the refinement on a bad approximation leads to a worse final result; see also the considerations in Section 5.2.1.

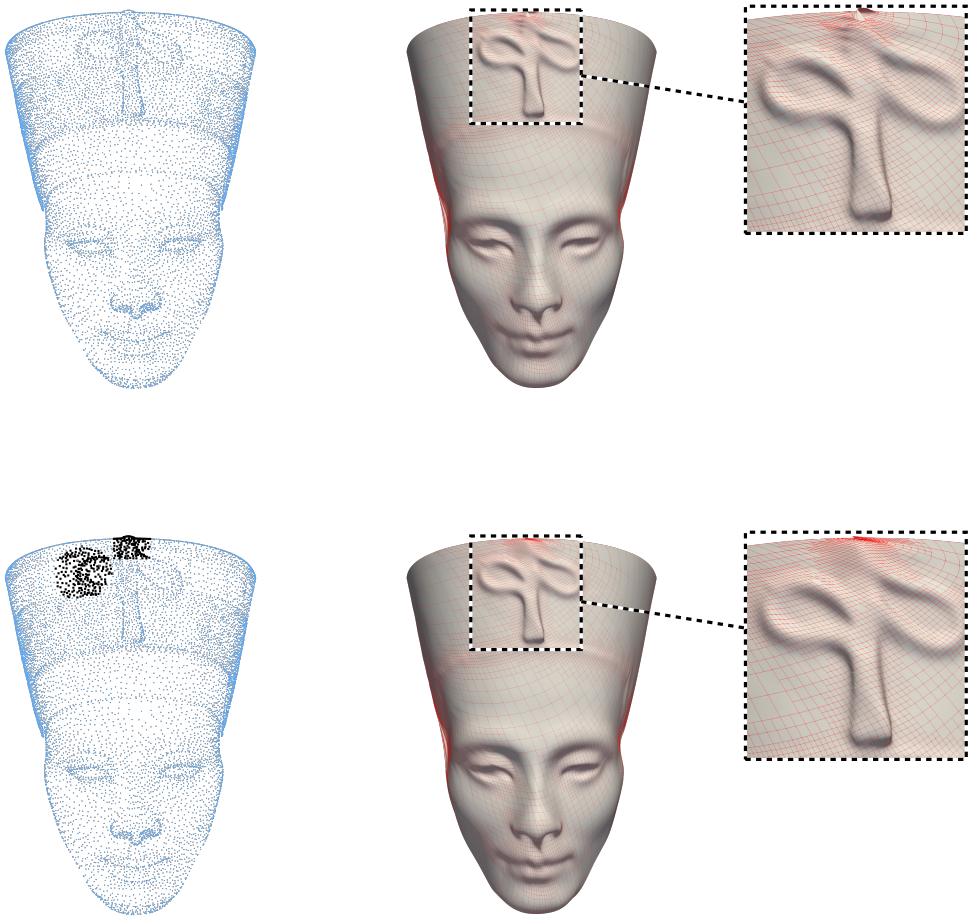


Figure 5.12: Adaptive surface fitting experiment described in Example 5.2.2 (e) using adaptive LS (top) and rWLS (bottom) for data with markers of type II. The data point clouds and the marked points are also shown.

We finally run all the adaptive alternating algorithms with 5 PC steps. In this case, also adaptive A-HDM with curvature-based weights shows good approximation performance, better than A-PDM. Note that the accuracy gain is kept for all the weight choices during the adaptive refinement. In particular, the constant choice of the weights results in the best choice for this example. The final geometries obtained with 5 PC steps are in Figure 5.14, together with their respective number of DOFs and MSE values. More specifically, the least number of DOFs and the lowest MSE are registered by A-HDM with constant weights. When using curvature-based weights, A-HDM requires around 35% more DOFs

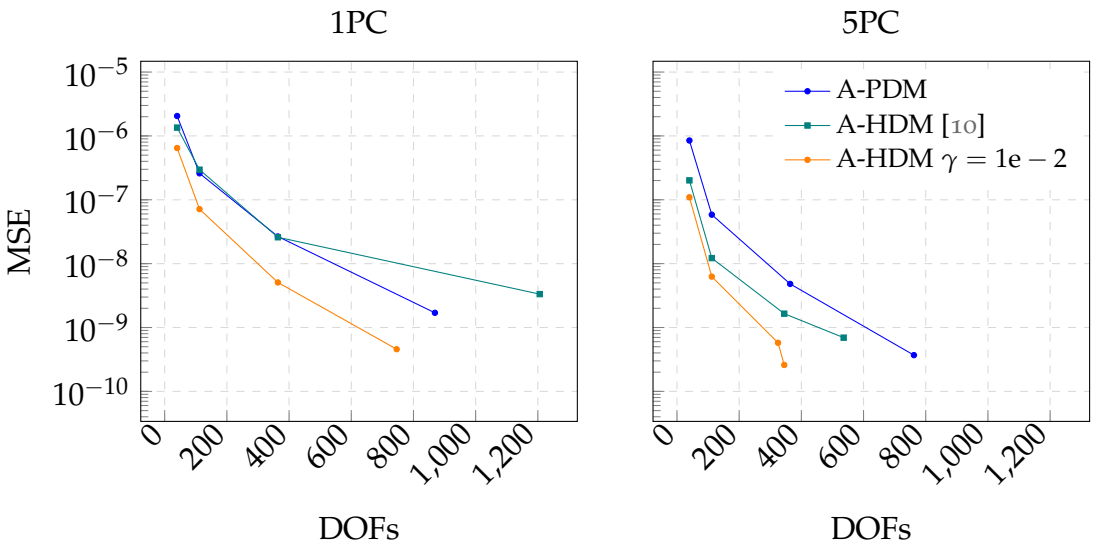
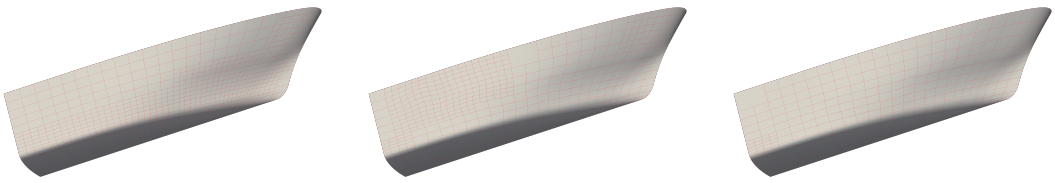


Figure 5.13: Adaptive spline approximation for the ship hull point cloud consisting of 3025 gridded data points parametrized with CEN method. Comparison of MSE between adaptive A-PDM (blue), A-HDM with the weights as in [10] (teal) and A-HDM with constant choice of the weights, for $\gamma = 1e-2$ (orange), for 1PC (left) and 5PC (right) steps within the adaptive loop.

to attain the same level of accuracy, whereas A-PDM requires about 55% more DOFs to achieve the same level of precision, yet registering a higher MSE.

5.3 ADAPTIVE FITTING WITH JOINT-OPTIMIZATION

Moving parameterization methods consider the point parameters as variables that might be adjusted during the fitting process in order to optimize the final geometry. The optimization problem proper of the alternating methods discussed in the previous Sections is linear with respect to the control points of the spline surface $\boldsymbol{s}$ and non-linear with respect to the point parameters $\mathcal{U}$. Going one step further, we propose in this section a Joint Optimization approach to solve simultaneously the parameterization and fitting problems for a fixed spline space, i. e. to address the problem in (5.1). Subsequently, we extend this surface fitting optimization method to the adaptive spline framework. More precisely, we address the parameterization problem within the first step (SOLVE) of the adaptive loop by *simultaneously* computing the optimal parameter sites $\mathcal{U}$



A-PDM
DOFs = 762
MSE = 3.70e − 10

A-HDM [10]
DOFs = 536
MSE = 6.94e − 10

A-HDM $\gamma = 1e-2$
DOFs = 345
MSE = 2.59e − 10

Figure 5.14: Adaptive spline approximation for the ship hull point cloud consisting of 3025 gridded data points and initial CEN parameterization in Example 5.3.3 (a). From left to right: final THB-spline model with adaptive A-PDM, A-HDM and the weights choice in [10], and A-HDM and constant weights $\gamma = 1e-2$, with 5PC steps within each adaptive loop. The DOFs and MSE are alro reported.

for $\mathcal{P}$ and control points for $\boldsymbol{s}$. Therefore, with this method, we avoid solving a linear system of equations and performing PC routines at every adaptive iteration.

The spline fitting problem is here addressed as a non-linear minimization problem, whose Jacobian can be explicitly computed by exploiting the properties of the B-spline basis functions, see Chapter 2 and the references therein. In particular, the explicit computation of the Jacobian matrix enables the use of numerical optimization techniques. Since the explicit calculation of the Hessian matrix can potentially be a costly procedure, we consider an optimization framework based on the Limited memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) optimizer [24, 121], which exploit the Broyden-Fletcher-Goldfarb-Shanno (BFGS) for the Hessian corrections, with the difference that these corrections are stored separately, and when the available storage is complete, the oldest correction is deleted to make space for the new one. The benefits of exploiting LBFGS-based methods can be particularly appreciated to solve problems in which the Hessian matrices are large, unstructured, dense, or even unavailable.

5.3.1 *Limited memory BFGS optimization*

In the following, we provide some background knowledge on the LBFGS method, used in our fitting method to minimize (5.1). Let $f : \mathbb{R}^n \to \mathbb{R}$ be a non-linear function, and let $g : \mathbb{R}^n \to \mathbb{R}^n$ be its *known* gradient. The LBFGS method

is a quasi-Newton algorithm for solving large, i. e. $n \gg 50$ [24], non-linear optimization problems, namely

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}), \tag{5.18}$$

where $\boldsymbol{x}$ is the set of unknown variables. The LBFGS method approximates the inverse of the Hessian matrix of the objective function H_k by a sequence of gradient vectors from previous iterations. The user specifies the number ν of BFGS corrections that are to be kept and provides an initial sparse symmetric and positive definite matrix H_0^0, which approximates the inverse of the Hessian of f. During the first ν iterations, the LBFGS method is identical to the BFGS method. For $k > \nu$, H_k is obtained by applying m BFGS updates to H_k^0 using information from the ν previous iterations. More precisely, at each iteration k, the current iterate $\boldsymbol{x}_k$, the function value f_k, the gradient $\boldsymbol{g}_k$, and the approximation of the Hessian H_k are given; therefore, this allows us to compute

$$\begin{aligned}
\boldsymbol{s}_k &= \boldsymbol{x}_{k+1} - \boldsymbol{x}_k, \\
\boldsymbol{y}_k &= \boldsymbol{g}_{k+1} - \boldsymbol{g}_k, \\
\rho_k &= \frac{1}{\boldsymbol{y}_k^\top \boldsymbol{s}_k}, \\
M_k &= I - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^\top, \\
H_k^0 &= \frac{\boldsymbol{s}_{k-1}^\top \boldsymbol{y}_{k-1}}{\boldsymbol{y}_{k-1}^\top \boldsymbol{y}_{k-1}} I
\end{aligned}$$

and thereafter,

$$\begin{aligned}
H_k = &\left(M_{k-1}^\top \dots M_{k-m}^\top\right) H_k^0 \left(M_{k-m} \dots M_{k-1}\right) \\
&+ \rho_{k-m} \left(M_{k-1}^\top \dots M_{k-m+1}^\top\right) \boldsymbol{s}_{k-m} \boldsymbol{s}_{k-m}^\top \left(M_{k-m+1} \dots M_{k-1}\right) \\
&+ \rho_{k-m+1} \left(M_{k-1}^\top \dots M_{k-m+2}^\top\right) \boldsymbol{s}_{k-m+1} \boldsymbol{s}_{k-m+1}^\top \left(M_{k-m+2} \dots M_{k-1}\right) \\
&\vdots \\
&+ \rho_{k-1} \boldsymbol{s}_{k-1} \boldsymbol{s}_{k-1}^\top.
\end{aligned}$$

Subsequently, at the end of the k-th iteration of LBFGS the variables are updated by

$$\begin{aligned}
\boldsymbol{d}_k &= -H_k \boldsymbol{g}_k, \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \gamma_k \boldsymbol{d}_k,
\end{aligned}$$

where γ_k is a scalar variable controlling the step-size, which satisfies the Wolfe conditions [183, 184], i. e.

$$\begin{aligned} f\left(\boldsymbol{x}_k + \gamma_k \boldsymbol{d}_k\right) &\leq f\left(\boldsymbol{x}_k\right) + c_1 \gamma_k \boldsymbol{g}_k^\top \boldsymbol{d}_k, \\ \boldsymbol{g}\left(\boldsymbol{x}_k + \gamma_k \boldsymbol{d}_k\right)^\top \boldsymbol{d}_k &\geq c_2 \boldsymbol{g}_k^\top \boldsymbol{d}_k, \end{aligned}$$

with $0 < c_1 < c_2 < 1$. Finally, the LBFGS algorithm terminates when the gradient of the objective function is smaller than a specified input tolerance, i. e. $\|\nabla f\| < \eta$ or a maximum number of iterations $K_{\max}$ is reached.

Remark 27. *The LBFGS algorithm uses a sequence of m gradient vectors to approximate the inverse of the Hessian matrix. Thus, a large value of m can potentially lead to a more accurate guess while simultaneously increasing the computational costs. According to the literature, the default suggested value to be used is* $\nu = 20$ *[140, 189].*

5.3.2 *Adaptive spline fitting with LBFGS*

We employ the LBFGS algorithm to address the problem of spline surface fitting. In particular, given a point cloud $\mathcal{P}$ as in (1.1), so that $\mathcal{P} = \mathcal{P}_I \dot{\cup} \mathcal{P}_B$, i. e. disjoint union between interior points and boundary points, for a fixed spline space $V = \operatorname{span}\{\beta_0, \ldots, \beta_n\}$ of dimension $n+1$, our goal consists in finding a spline model $\boldsymbol{s} \in V$, which satisfied (5.1). The outline of the proposed optimization approach consists in the following three steps:

1. define an initial parameterization $\mathcal{U}$;
2. define the initial control points, hence the initial spline geometry $\boldsymbol{s}_0 \in V$;
3. run the HLBFGS optimizer until convergence.

Note that there are no requirements concerning the initial guesses in steps 1 and 2. For instance, any method discussed in Chapter 4 can be suitably applied to compute the initial parameterization, as any fitting method presented in Chapter 3 or in the first part of the present Chapter to compute the initial spline configuration. Nevertheless, both the chosen parameterization and spline geometry need to be *good enough* to guarantee fast convergence to a more accurate solution.

The unknowns of this problem are both the coefficients of the spline model and the parametric sites $\mathcal{U}$ associated to the data points. We then arrange them in the matrices

$$\boldsymbol{c} \in \mathbb{R}^{(n+1)\times N} \quad \text{and} \quad \boldsymbol{u} \in \mathbb{R}^{m \times D},$$

respectively. Moreover, we consider the collocation matrix $B = B(\mathcal{U})$ as in (3.4), and we obtain the matrix form of the objective function (5.1), i. e.

$$f(\boldsymbol{c}, \boldsymbol{u}) = \|B\boldsymbol{c} - \boldsymbol{p}\|_F^2 + \lambda J(\boldsymbol{c}) = \text{trace}\left\{[B\boldsymbol{c} - \boldsymbol{p}]\,[B\boldsymbol{c} - \boldsymbol{p}]^\top\right\} + \lambda J(\boldsymbol{c}), \quad (5.19)$$

where $\|\cdot\|_F$ represents the Frobenius norm, $\boldsymbol{p} \in \mathbb{R}^{m\times N}$ is a matrix containing the data points $\mathcal{P}$, and $J(\boldsymbol{c})$ represents the contribution of the smoothing term. In order to exploit the LBFGS optimizer, we need to provide the gradient of the objective function (5.19), namely to compute the (partial) derivative of our fitting spline object with respect to each coefficient as well as with respect to each parametric site. More precisely,

$$\nabla f(\boldsymbol{c}, \boldsymbol{u}) = \left(\frac{\partial f}{\partial c_0^{(1)}}, \dots, \frac{\partial f}{\partial c_n^{(1)}}, \dots, \frac{\partial f}{\partial c_0^{(N)}}, \dots, \frac{\partial f}{\partial c_n^{(N)}}, \frac{\partial f}{\partial u_1^{(1)}}, \dots, \frac{\partial f}{\partial u_m^{(1)}}, \dots, \frac{\partial f}{\partial u_1^{(D)}}, \dots, \frac{\partial f}{\partial u_m^{(D)}}\right), \quad (5.20)$$

hence, by letting G as in (5.6),

$$\begin{aligned} \nabla_{\boldsymbol{c}} f(\boldsymbol{c}, \boldsymbol{u}) &= \left(B^\top B + \lambda G\right)\boldsymbol{c} - B^\top \boldsymbol{p} \\ \nabla_{u^{(k)}} f(\boldsymbol{c}, \boldsymbol{u}) &= \frac{1}{2}\nabla_{u^{(k)}} \text{trace}\left((B\boldsymbol{c} - \boldsymbol{p})\,(B\boldsymbol{c} - \boldsymbol{p})^\top\right) \\ &= \text{diag}\left((B\boldsymbol{c} - \boldsymbol{p})\left(\partial_{u^{(k)}} B\boldsymbol{c}\right)^\top\right), \quad k = 1, \dots, D. \end{aligned} \quad (5.21)$$

Similarly to Remark 22, we are again dealing with the disjoint union of interior and boundary points and, consequently, with interior and boundary parameters, i. e. $\mathcal{U} = \mathcal{U}_I \dot\cup \mathcal{U}_B$. In particular, in this case, we constrained the boundary parameters $\mathcal{U}_B$ to live only on the boundary curve they belong to. This can be achieved by discarding from the optimization variables, hence also from (5.20), the parametric directions $u_j^{(k)}$, for $j = n+1, \dots, m$ and some $k = 1, \dots, D$, pointing towards the interior of Ω. Figure 5.15 illustrates the boundary constraints in the case of $D = 2$ and $\Omega = [0,1]^2$, with 4 boundary edges. More precisely, the parametric site $\boldsymbol{u}_i = (u_i^{(1)}, u_i^{(2)})$ belongs to the west edge of the domain; hence, the only component of $\boldsymbol{u}_i$ that plays a role in the optimization problem (5.1) is $u_i^{(2)}$, whereas $u_i^{(1)}$ is kept fixed to 0. Similarly, $\boldsymbol{u}_j$ belongs to the south edge of the domain; hence, we optimize $u_j^{(1)}$ while we set $u_j^{(2)} = 0$.

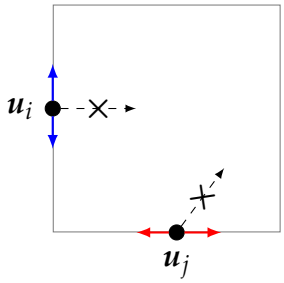


Figure 5.15: Two parametric sites u_i and u_j respectively constrained to the west and the south boundary edge of the rectangular domain Ω. The only possible directions are highlighted by blue arrows for u_i and red arrows for u_j. In addition, not acceptable directions are represented by dashed arrows, marked by a cross symbol.

The advantage of the proposed fitting method consists in avoiding the computation of the foot-point projections and solving a very large linear system of equations at every time step. For completeness, we recall that a B-spline curve fitting method based on LBFGS has been developed in [189].

In this Section, we extend the joint optimization fitting method to the adaptive THB-spline framework. In this way, we provide a new strategy to embed the moving parameterization within adaptive spline fitting schemes.

Given a point cloud $\mathcal{P} = \mathcal{P}_I \dot{\cup} \mathcal{P}_B$, a suitable parameterization $\mathcal{U}$, and a THB-spline geometry $s \in V$ as in (2.15), belonging to a hierarchical spline space V, at the beginning of each adaptive routine (1.4), we perform the joint optimization fitting algorithm for the current fixed spline space V, using as initial guesses the parameter and the control points coming from the previous adaptive loop. Subsequently, the output of the LBFGS optimizer defines the new control points $c_j \in \mathbb{R}^N$, for $j \in \mathcal{A}_k^\ell$ and $\ell = 0, \dots, L-1$, of the spline geometry $s \in V$ and the new parametric sites $\mathcal{U}$ of the data points, which are not kept fixed over the adaptive fitting scheme, but updated at each iteration. The remaining three adaptive steps are similar to the ones implemented for the adaptive alternating methods. To summarize, the optimal geometry s is evaluated on the new parametric sites $\mathcal{U}$ and the point-wise error estimator is computed, namely

$$\|s(u_i) - p_i\|_2 \,, \text{ for each } i = 1, \dots, m.$$

We identify the cells of the current hierarchical level ℓ, which contain the parameters u_i so that $\|s(u_i) - p_i\|_2 > \epsilon$, for a certain input error threshold $\epsilon > 0$, and mark them for refinement, together with two surrounding rings of

Algorithm 8: Adaptive joint optimization fitting scheme.

Input: Point cloud and parameters $\{\boldsymbol{u}_i, \boldsymbol{p}_i\}_{i=1}^m$, an initial tensor-product B-spline space V^0, the error tolerance $\epsilon > 0$, a maximum number of hierarchical levels L.

Compute the initial tensor-product LS B-spline approximation $\boldsymbol{s} \in V^0$ and the point-wise errors $e_i = \|\boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i\|_2$, for each $i = 1, \ldots, m$ and set loop $= 0$

while $\max_i e_i > \epsilon$ *and* loop $<$ L **do**

  Solve the non linear least squares problem

$$\boldsymbol{s} = \underset{\substack{\boldsymbol{c}_j,\ j\in\mathcal{A}^\ell,\ell=0,\ldots,\text{loop} \\ \boldsymbol{u}_i\in\Omega, i=1,\ldots,m}}{\arg\min} \frac{1}{2}\sum_{i=1}^{m} \|\boldsymbol{s}\,(\boldsymbol{u}_i) - \boldsymbol{p}_i\|_2^2 + \lambda J(\boldsymbol{c}).$$

  Compute the errors $e_i = \|\boldsymbol{s}(\boldsymbol{u}_i) - \boldsymbol{p}_i\|_2$, for $i = 1, \ldots, m$.

  Mark the domain elements where $e_i > \epsilon$.

  Refine the marked cells and two surrounding rings of cells.

  Update the hierarchical mesh $\mathcal{M}$ and hierarchical space V.

  Project the solution $\boldsymbol{s}$ onto the refine space V.

  Set loop $=$ loop $+ 1$.

end

Output: $\boldsymbol{s} \in V$, the THB-spline approximant.

cells in the hierarchical mesh. Finally, the marked cells are dyadically split in order to refine the hierarchical space.

Remark 28. *After refinement, the hierarchical space has been updated by the insertion of a new hierarchical level, and a new iteration of the adaptive loop can potentially begin. Note that, in order to initialize the HLBFGS optimizer on the enlarged hierarchical spline space, we need to project the solution from the previous iteration to the new spline space.*

Finally, we suggest as automatic approach to generating the initial tensor-product B-spline fitting geometry to solve the (weighted) LS problem for a given input parameterization, see Section 3.1. The joint optimization routine is summarized in Algorithm 8.

5.3.3 *Numerical results*

In this Section, we present a comparison between the adaptive fitting schemes with moving parameterization presented in this Chapter, i. e. the adaptive A-PDM, A-HDM methods of Section 5.2, and the adaptive J-PDM of Section 5.3. In Example 5.3.3 (a) we approximate a ship hull point cloud of 3025 gridded data, for which we investigate different fine-tuning strategies for adaptive J-PDM. In Example 5.3.3 (a), we approximate a real-world data set of 9000 points representing a Nefertiti bust. From both examples, the J-PDM method results in the best approximation accuracy from a qualitative and quantitative perspective, being on the other hand the most computationally expensive.

The algorithms related to the adaptive fitting schemes with THB-splines have been implemented in C++ with the open-source G+Smo library [94, 129]. The developed code for the proposed algorithms has been integrated and will be available in the next releases. As concerns the implementation of the LBFGS optimization method, we exploit the HLBFGS C++ library [124], which provides the LBFGS method, the preconditioned LBFGS method [93], and the preconditioned conjugate gradient method [150, 151].

Example 5.3.3 (a)

Comparison on the ship hull point cloud

In this example, we revisit the adaptive approximation analyzed in Example 5.2.2 (f). We start with the same initial configuration, i. e. we approximate a ship hull point cloud consisting of 3025 gridded data points and initial CEN parameterization, starting with a tensor-product B-spline of bi-degree $\boldsymbol{d} = (2, 2)$ on a 5×2 mesh. Differently from Example 5.2.2 (f), we lower the refinement threshold to $\epsilon = 1\mathrm{e}-5$ to induce more refinement levels.

As concerns the adaptive A-PDM we perform 10 steps of PC within each adaptive loop. As concerns adaptive J-PDM, we exploit two configurations for its hyper-parameters, namely the tolerance on the norm of the gradient ν and the maximum number of iterations $K_{\max}$.

(a) In this first configuration we set $\eta = 1\mathrm{e}-6$ and $K_{\max} = 500$.

(b) In this second configuration, we set $\eta_0 = 1\mathrm{e}-5$ and $K^0_{\max} = 2^\ell \cdot 200$ and adaptively change them with the number of hierarchical level, i. e.

$$\eta^\ell = \frac{\eta_0}{2^\ell}, \text{ and } K^\ell_{\max} = 2^\ell \cdot K^0_{\max}, \tag{5.22}$$

for $\ell = 0, \dots, L-1$.

The configuration in (b) arises from the idea that spline space V poses a constraint to the solution of (5.1). In particular, if the approximation space V is too coarse, it is not reasonable to spend computational resources, by setting ν too low or $K_{\max}$ too high, to find an optimal solution in V, which will not satisfy the refinement threshold ϵ and will be anyway recomputed after refinement. On the other hand, when the space V enlarges, then it is more plausible that the solution satisfying the input tolerance ϵ belongs to it; hence, it is valuable to search for an optimum within this space. Higher accuracy is achieved by lowering the threshold ν; consequently, we also increase the LBFGS maximum number of iterations $K_{\max}$ in order not to stop the optimization routine too early. Note that a too low choice for η^0 and $K^0_{\max}$ might result in a geometry with too poor quality at the end of the first loop of the adaptive scheme, which would also affect the refinement and the whole adaptive approximation results, similarly to the analysis in Section 5.2.1.

The results are shown in Figure 5.16 in terms of MSE (left) and MAX (right) for the adaptive A-PDM (solid blue line), J-PDM for the configuration (a) (solid magenta line), and J-PDM for the configuration (b) (dashed magenta line). We can observe that both J-PDM configurations show considerably better approximation accuracy with respect to A-PDM in terms of MSE and MAX errors. More specifically, we can note that the convergence rate of the adaptive J-PDM with the hyper-parameter configuration in (b) overcomes all the other schemes, satisfying the input tolerance ϵ with less than 400 DOFs, whereas in all the other configurations more than 1000 DOFs are required, up to 2329 DOFs for adaptive A-PDM.

The reconstructed geometries are shown in Figure 5.17, from left to right for A-PDM, J-PDM (a), and J-PDM (b), together with their DOFs, MSE and MAX values.

Example 5.3.3 (b)

Comparison on the Nefertiti real-wrold point cloud

In this last example, we compare the adaptive fitting methods with moving parameterization on a real-world dataset, corresponding to 9000 scattered points representing a Nefertiti bust already considered in Example 5.2.2 (e), and we compute the initial parameterization with the BIDGCN method, presented in Chapter 4. We start the adaptive loop from a polynomial space of bi-degree

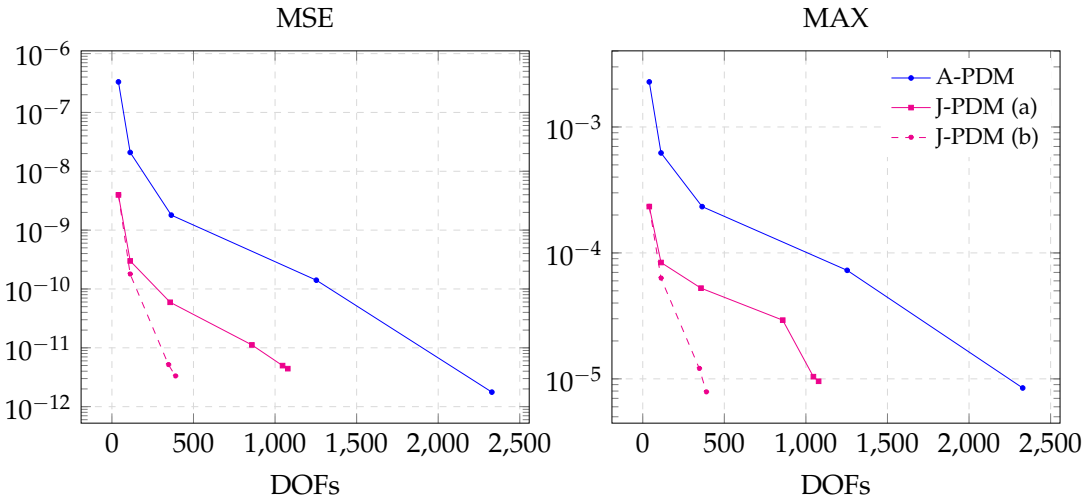
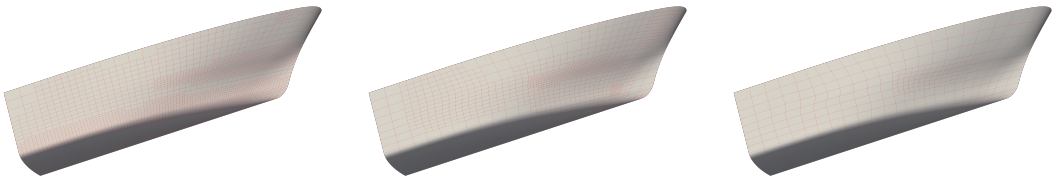


Figure 5.16: Adaptive spline approximation for the ship hull point cloud consisting of 3025 gridded data points and initial CEN parameterization in Example 5.3.3 (a). Comparison of MSE (left) and MAX (right) between adaptive A-PDM (blue) and the J-PDM approach with constant hyper parameters (solid magenta) and adaptive hyper parameters (dashed magenta).



A-PDM	J-PDM (a)	J-PDM (b)
DOFs = 2329	DOFs = 1078	DOFs = 634
MSE = 1.76e − 12	MSE = 4.21e − 12	MSE = 3.29e − 12
MAX = 8.48e − 6	MAX = 9.56e − 6	MAX = 7.84e − 6

Figure 5.17: Adaptive spline approximation for the ship hull point cloud consisting of 3025 gridded data points and initial CEN parameterization in Example 5.3.3 (a). From left to right: final THB-spline model with adaptive A-PDM, J-PDM (a) and J-PDM (b). The DOFs, MSE and MAX values are also reported.

$\boldsymbol{d} = (2, 2)$ and set the refinement threshold to $\epsilon = 2.5\text{e} - 3$. The input point cloud is characterized by high-curvature regions; hence, the point projection involved in the adaptive alternating methods is very challenging. Consequently, performing too many PC steps affects the quality of the final reconstructed geometry. For this reason, we perform only 1 PC step within each adaptive

loop of the alternating methods. By the analysis conducted in Section 5.1.3 and Example 5.2.2 (f), adaptive A-HDM method with curvature-/error-based blending weights does not show a stable behaviour after only 1 step of PC, hence proceeding with the refinement ruins the accuracy of the final reconstructed hierarchical model. For this reason, we consider A-HDM with constant blending weights, with $\gamma = 1/3$, chosen after an appropriate fine-tuning. As concerns adaptive J-PDM, we set $\eta = 1e-5$ and $K_{\max} = 100$ and we keep them constant.

The results in terms of MSE and MAX are illustrated in Figure 5.18, whereas Figure 5.19 shows the qualitative results. More precisely, from left to right, the moved parameters, the hierarchical mesh, the hierarchical reconstructed geometry, and the scaled point-wise error distributions for the three considered fitting schemes, adaptive A-PDM (top), A-HDM (middle), and J-PDM (bottom).

We can notice that A-PDM and A-HDM show very similar approximation behaviour in terms of MSE. This can be motivated by the choice of the initial parameterization: BIDGCN computes a parameterization that minimizes the PDM error term, see Section 4.3.3, and the optimality is maintained also for the adaptive A-PDM scheme, characterized by the minimization of the same error term. On the other hand, the MAX error stagnates at $\sim 2.00e-2$ for adaptive A-PDM, whereas it keeps decreasing for adaptive A-HDM until $8.35e-3$, achieved with 2249 and 2197 DOFs, respectively. Finally, adaptive J-PDM exhibits a better approximation capacity both for MSE and MAX if compared with the adaptive alternating methods. As concerns the MAX, adaptive A-HDM and J-PDM converge to very similar values. More precisely, J-PDM registers a MAX error of $7.96e-3$, obtained with 2116 DOFs. However, the quality of the reconstructed geometry is higher for J-PDM, as can be observed in Figure 5.19, especially in the area around the eyes.

5.4 INDUSTRIAL APPLICATIONS WITH ADAPTIVE A-PDM AND A-QI

In this Section, we illustrate the performance of the proposed fitting methods within an industrial environment. More precisely, we consider the reverse engineering problem of reconstructing highly accurate CAD models of aircraft engine components. This reverse engineering process consists of three *major* steps, i. e.

1. data acquisition;

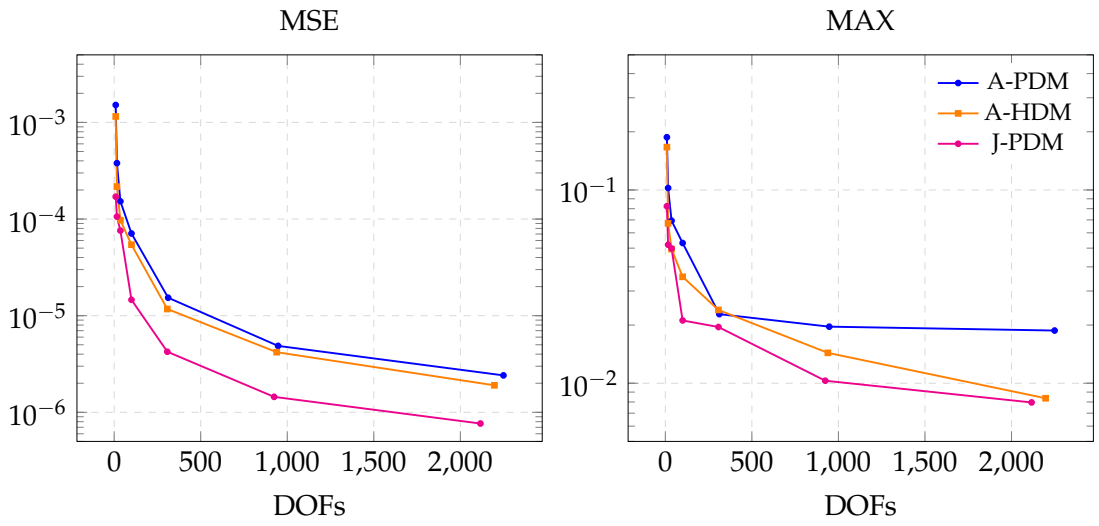


Figure 5.18: Adaptive spline approximation for the Nefertiti point cloud consisting of 9000 scattered data points with initial BIDGCN parameterization. Comparison of MSE (left) and MAX (right) between adaptive A-PDM (blue), A-HDM (orange) and J-PDM (magenta).

2. point processing;
3. surface fitting;
4. geometric model post-processing.

More specifically, step 1 consists of the collection of the data from a physical part through scanning, e. g., white light scanners, laser scanners, charge-coupled device (CCD) sensors, or coordinate-measuring machines (CMM). Step 2 concerns more *specific* procedures, such as the construction of a suitable triangulation, trimming, feature extractions, and data segmentation, among others. In particular, these routines are needed to recover intrinsic features of the geometric part, e. g., holes, slots, or curves. In step 3, dynamic fitting algorithms are used for surface generation. That is the case of our examples, where the considered data, due to the acquisition process, are scattered and affected by noise, yet the reconstructed geometric models are required to be compact and smooth while simultaneously capturing key geometric features of the engine parts. Finally, step 4 consists of the post-processing of the reconstructed surface models to shape and assemble the final CAD model. For instance, trimming routines can be performed to introduce holes, see e. g., [133], whereas multi-patch representations can be exploited to merge several components of the CAD model, see

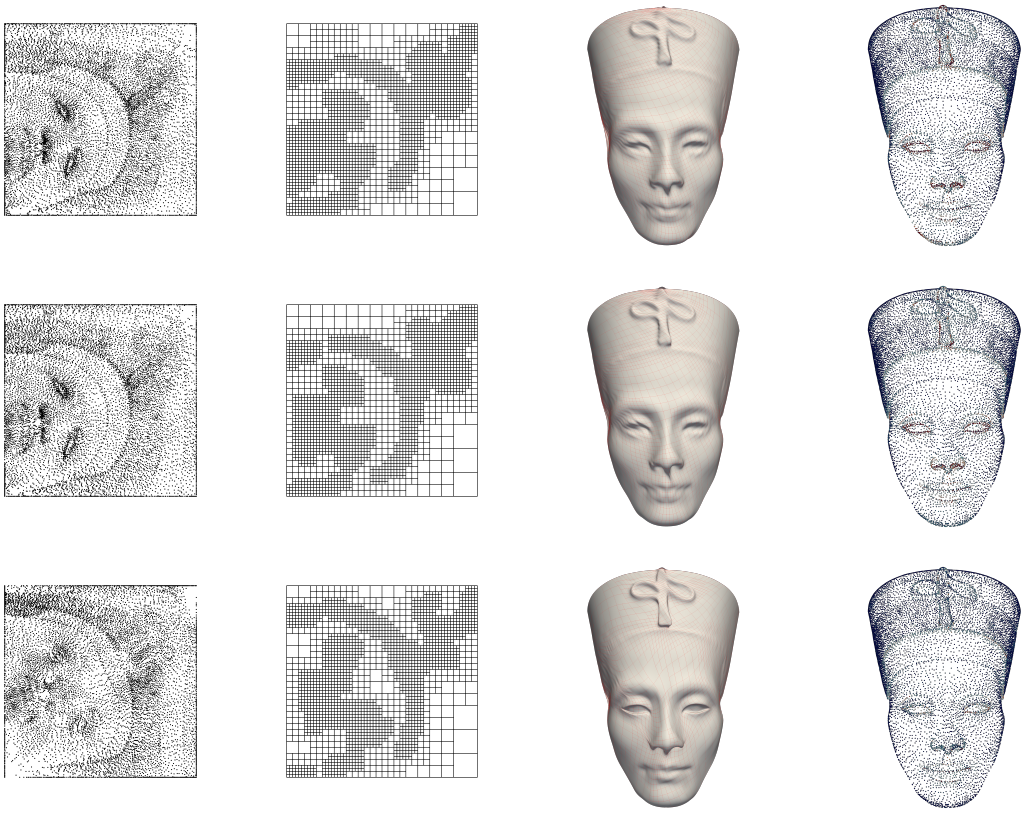


Figure 5.19: Adaptive spline approximation for the Nefertiti point cloud consisting of 9000 scattered data points with initial BIDGCN parameterization. Comparison of adaptive A-PDM (top line), A-HDM (central line), and J-PDM (bottom line). From left to right: the final (moved) parameters, the final hierarchical mesh, the final approximating THB-spline geometry ad the scaled point wise error.

e. g., [176]. Subsequently, the virtual models obtained are used for redesign operations, further manufacturing, or numerical simulations.

Throughout this section, we use two fitting methods, namely the global adaptive PDM approximation with THB-splines and the QI scheme using local spline approximation, both described in Chapter 3, to assess the benefits of the moving parameterization approach. Hence, we compare adaptive PDM and QI with adaptive A-PDM and A-QI on scanned data provided by MTU Aero Engines.

The algorithms related to the adaptive fitting schemes with THB-splines have been implemented in C++ with the open-source G+Smo library [94, 129]. The

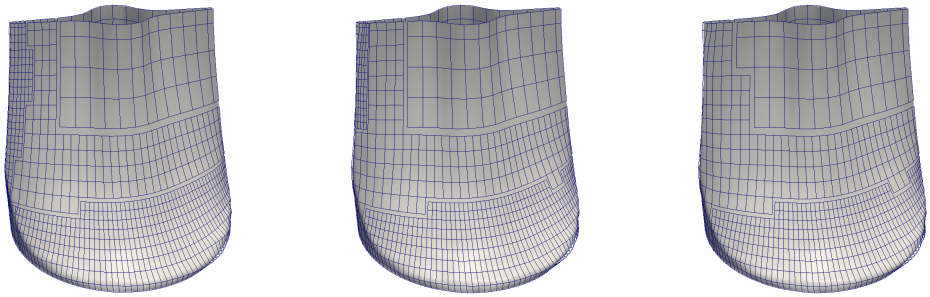


Figure 5.20: Adaptive THB-spline QI with moving parameters (adaptive A-QI) in Section 5.4.1. From left to right: models computed with 0, 1, and 2 PC steps.

developed code for the proposed algorithms has been integrated and will be available in the next releases.

5.4.1 *Tensile with moving parameterization*

In this example, we revisit the second configuration of Section 3.3.1, where the point cloud of a tensile part is initially approximated by a bi-quadratic spline on a tensor-product basis defined on a 4×16 mesh, with tolerance $\epsilon = 5\text{e} - 5\text{m}$. We compare the performance of the QI scheme without PC and with one or more rounds of PC after the adaptive refinement step (adaptive A-QI). The LBFGS optimizer with minimum step-size $s = 1\text{e} - 9$ is used for the foot-point projection. In particular, the resulting geometries when applying 0, 1 or 2 rounds of PC after each refinement step are shown in Figure 5.20. We can observe that all the fitting surfaces are of good quality. Nevertheless, in this case suitably embedding the parameter correction routine in the adaptive loop led to gaining the same precision in terms of MAX and MSE errors, while using fewer degrees of freedom to reconstruct the final geometric model, as summarized in the first three lines of Table 5.2. In particular, applying 3 steps of PC after each hierarchical mesh refinement produces an approximation with 15.20% fewer DOFs with respect to the one achieved with the standard adaptive QI scheme, while simultaneously reducing the MAX error by 9.89% and the MSE by 17.20%. Note that additional rounds (from 4 to 6 in Table 5.2) of parameter correction, yield diminishing improvements, since the DOFs are still reduced, but there is no gain in terms of MAX and MSE. This suggests that from 1 to 3 PC steps are in general a proper choice when including the parameter correction within A-QI.

method	%pts $< \epsilon$	MAX (m)	MSE (m^2)	DOFs	
QI	99.36	$8.11e-5$	$1.42e-10$	1960	
A-QI, 1 PC	99.33	$8.17e-5$	$1.41e-10$	1834	6.43%
A-QI, 2 PC	99.40	$7.97e-5$	$1.25e-10$	1718	12.35%
A-QI, 3 PC	99.46	$7.31e-5$	$1.17e-10$	1662	15.20%
A-QI, 4 PC	99.15	$8.66e-5$	$1.22e-10$	1659	15.36%
A-QI, 5 PC	99.02	$8.52e-5$	$1.23e-10$	1621	17.30%
A-QI, 6 PC	99.11	$9.19e-5$	$1.15e-10$	1621	17.30%

Table 5.2: Analysis in terms of points within the prescribed tolerance (%pts $< \epsilon$), MAX or MSE, and final number of DOFs for the adaptive THB-spline QI scheme in Section 5.4.1. The percentages of the DOFs reduction with respect to QI without PC are also reported.

5.4.2 *Blade with moving parameterization*

In this example, we revisit Section 3.3.2. However, in view of the considerations of Section 5.2.1, the initial tensor-product basis of bi-degree $\boldsymbol{d} = (3,3)$ is now more refined, i. e. we start with a 8×8 tensor product mesh. The results are still comparable, as the lowest level basis is entirely refined in the configuration considered in Section 3.3.2. We then fit the set of 27191 measured data points from a blade geometry that has a length of about $5e-2$m. We compare adaptive A-PDM and A-QI, by always considering as smoothing coefficient $\lambda = 1e-8$. In both cases, we stop when at least 95% of data points are within the tolerance $\epsilon = 2e-5$m.

Figure 5.21 shows the final THB-spline models obtained with adaptive A-PDM (top) and A-QI (bottom) when 0 (PDM, QI), 1, or 2 PC steps are considered in the adaptive THB-spline fitting schemes. In particular, we can observe that the A-QI geometries are characterized by a smaller amount of oscillations along the sharp features. To have more insights into this phenomenon, we present the reflection lines on the different THB-spline models in Figure 5.22. By analyzing the top and bottom row of this figure from left to right, we see that the PC improves the surface quality by reducing the oscillations near the top left corner. Moreover, by comparing the A-PDM (top) and A-QI (bottom) results in each column, we can see that the oscillations just under the feature in the top right area of the blade are reduced in the THB-spline models obtained with A-QI scheme.

The quantitative analysis for this example is detailed in Table 5.3. In particular, both for A-PDM and A-QI schemes, we report the final percentage of points within

method	%pts $< \epsilon$	MAX (m)	MSE (m^2)	DOFs	
PDM	99.99	$2.12e-5$	$6.32e-12$	8164	
A-PDM, 1 PC	99.19	$4.04e-5$	$2.48e-11$	2314	71.66%
A-PDM, 2 PC	99.65	$2.90e-5$	$2.37e-11$	2212	72.91%
QI	99.98	$2.74e-5$	$1.00e-11$	8278	
A-QI, 1 PC	97.42	$8.86e-5$	$4.71e-11$	2554	69.15%
A-QI, 2 PC	95.97	$5.71e-5$	$5.93e-11$	2539	69.33%

Table 5.3: Analysis in terms of points within the tolerance (%pts $< \epsilon$), MAX or MSE, and number of DOFs for the adaptive THB-spline A-PDM and A-QI schemes in Section 5.4.2. The percentages of the DOFs reduction with respect to A-PDM and A-QI without PC are also reported.

the tolerance, the MAX and MSE errors, as well as the number of DOFs, when 0 (PDM, QI), 1, or 2 parameter corrections are considered. Note that the stopping criterion (95% pts $< 2e-5m$) is met in all cases, with a significant reduction ($\sim$ 70%) of DOFs both for A-PDM and A-QI schemes when 1 or 2 PC steps are applied. This is due to the fact that the prescribed precision is achieved two iterations earlier, preventing the introduction of two additional hierarchical levels. This can be explicitly seen in Figure 5.23 for A-PDM (left) and A-QI (right), where the number of points below the prescribed tolerance versus the amount of DOFs at each adaptive iteration is reported. In this example, either one or two PC steps, after each refinement procedure, are a good choice, since the results are very similar, iteration by iteration, both for A-PDM and A-QI.

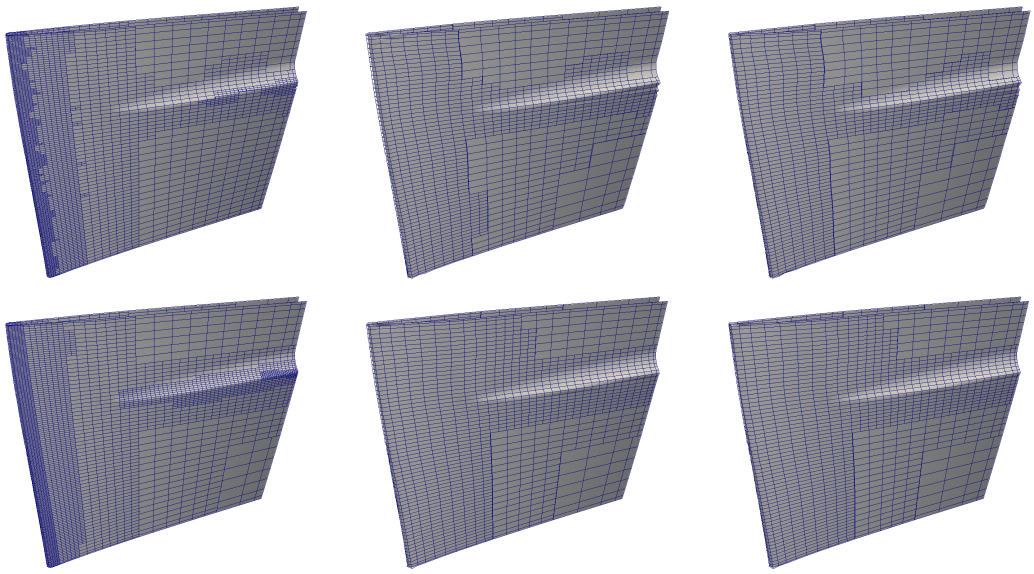


Figure 5.21: THB-spline models with control nets in Example 5.4.2. Top row, from left to right: PDM, A-PDM with 1, and 2 PC steps; bottom row: from left to right: QI, A-QI with 1, and 2 PC steps.

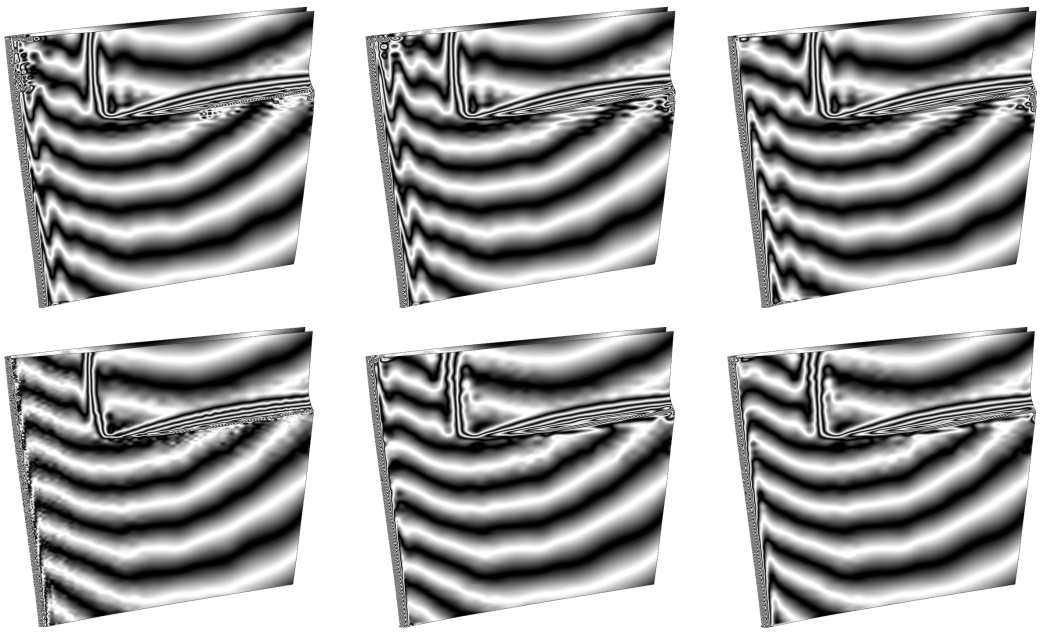


Figure 5.22: THB-spline models with reflection lines in Section 5.4.2. Top row, from left to right: PDM, A-PDM with 1, and 2 PC steps; bottom row: from left to right: QI, A-QI with 1, and 2 PC steps.

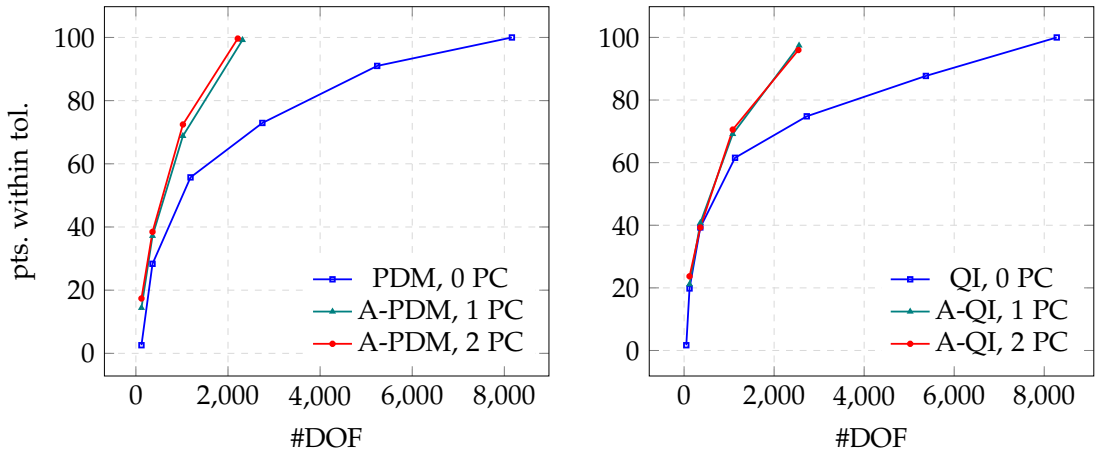


Figure 5.23: Percentage of points within the prescribed tolerance during the THB-spline PDM and A-PDM schemes in Section 5.4.2 (left) and during the THB-spline QI and A-QI schemes in Section 5.4.2 (right).

6

CONCLUSION AND FUTURE DEVELOPMENT

In this Thesis, we presented novel (adaptive) fitting methods with THB-splines for the (re-)construction of highly accurate CAD models from input point clouds. We combined CAGD approaches with DL technology to produce robust, automatic, and efficient fitting schemes.

In Chapter 2, we collected a selection of preliminaries notions on B-splines, THB-splines and NNs, with special focus on CNNs and GCNs.

In Chapter 3, we reviewed interpolation and least squares approximation schemes and provided a new general formulation for reweighted least squares as a convex combination of certain interpolants. Furthermore, we exploited the weights for spline fitting problems, also in the case of adaptive THB-spline constructions. We proposed a strategy to automatically update the weights within the fitting scheme, either to emphasize data marked as sharp features or to smoothen data marked as corrupted [67]. The two-stage hierarchical QI scheme for the approximation of scattered dataset using THB-splines [17, 19] was then revisited. We modified the first stage of the scheme by introducing local B-spline approximations to handle distribution with varying density of points. This choice improved the performance of the existing scheme by increasing the accuracy of the model and simultaneously reducing the computational costs. The advantages of this choice was proven in the numerical examples, where the reconstruction of industrial geometries was addressed [16].

In Chapter 4, we proposed novel data driven models both for gridded and scattered point cloud parameterization. These models were characterized by suitable NNs architectures based on convolutional operators, defined on the considered domain. The PARCNN model [41] was based on a *pure* CNN architecture, i. e. consisting only of convolutional blocks. This choice was made to take advantage of the locality of convolutional operators in order to be able to support variable input sizes without any additional effort and/or pre- or post-processing of the data. To overcome the limitation of CNNs to process only data with a grid-like topology, we then considered GCNs to address the parameterization learning problem of *scattered* data. PARGCN [71] processed data with a *graph* structure, corresponding to the radius neighbours graph of the input scattered point cloud. Subsequently, we devised BIDGCN [70], a new GCN architecture for the parameterization of scattered data that takes into account

boundary conditions in addition to the standard vertex features of the discrete surface.

All the proposed methods were agnostic to the size of the input point cloud, were robust to noise, and generalized to point clouds different from the ones used during the training phase. They outperformed both closed form, heuristic and data-driven parameterization choices and produced high-quality parameterizations for (TH)B-spline reconstruction schemes. In addition, BIDGCN, once trained, was computationally more efficient than the classical meshless parameterization methods.

In Chapter 5, we introduced novel adaptive fitting schemes with moving parameterization and THB-splines, based on the optimization of different error metrics [68, 69]. The first strategy to move the parameters, consisted of enriching the adaptive approximating loop with the PC routine [90]. As concerns the control points, we proposed different diverse update rules, which bring to the development of adaptive A-PDM, A-TDM, and A-HDM. In addition, we exploited the introduction of PC within the adaptive hierarchical QI scheme presented in Chapter 3. The second strategy to move the parameters consisted of addressing the parameterization problem together with the computation of the control points in the first step of the adaptive loop. This could be achieved by solving a non-linear joint optimization problem, J-PDM, which simultaneously computed the optimal parameter sites for the input point cloud and the optimal control points for the approximating surface. With this method, we avoided the need of solving a linear system of equations and performing PC at every adaptive iteration.

Our study revealed that, independently from the chosen strategy, addressing moving the parameterization within the adaptive loop could improve the fitting results while also reducing the number of degrees of freedom required to achieve a certain accuracy. This technique could lead to earlier termination of the adaptive process, thus providing more compact models with less refinement levels, being at the same time more accurate.

As concerns the parameterization of scattered point clouds, the methods in this Thesis assume the data to be already partitioned between interior and boundary points. In order to fully automate the point cloud reconstruction, a future research direction consists in developing new data-driven techniques to address the point cloud boundary detection problem, see e. g., [86, 126]. It is also of interest to develop new data-driven parameterization methods that learn quasi-conformal surface parameterization and reduce possible geometric distortion [30]. Finally, also domain parameterization could be enhanced by the development of suitable learning techniques, see e. g., [188].

As concerns fitting schemes, it is of interest to extend the methods presented in this Thesis to different adaptive spline functions e. g., [18, 97, 120]. Moreover, to design a full CAD model, multi-patch constructions need to be addressed. Consequently, it is of particular interest to develop multi-patch fitting schemes, see e. g., [132].

BIBLIOGRAPHY

[1] C. C. Aggarwal. *Neural networks and deep learning*. Springer, 2018.

[2] C. Balta, S. Öztürk, M. Kuncan, and I. Kandilli. «Dynamic Centripetal Parameterization Method for B-Spline Curve Interpolation.» In: *IEEE Access* 8 (2020), pp. 589–598.

[3] A. Beck. «On the Convergence of Alternating Minimization for Convex Programming with Applications to Iteratively Reweighted Least Squares and Decomposition Schemes.» In: *SIAM Journal on Optimization* 25.1 (2015), pp. 185–209.

[4] J. L. Bentley, D. F. Stanat, and E. H. Williams. «The complexity of finding fixed-radius near neighbors.» In: *Information Processing Letters* 6 (6 Dec. 1977), pp. 209–212.

[5] J. Berner, P. Grohs, G. Kutyniok, and P. Petersen. «The Modern Mathematics of Deep Learning.» In: *Mathematical Aspects of Deep Learning*. Ed. by P. Grohs and G. Kutyniok. Cambridge University Press, 2022, 1–111.

[6] S. Berrone, F. Della Santa, A. Mastropietro, S. Pieraccini, and F. Vaccarino. «Graph-Informed Neural Networks for Regressions on Graph-Structured Data.» In: *Mathematics* 10.5 (2022), p. 786.

[7] C. M. Bishop. *Patter Recognition and Machine Learning*. 1st ed. Information Science and Statistics. Springer New York, 2006.

[8] Å. Björck. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, 1996.

[9] A. Blake and M. Isard. *Active Contours*. Springer London, 1998.

[10] P. Bo, R. Ling, and W. Wang. «A revisit to fitting parametric surfaces to point clouds.» In: *Computers & Graphics* 36.5 (2012). Shape Modeling International (SMI) Conference 2012, pp. 534–540.

[11] F. L. Bookstein. *The Measurement of Biological Shape and Shape Change*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1978.

[12] C. de Boor. *A Practical Guide to Splines*. Springer New York, NY, 1978.

[13] C. de Boor. «On calculating with B-splines.» In: *Journal of Approximation Theory* 6.1 (1972), pp. 50–62.

[14] C. de Boor, K. Höllig, and S. Riemenschneider. *Box Splines*. Springer New York, NY, 1993.

[15] L. Bos, A. Sommariva, and M. Vianello. «Least-squares polynomial approximation on weakly admissible meshes: Disk and triangle.» In: *Journal of Computational and Applied Mathematics* 235.3 (2010), pp. 660–668.

[16] C. Bracco, C. Giannelli, D. Großmann, S. Imperatore, D. Mokriš, and A. Sestini. «THB-Spline Approximations for Turbine Blade Design with Local B-Spline Approximations.» In: *Mathematical and Computational Methods for Modelling, Approximation and Simulation*. Ed. by D. Barrera, S. Remogna, and D. Sbibih. SEMA SIMAI. https://doi.org/10.1007/978-3-030-94339-4_3. Cham: Springer International Publishing, 2022, pp. 63–82.

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

[18] C. Bracco, C. Giannelli, F. Patrizi, and A. Sestini. «Local spline refinement driven by fault jump estimates for scattered data approximation.» In: *arXiv preprint arXiv:2311.09442* (2023).

[19] C. Bracco, C. Giannelli, and A. Sestini. «Adaptive scattered data fitting by extension of local approximations to hierarchical splines.» In: *Computer Aided Geometric Design* 52 (2017), pp. 90–105.

[20] J. G. Broida and S. G. Williamson. *A Comprehensive Introduction to Linear Algebra*. Advanced book program. Addison-Wesley, 1989.

[21] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. «Geometric Deep Learning: Going beyond Euclidean data.» In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42.

[22] L. Brugnano, D. Giordano, F. Iavernaro, and G. Rubino. «An entropy-based approach for a robust least squares spline approximation.» In: *Journal of Computational and Applied Mathematics* 443 (2024), p. 115773.

[23] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. «Spectral networks and locally connected networks on graphs.» In: *International Conference on Learning Representations*. 2014.

[24] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. «A Limited Memory Algorithm for Bound Constrained Optimization.» In: *SIAM Journal on Scientific Computing* 16.5 (1995), pp. 1190–1208.

[25] R. Cavoretto, A. De Rossi, F. Dell'Accio, and F. Di Tommaso. «Fast computation of triangular Shepard interpolants.» In: *Journal of Computational and Applied Mathematics* 354 (2019), pp. 457–470.

[26] R. Cavoretto, A. De Rossi, F. Dell'Accio, and F. Di Tommaso. «An Efficient Trivariate Algorithm for Tetrahedral Shepard Interpolation.» In: *Journal of Scientific Computing* 82.3 (2020).

[27] Q. Chang, C. Deng, and M. S. Floater. «An interpolatory view of polynomial least squares approximation.» In: *Journal of Approximation Theory* 252 (2020), p. 105360.

[28] O. Chapelle, B. Schölkopf, and A. Zien, eds. *Semi-supervised learning*. Adaptive Computation and Machine Learning. The MIT Press, 2006.

[29] K.-S. Cheng, W. Wang, H. Qin, K.-Y. Wong, H. Yang, and Y. Liu. «Fitting subdivision surfaces to unorganized point data using SDM.» In: *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.* 2004, pp. 16–24.

[30] G. P. T. Choi and L. M. Lui. «Recent Developments of Surface Parameterization Methods Using Quasi-conformal Geometry.» In: *Handbook of Mathematical Models and Algorithms in Computer Vision and Imaging: Mathematical Imaging and Vision*. Ed. by K. Chen, C.-B. Schönlieb, X.-C. Tai, and L. Younes. Cham: Springer International Publishing, 2023, pp. 1483–1523.

[31] L. Choong-Gyoo. «A universal parametrization in B-spline curve and surface interpolation.» In: *Computer Aided Geometric Design* 16.5 (1999), pp. 407–422.

[32] F. R. K. Chung. *Spectral graph theory*. Vol. 92. American Mathematical Society, 1997.

[33] W. S. Cleveland. «Robust Locally Weighted Regression and Smoothing Scatterplots.» In: *Journal of the American Statistical Association* 74.368 (1979), pp. 829–836.

[34] W. S. Cleveland. «LOWESS: A Program for Smoothing Scatterplots by Robust Locally Weighted Regression.» In: *The American Statistician* 35.1 (1981), pp. 54–54.

[35] M. G. Cox. «The Numerical Evaluation of B-Splines.» In: *IMA Journal of Applied Mathematics* 10.2 (1972), pp. 134–149.

[36] P. Craven and G. Wahba. «Smoothing noisy data with spline functions.» In: *Numerische Mathematik* 31.4 (1978), pp. 377–403.

[37] H. B. Curry and I. J. Schoenberg. «On Pólya frequency functions IV: the fundamental spline functions and their limits.» In: *Journal d'Analyse Mathématique* 17 (1966), pp. 71–107.

[38] H. B. Curry and I. J. Schoenberg. «On spline distributions and their limits-the Pólya distribution functions.» In: *Bulletin of the American Mathematical Society*. Vol. 53. 11. 1947, p. 1114.

[39] P. J. Davis. *Interpolation and Approximation*. Dover Books on Mathematics. Dover Publications, 1975.

[40] S. De Marchi, F. Dell'Accio, and M. Mazza. «On the constrained mock-Chebyshev least-squares.» In: *Journal of Computational and Applied Mathematics* 280 (2015), pp. 94–109.

[41] M. De Vita, C. Giannelli, S. Imperatore, and A. Mantzaflaris. «Parameterization learning with convolutional neural networks for gridded data fitting.» In: *Advances in Information and Communication*. Ed. by K. Arai. Lectures Notes in Networks and Systems. https://doi.org/10.1007/978-981-99-7886-1_32. Cham: Springer Nature Switzerland, 2024, pp. 393–412.

[42] M. Defferrard, X. Bresson, and P. Vandergheynst. «Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering.» In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc., 2016.

[43] B. Delaunay. «Sur la sphère vide. A la mémoire de Georges Voronoï.» In: *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na* 6 (1934), pp. 793–800.

[44] F. Dell'Accio, F. Di Tommaso, O. Nouisser, and N. Siar. «Solving Poisson equation with Dirichlet conditions through multinode Shepard operators.» In: *Computers & Mathematics with Applications* 98 (2021), pp. 254–260.

[45] F. Dell'Accio, F. Di Tommaso, and F. Nudo. «Generalizations of the constrained mock-Chebyshev least squares in two variables: Tensor product vs total degree polynomial interpolation.» In: *Applied Mathematics Letters* 125 (2022), p. 107732.

[46] J. Deng, F. Chen, X. Li, C. Hu, W. Tong, Z. Yang, and Y. Feng. «Polynomial splines over hierarchical T-meshes.» In: *Graphical Models* 70.4 (2008), pp. 76–86.

[47] J. Devlin, M. Chang, K. Lee, and K. Toutanova. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.» In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by J. Burstein, C. Doran, and T. Solorio. Association for Computational Linguistics, 2019, pp. 4171–4186.

[48] T. Dokken, T. Lyche, and K. F. Pettersen. «Polynomial splines over locally refined box-partitions.» In: *Computer Aided Geometric Design* 30.3 (2013), pp. 331–356.

[49] S. Dreyfus. «The numerical solution of variational problems.» In: *Journal of Mathematical Analysis and Applications* 5.1 (1962), pp. 30–45.

[50] N. Engleitner and B. Jüttler. «Patchwork B-spline refinement.» In: *Computer-Aided Design* 90 (2017), pp. 168–179.

[51] L. Fan, D. Ji, and P. Lin. «Arbitrary surface data patching method based on geometric convolutional neural network.» In: *Neural Computing and Applications* 35.12 (2023), pp. 8763–8774.

[52] J.-J. Fang and C.-L. Hung. «An improved parameterization method for B-spline curve and surface interpolation.» In: *Computer-Aided Design* 45.6 (2013), pp. 1005–1028.

[53] S. Farahmand and G. B. Giannakis. «Robust RLS in the presence of correlated noise using outlier sparsity.» In: *IEEE transactions on signal processing* 60.6 (2012), pp. 3308–3313.

[54] S. Farahmand, G. B. Giannakis, and D. Angelosante. «Doubly robust smoothing of dynamical processes via outlier sparsity constraints.» In: *IEEE Transactions on Signal Processing* 59.10 (2011), pp. 4529–4543.

[55] G. Farin. *Curves and surfaces for CAGD: a practical guide*. Ed. by G. Farin. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, 2002.

[56] G. Farin, J. Hoschek, and M.-S. Kim, eds. *Handbook of Computer Aided Geometric Design*. North-Holland, 2002.

[57] M. Fey and J. E. Lenssen. «Fast Graph Representation Learning with PyTorch Geometric.» In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.

[58] M. S. Floater. «Parametrization and smooth approximation of surface triangulations.» In: *Computer aided geometric design* 14.3 (1997), pp. 231–250.

[59] M. S. Floater. «Mean value coordinates.» In: *Computer aided geometric design* 20.1 (2003), pp. 19–27.

[60] M. S. Floater. «On the deviation of a parametric cubic spline interpolant from its data polygon.» In: *Computer Aided Geometric Design* 25.3 (2008), pp. 148–156.

[61] M. S. Floater and K. Hormann. «Surface parameterization: a tutorial and survey.» In: *Advances in multiresolution for geometric modelling* (2005), pp. 157–186.

[62] M. S. Floater and M. Reimers. «Meshless parameterization and surface reconstruction.» In: *Computer Aided Geometric Design* 18.2 (2001), pp. 77–92.

[63] M. S. Floater and T. Surazhsky. «Parameterization for Curve Interpolation.» In: *Topics in Multivariate Approximation and Interpolation*. Ed. by K. Jetter, M. D. Buhmann, W. Haussmann, R. Schaback, and J. Stöckler. Vol. 12. Studies in Computational Mathematics. Elsevier, 2006, pp. 39–54.

[64] D. R. Forsey and R. H. Bartels. «Hierarchical B-Spline Refinement.» In: *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '88. New York, NY, USA: Association for Computing Machinery, 1988, 205–212.

[65] D. R. Forsey and R. H. Bartels. «Surface Fitting with Hierarchical Splines.» In: *ACM Trans. Graph.* 14.2 (1995), pp. 134–161.

[66] J. H. Friedman. «Multivariate Adaptive Regression Splines.» In: *The Annals of Statistics* 19.1 (1991), pp. 1 –67.

[67] C. Giannelli, S. Imperatore, L. M. Kreusser, E. Loayza-Romero, F. Mohammadi, and N. Villamizar. *A general formulation of reweighted least squares fitting*. Accepted for publication in Mathematics and Computers in Simulation. arXiv preprint arXiv:2404.03742. 2024.

[68] C. Giannelli, S. Imperatore, A. Mantzaflaris, and D. Mokriš. «Leveraging moving parameterization and adaptive THB-splines for CAD surface reconstruction of aircraft engine components.» In: *Smart Tools and Applications in Graphics - Eurographics Italian Chapter Conference*. Ed. by F. Banterle, G. Caggianese, N. Capece, U. Erra, K. Lupinetti, and G. Manfredi. https://doi.org/10.2312/stag.20231301. The Eurographics Association, 2023.

[69] C. Giannelli, S. Imperatore, A. Mantzaflaris, and D. Mokriš. *Efficient alternating and joint distance minimization methods for adaptive surface fitting with THB-splines*. Submitted to Graphical Models. Invited for submission to the "Special Issue STAG 2023: Smart Tool and Applications in Graphics". 2024.

[70] C. Giannelli, S. Imperatore, A. Mantzaflaris, and F. Scholz. *BIDGCN: boundary informed dynamic graph convolutional network for adaptive spline fitting of scattered data*. Under second revision for Neural Computing and Applications. https://inria.hal.science/hal-04313629v1. 2023.

[71] C. Giannelli, S. Imperatore, A. Mantzaflaris, and F. Scholz. «Learning Meshless Parameterization with Graph Convolutional Neural Networks.» In: *Intelligent Sustainable Systems*. Ed. by A. K. Nagar, D. S. Jat, D. K. Mishra, and A. Joshi. https://doi.org/10.1007/978-981-99-7886-1_32. Singapore: Springer Nature Singapore, 2024, pp. 375–387.

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

[73] C. Giannelli and H. S. Jüttler. «Strongly stable bases for adaptively refined multilevel spline spaces.» In: *Advances in Computational Mathematics* 40.2 (2014), pp. 459–490.

[74] C. Giannelli, B. Jüttler, and H. Speleers. «THB-splines: The truncated basis for hierarchical splines.» In: *Computer Aided Geometric Design* 29.7 (2012). Geometric Modeling and Processing 2012, pp. 485–498.

[75] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. «Neural message passing for quantum chemistry.» In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[76] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[77] M. Gori, G. Monfardini, and F. Scarselli. «A new model for learning in graph domains.» In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. 2005, pp. 729–734.

[78] J. L. Gross, J. Yellen, and M. Anderson. *Graph theory and its applications*. Chapman and Hall/CRC, 2018, p. 265.

[79] C. Gu. «Cross-Validating Non-Gaussian Data.» In: *Journal of Computational and Graphical Statistics* 1.2 (1992), pp. 169–179.

[80] C. Gu. *Smoothing spline ANOVA models*. Springer, 2002.

[81] L. J. Guibas, D. E. Knuth, and M. Sharir. «Randomized incremental construction of Delaunay and Voronoi diagrams.» In: *Algorithmica* 7.1 (1992), pp. 381–413.

[82] A. Gupta, O. P. Kharbanda, V. Sardana, R. Balachandran, and H. K. Sardana. «A knowledge-based algorithm for automatic detection of cephalometric landmarks on CBCT images.» In: *International journal of computer assisted radiology and surgery* 10 (2015), pp. 1737–1752.

[83] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or. «Meshcnn: a network with an edge.» In: *ACM Transactions on Graphics (TOG)* 38.4 (2019), pp. 1–12.

[84] F. Harary and R. Z. Norman. «Some properties of line digraphs.» In: *Rendiconti del Circolo Matematico di Palermo* 9.2 (1960), pp. 161–168.

[85] K. He, X. Zhang, S. Ren, and J. Sun. «Deep Residual Learning for Image Recognition.» In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2016, pp. 770–778.

[86] C.-E. Himeur, T. Lejemble, T. Pellegrini, M. Paulin, L. Barthe, and N. Mellado. «Pcednet: A lightweight neural network for fast and interactive edge detection in 3d point clouds.» In: *ACM Transactions on Graphics* 41.1 (2021), pp. 1–21.

[87] S. Hochreiter and J. Schmidhuber. «Long Short-Term Memory.» In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[88] P. W. Holland and R. E. Welsch. «Robust regression using iteratively reweighted least-squares.» In: *Communications in Statistics - Theory and Methods* 6.9 (1977), pp. 813–827.

[89] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. «Piecewise Smooth Surface Reconstruction.» In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '94. Association for Computing Machinery, 1994, pp. 295–302.

[90] J. Hoschek. «Intrinsic parametrization for approximation.» In: *Computer Aided Geometric Design* 5.1 (1988), pp. 27–31.

[91] J. Hoschek, D. Lasser, and L. L. Schumaker. *Fundamentals of Computer Aided Geometric Design*. USA: A. K. Peters, Ltd., 1993.

[92] T. M. Inc. *Curve Fitting Toolbox Version 3.8 (R2022b)*. 2024.

[93] L. Jiang, R. H. Byrd, E. Eskow, and R. Schnabel. *A Preconditioned L-BFGS Algorithm with Application to Molecular Energy Minimization*. Tech. rep. Colorado University at Boulder Department of Computer Science, 2004.

[94] B. Jüttler, U. Langer, A. Mantzaflaris, S. E. Moore, and W. Zulehner. «Geometry + Simulation Modules: Implementing Isogeometric Analysis.» In: *PAMM* 14.1 (2014), pp. 961–962.

[95] H. Kang, F. Chen, and J. Deng. «Modified T-splines.» In: *Computer Aided Geometric Design* 30.9 (2013), pp. 827–843.

[96] H. Kang, F. Chen, and J. Deng. «Hierarchical Box Splines.» In: *2015 14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)*. 2015, pp. 73–80.

[97] H. Kang, Z. Yong, and X. Li. «Quasi-interpolation for analysis-suitable T-splines.» In: *Computer Aided Geometric Design* 98 (2022), p. 102147.

[98] H. J. Kelley. «Gradient theory of optimal flight paths.» In: *Ars Journal* 30.10 (1960), pp. 947–954.

[99] N. Ketkar and J. Moolayil. «Introduction to PyTorch.» In: *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch*. Apress, 2021, pp. 27–91.

[100] G. Kiss, C. Giannelli, and B. Jüttler. «Algorithms and data structures for truncated hierarchical B–splines.» In: *Mathematical Methods for Curves and Surfaces: 8th International Conference, MMCS 2012, Oslo, Norway, June 28–July 3, 2012, Revised Selected Papers 8*. Springer. 2014, pp. 304–323.

[101] G. Kiss, C. Giannelli, U. Zore, B. Jüttler, D. Großmann, and J. Barner. «Adaptive CAD model (re-) construction with THB-splines.» In: *Graphical models* 76.5 (2014), pp. 273–288.

[102] K. Ko and T. Sakkalis. «Orthogonal projection of points in CAD/CAM applications: an overview.» In: *Journal of Computational Design and Engineering* 1.2 (2014), pp. 116–127.

[103] R. Kraft. «Adaptive and Linearly Independent Multilevel B-Splines.» In: *Surface Fitting and Multiresolution Methods*. Ed. by A. Le Méhauté, C. Rabut, and L. L. Schumaker. Vanderbilt University Press, 1997, pp. 209–218.

[104] A. Krizhevsky, I. Sutskever, and G. E. Hinton. «ImageNet Classification with Deep Convolutional Neural Networks.» In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. C. J. Burges, L. Bottou, and K. Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012, pp. 1097–1105.

[105] M.-J. Lai and L. L. Schumaker. *Spline Functions on Triangulations*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2007.

[106] P. Laube, M. O. Franz, and G. Umlauf. «Deep Learning Parametrization for B-Spline Curve Approximation.» In: *2018 International Conference on 3D Vision (3DV)*. 2018, pp. 691–699.

[107] P. Laube, M. O. Franz, and G. Umlauf. «Learnt knot placement in B-spline curve approximation using support vector machines.» In: *Computer Aided Geometric Design* 62 (2018), pp. 104–116. ISSN: 0167-8396.

[108] Y. Le Cun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. «Handwritten digit recognition: applications of neural network chips and automatic learning.» In: *IEEE Communications Magazine* 27.11 (1989), pp. 41–46.

[109] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. «Backpropagation Applied to Handwritten Zip Code Recognition.» In: *Neural Computation* 1.4 (1989), pp. 541–551.

[110] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. «Gradient-based learning applied to document recognition.» In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[111] Y. LeCun, Y. Bengio, and G. Hinton. «Deep Learning.» In: *Nature* 521.7553 (2015), pp. 436–444.

[112] E. Lee. «Choosing nodes in parametric curve interpolation.» In: *Computer-Aided Design* 21.6 (1989), pp. 363–370.

[113] S. Lee, G. Wolberg, and S. Y. Shin. «Scattered data interpolation with multilevel B-splines.» In: *IEEE Transactions on Visualization and Computer Graphics* 3.3 (1997), pp. 228–244.

[114] D. Lenz, R. Yeh, V. Mahadevan, I. Grindeanu, and T. Peterka. «Customizable adaptive regularization techniques for B-spline modeling.» In: *Journal of Computational Science* 71 (2023).

[115] R. Levie, W. Huang, L. Bucci, M. M. Bronstein, and G. Kutyniok. «Transferability of Spectral Graph Convolutional Neural Networks.» In: *Journal of Machine Learning Research* 22.1 (2022), pp. 12462–12520.

[116] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein. «CayleyNets: Graph Convolutional Neural Networks With Complex Rational Spectral Filters.» In: *IEEE Transactions on Signal Processing* 67 (2017), pp. 97–109.

[117] X. Li and M. A. Scott. «Analysis-suitable T-splines: Characterization, refineability, and approximation.» In: *Mathematical Models and Methods in Applied Sciences* 24.06 (2014), pp. 1141–1164.

[118] X. Li and T. W. Sederberg. «S-splines: A simple surface solution for IGA and CAD.» In: *Computer Methods in Applied Mechanics and Engineering* 350 (2019), pp. 664–678.

[119] X. Li and J. Zhang. «AS++ T-splines: Linear independence and approximation.» In: *Computer Methods in Applied Mechanics and Engineering* 333 (2018), pp. 462–474.

[120] H. Lin and Z. Zhang. «An Efficient Method for Fitting Large Data Sets Using T-Splines.» In: *SIAM Journal on Scientific Computing* 35.6 (2013), A3052–A3068.

[121] D. C. Liu and J. Nocedal. «On the limited memory BFGS method for large scale optimization.» In: *Mathematical Programming* 45.1 (1989), pp. 503–528.

[122] Y. Liu, H. Pottmann, and W. Wang. «Constrained 3D shape reconstruction using a combination of surface fitting and registration.» In: *Computer-Aided Design* 38.6 (2006), pp. 572–583.

[123] Y. Liu and W. Wang. «A Revisit to Least Squares Orthogonal Distance Fitting of Parametric Curves and Surfaces.» In: *Advances in Geometric Modeling and Processing*. Ed. by F. Chen and B. Jüttler. Springer Berlin Heidelberg, 2008, pp. 384–397.

[124] Y. Liu. *HLBFGS: a hybrid L-BFGS optimization framework*. 2010.

[125] Z. Liu, H. Leung, L. Zhou, and H. P. Shum. «High quality compatible triangulations for 2D shape morphing.» In: *Proceedings of the 21st ACM Symposium on Virtual Reality Software and Technology*. 2015, pp. 85–94.

[126] M. Loizou, M. Averkiou, and E. Kalogerakis. «Learning part boundaries from 3d point clouds.» In: *Computer Graphics Forum*. Vol. 39. 5. Wiley Online Library. 2020, pp. 183–195.

[127] G. G. Lorentz. *Bernstein polynomials*. Mathematical Expositions 8. University of Toronto Press, 1953.

[128] S. Mallat. *A wavelet tour of signal processing*. Elsevier Science, 1999, p. 832.

[129] A. Mantzaflaris. «An Overview of Geometry Plus Simulation Modules.» In: *Mathematical Aspects of Computer and Information Sciences*. Ed. by D. Slamanig, E. Tsigaridas, and Z. Zafeirakopoulos. Springer, 2020, pp. 453–456.

[130] M. Marinov and L. Kobbelt. «Optimization methods for scattered data approximation with subdivision surfaces.» In: *Graphical Models* 67.5 (2005), pp. 452–473.

[131] S. Mariyam Hj. Shamsuddin and M. A. Ahmed. «A hybrid parameterization method for NURBS.» In: *Proceedings. International Conference on Computer Graphics, Imaging and Visualization, 2004. CGIV 2004.* 2004, pp. 15–20.

[132] M. Marsala, A. Mantzaflaris, and B. Mourrain. «G1 spline functions for point cloud fitting.» In: *Applied Mathematics and Computation* 460 (2024), p. 128279.

[133] B. Marussig and T. J. R. Hughes. «A Review of Trimming in Isogeometric Analysis: Challenges, Data Exchange and Simulation Aspects.» In: *Archives of Computational Methods in Engineering* 25.4 (2018), pp. 1059–1127.

[134] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon. «A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955.» In: *AI Magazine* 27.4 (2006), p. 12.

[135] W. S. McCulloch and W. Pitts. «A logical calculus of the ideas immanent in nervous activity.» In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[136] S. Merchel, B. Jüttler, and D. Mokriš. «Adaptive and local regularization for data fitting by tensor-product spline surfaces.» In: *Advances in Computational Mathematics* 49.4 (2023), p. 58.

[137] T. Mitchell. *Machine Leaning*. 1st ed. New York, USA: McGraw-Hill, 1997, p. 461.

[138] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. «Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs.» In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2017, pp. 5425–5434.

[139] X. Ning, F. Li, G. Tian, and Y. Wang. «An efficient outlier removal method for scattered point cloud data.» In: *PloS one* 13.8 (2018), e0201280.

[140] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer New York, 1999.

[141] S. Ohrhallinger, J. Peethambaran, A. D. Parakkat, T. K. Dey, and R. Muthuganapathy. «2D points curve reconstruction survey and benchmarks.» In: *Computer Graphics Forum* 40 (2021), Number 2.

[142] M. R. Osborne. *Finite Algorithms in Optimization and Data Analysis*. USA: John Wiley & Sons, Inc., 1985. ISBN: 0471905399.

[143] A. Paszke et al. «PyTorch: An Imperative Style, High-Performance Deep Learning Library.» In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2019.

[144] T. Pavlidis. «Curve Fitting with Conic Splines.» In: *ACM Trans. Graph.* 2.1 (1983), pp. 1–31.

[145] L. Piegl and W. Tiller. *The NURBS Book*. Monographs in Visual Communication 2. Springer Berlin, Heidelberg, 1997, p. 646.

[146] M. Plass and M. Stone. «Curve-Fitting with Piecewise Parametric Cubics.» In: *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '83. Association for Computing Machinery, 1983, pp. 229–239.

[147] H. Pottmann, S. Leopoldseder, and M. Hofer. «Approximation with active B-spline curves and surfaces.» In: *10th Pacific Conference on Computer Graphics and Applications, 2002. Proceedings.* 2002, pp. 8–25.

[148] H. Pottmann and M. Hofer. «Geometry of the Squared Distance Function to Curves and Surfaces.» In: *Visualization and Mathematics III*. Ed. by H.-C. Hege and K. Polthier. Springer Berlin Heidelberg, 2003, pp. 221–242.

[149] H. Pottmann and S. Leopoldseder. «A concept for parametric surface fitting which avoids the parametrization problem.» In: *Computer Aided Geometric Design* 20.6 (2003), pp. 343–362. ISSN: 0167-8396.

[150] R. Pytlak. *Conjugate gradient algorithms in nonconvex optimization*. Vol. 89. Nonconvex Optimization and Its Applications. Springer Berlin, Heidelberg, 2008.

[151] R. Pytlak and T. Tarnawski. «Preconditioned conjugate gradient algorithms for nonconvex problems.» In: *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*. Vol. 3. 2004, pp. 3191–3196.

[152] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. «Pointnet: Deep learning on point sets for 3d classification and segmentation.» In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[153] D. F. Rogers. *An Introduction to NURBS With Historical Perspective*. Ed. by D. F. Rogers. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, 2001.

[154] F. Rosenblatt. «The perceptron: A probabilistic model for information storage and organization in the brain.» In: *Psychological review* 65.6 (1958), p. 386.

[155] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. «Learning representations by back-propagating errors.» In: *Nature* 323.6088 (1986), pp. 533–536.

[156] C. Runge. «Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten.» In: *Zeitschrift für Mathematik und Physik* 46.224-243 (1901), p. 20.

[157] D. Saad, ed. *On-Line Learning in Neural Networks*. Publications of the Newton Institute. Cambridge University Press, 1999.

[158] A. L. Samuel. «Some Studies in Machine Learning Using the Game of Checkers.» In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229.

[159] E. Saux and M. Daniel. «An improved Hoschek intrinsic parameterization.» In: *Computer Aided Geometric Design* 20 (2003), pp. 513–521.

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

[161] T. W. Sederberg, D. L. Cardon, G. T. Finnigan, N. S. North, J. Zheng, and T. Lyche. «T-spline simplification and local refinement.» In: *ACM Transaction on Graphics* 23.3 (2004), 276–283.

[162] T. W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri. «T-splines and T-NURCCs.» In: *ACM Transaction on Graphics* 22.3 (2003), 477–484.

[163] N. Sharp, S. Attaiki, K. Crane, and M. Ovsjanikov. «DiffusionNet: Discretization agnostic learning on surfaces.» In: *ACM Transactions on Graphics (TOG)* 41.3 (2022), pp. 1–16.

[164] Siemens Software. *Parasolid*.

[165] H. Speleers and C. Manni. «Effortless quasi-interpolation in hierarchical spaces.» In: *Numerische Mathematik* 132.1 (2016), pp. 155–184.

[166] R. K. Srivastava, K. Greff, and J. Schmidhuber. «Training Very Deep Networks.» In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. Curran Associates, Inc., 2015.

[167] D. Stoller, S. Ewert, and S. Dixon. «Wave-u-net: A multi-scale neural network for end-to-end audio source separation.» In: *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, September 23-27, 2018*. Ed. by E. Gómez, X. Hu, E. Humphrey, and E. Benetos. 2018, pp. 334–340.

[168] T. Strutz. *Data Fitting and Uncertainty: A Practical Introduction to Weighted Least Squares and Beyond*. 2nd. Springer Vieweg, 2015.

[169] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. «On the importance of initialization and momentum in deep learning.» In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by S. Dasgupta and D. McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 2013, pp. 1139–1147.

[170] R. S. Sutton and A. G. Barto. *Reinforcment Learning*. 2nd ed. Adaptive Computation and Machine Learning. The MIT Press, 2018.

[171] D. C. Thomas, L. Engvall, S. K. Schmidt, K. Tew, and M. A. Scott. «U-splines: Splines over unstructured meshes.» In: *Computer Methods in Applied Mechanics and Engineering* 401 (2022), p. 115515.

[172] A. M. Turing. «I.—COMPUTING MACHINERY AND INTELLIGENCE.» In: *Mind* LIX.236 (1950), pp. 433–460.

[173] C. Valerio. *La matematica è politica*. Einaudi, 2020.

[174] V. N. Vapnik. *The Nature of Statistical Learning Theory*. 2nd ed. Information Science and Statistics. Springer New York, 1999.

[175] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. «Attention is All you Need.» In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett. Vol. 30. 2017, pp. 5998–6008.

[176] H. M. Verhelst, P. Weinmüller, A. Mantzaflaris, T. Takacs, and D. Toshniwal. «A comparison of smooth basis constructions for isogeometric analysis.» In: *Computer Methods in Applied Mechanics and Engineering* 419 (2024), p. 116659.

[177] A. Vuong, C. Giannelli, B. Jüttler, and B. Simeon. «A hierarchical approach to adaptive local refinement in isogeometric analysis.» In: *Computer Methods in Applied Mechanics and Engineering* 200.49-52 (2011), pp. 3554–3567.

[178] G. Wahba. *Spline models for observational data*. CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM, 1990.

[179] W. Wang, H. Pottmann, and Y. Liu. «Fitting B-Spline Curves to Point Clouds by Curvature-Based Squared Distance Minimization.» In: *ACM Trans. Graph.* 25.2 (2006), pp. 214–238.

[180] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. «Dynamic Graph CNN for Learning on Point Clouds.» In: *ACM Transactions on Graphics* 38.5 (2019).

[181] X. Wei, Y. Zhang, L. Liu, and T. J. Hughes. «Truncated T-splines: Fundamentals and methods.» In: *Computer Methods in Applied Mechanics and Engineering* 316 (2017), pp. 349–372.

[182] M. Welling and T. N. Kipf. «Semi-supervised classification with graph convolutional networks.» In: *International Conference on Learning Representations (ICLR)*. 2017.

[183] P. Wolfe. «Convergence Conditions for Ascent Methods.» In: *SIAM Review* 11.2 (1969), pp. 226–235.

[184] P. Wolfe. «Convergence Conditions for Ascent Methods. II: Some Corrections.» In: *SIAM Review* 13.2 (1971), pp. 185–188.

[185] R. Wolke and H. Schwetlick. «Iteratively Reweighted Least Squares: Algorithms, Convergence Analysis, and Numerical Comparisons.» In: *SIAM Journal on Scientific and Statistical Computing* 9.5 (1988), pp. 907–921.

[186] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. «A Comprehensive Survey on Graph Neural Networks.» In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2021), pp. 4–24.

[187] D. Yarotsky. «Error bounds for approximations with deep ReLU networks.» In: *Neural Networks* 94 (2017), pp. 103–114.

[188] Z. Zhan, W. Wang, and F. Chen. «Simultaneous Boundary and Interior Parameterization of Planar Domains Via Deep Learning.» In: *Computer-Aided Design* 166 (2024), p. 103621.

[189] W. Zheng, P. Bo, Y. Liu, and W. Wang. «Fast B-spline curve fitting by L-BFGS.» In: *Computer Aided Geometric Design* 29.7 (2012). Geometric Modeling and Processing 2012, pp. 448–462.

[190] S. Zulqarnain Gilani, F. Shafait, and A. Mian. «Shape-based automatic detection of a large number of 3D facial landmarks.» In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 4639–4648.