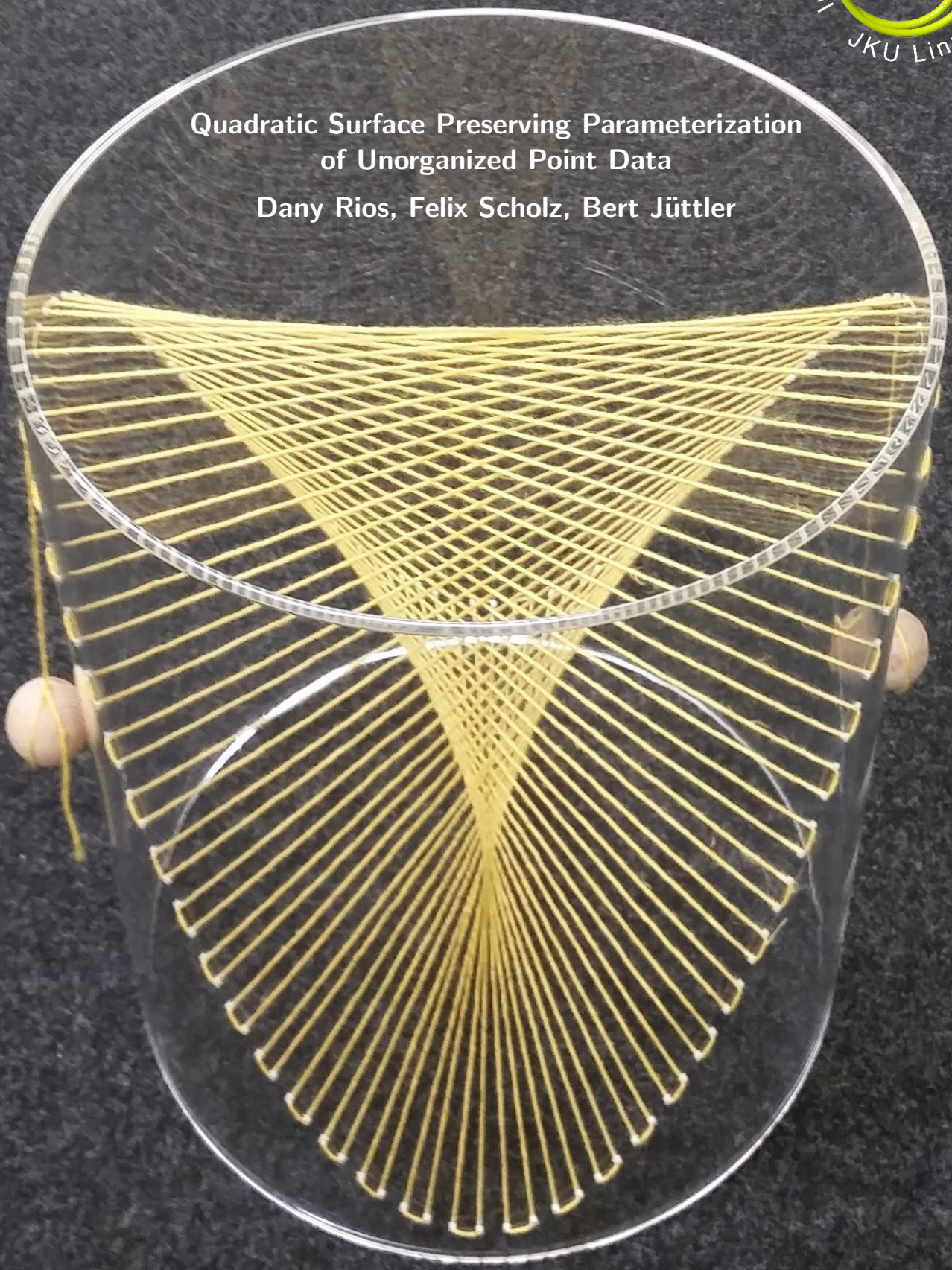
JKU

Institute of Applied Geometry
JKU Linz

Quadratic Surface Preserving Parameterization of Unorganized Point Data

Dany Rios, Felix Scholz, Bert Jüttler



AG Report No. 102
December 13, 2023
www.ag.jku.at

Quadratic Surface Preserving Parameterization of Unorganized Point Data

Dany Rios[a], Felix Scholz[a], Bert Jüttler[a]

[a]*Institute of Applied Geometry, Johannes Kepler University Linz, Altenberger Straße 69, 4040, Linz, Austria*

Abstract

Finding parameterizations of spatial point data is a fundamental step for surface reconstruction in Computer Aided Geometric Design. Especially the case of unstructured point clouds is challenging and not widely studied. In this work, we show how to parameterize a point cloud by using barycentric coordinates in the parameter domain, with the aim of reproducing the parameterizations provided by quadratic triangular Bézier surfaces. To this end, we train an artificial neural network that predicts suitable barycentric parameters for a fixed number of data points. In a subsequent step we improve the parameterization using non-linear optimization methods. We then use a number of local parameterizations to obtain a global parameterization using a new overdetermined barycentric parameterization approach. We study the behavior of our method numerically in the zero-residual case (i.e., data sampled from quadratic polynomial surfaces) and in the non-zero residual case and observe an improvement of the accuracy in comparison to standard methods. We also compare different approaches for non-linear surface fitting such as tangent distance minimization, squared distance minimization and the Levenberg Marquardt algorithm.

Keywords: surface approximation, neural networks, parameterization, point cloud parameterization, reverse engineering

1. Introduction

Many applications in computer-aided geometric design, like surface fitting, texture mapping and remeshing of 3D points (or a point cloud) require a parameterization. More precisely, given a set of points $x_i \in \mathbb{R}^3$, which is assumed to be sampled from a surface, the task is to find parameter values $u_i \in \Omega$ for each point where $\Omega \subseteq \mathbb{R}^2$. The specific requirements for the choice of parameters u_i depend on the one hand on the application and on the other hand on the structure of point data, which can be either organized or unorganized. Here, *organized* means that the point cloud comes with a triangulation T while an *unorganized* point cloud does not include any information about neighborhood relationships between points. In the organized case, the parameters are sought such that the triangulation T induces a planar triangulation $\hat{T}$ on the 2D points u_i. In the unorganized case one can consider the opposite direction: Since triangulating a set of planar points is a simple task, finding a parameterization leads to a triangulation of the point cloud. This can also be interpreted as fitting a surface to the point cloud. In order to obtain a smooth approximating surface, a parameterization of the point cloud can also be used for fitting with a parametric surface $P : \Omega \to \mathbb{R}^3$. For example, if the parameter domain Ω is a triangle, P can be a triangular Bézier surface.

Numerous results have been presented over the past years for both cases considering local or global parameterizations, different topologies and therefore a large range of applications. In the case of *organized* points, several methods have been proposed that minimize the energy functional

$$E = \frac{1}{2} \sum_{\{i,j\} \in \text{Edges}} \lambda_{ij} \|u_i - u_j\|^2 \tag{1}$$

subject to boundary conditions (i.e., constraints on the positions of the parameters associated with the boundary points), where i, j represents the index of an edge in the triangulation and u_i are the planar parameters. The proposed methods differ in how the weights $\lambda_{ij} \in \mathbb{R}$ are chosen.

Floater (1997) presents three particular parameterizations: *uniform*, *weighted least squares* and *shape-preserving*. They are generalizations of the well-known uniform and chord length parameterizations for curves. The theorem of Tutte (1963) provides the theoretical basis for Floater's method (cf. Groiss et al., 2021).

Greiner and Hormann (1997) consider a variation of (1)

$$E = \frac{1}{2} \sum_{\{i,j\} \in \text{Edges}} \lambda_{ij} \left(\|u_i - u_j\| - \ell_{ij} \right)^2 ,$$

where $\ell_{ij} = \alpha \|x_i - x_j\|$ for different choices of $\alpha \geq 0$. They also choose the weights to be $\lambda_{ij} = \frac{1}{\|x_i - x_j\|^r}$ with $r > 0$. Similarly, Eck et al. (1995) perform local parameterizations based on harmonic mappings. To this end they use weights λ_{ij} that correspond to the Dirichlet energy of piece-wise linear functions on the triangulation.

Besides the choice of the weights λ_{ij}, the above methods differ in how the boundary parameters are determined. While Floater (1997) maps the boundary of the parameter domain to the boundary of the unit square using chord length parameterization of curves, Greiner and Hormann (1997) project and fix the boundary points into the plane that fits all of them in the least square sense. Eck et al. (1995) map the boundary of the parameter domain to the unit circle.

Unlike the previous methods, Hormann and Greiner (2000) introduced a method that estimates both interior and boundary parameters. The method minimizes the deformation made by local atomic linear maps from the surface triangles to the corresponding parameter triangles.

Some methods for parameterizing *unorganized* point clouds have been proposed. Floater and Reimers (2001) generalize the method of Floater (1997) to unorganized point clouds by finding neighborhoods N_i of each vertex x_i and choosing corresponding weights λ_{ij} in several different ways. This leads to a global system

$$u_i - \sum_{j \in N_i} \lambda_{ij} u_j = 0, \tag{2}$$

where the boundary vertices are assumed to be parameterized over a convex polygon. In Giannelli et al. (2023), the authors propose training a graph neural network to predict suitable weights for the system (2).

A different approach was proposed by Barhak and Fischer (2001). They project the sample points onto a 3D base surface without segmenting the data set. Two main problems have been observed in similar approaches: the base surface could have self intersections and the grid could not reflect the density of the sample points. The authors overcome these problems via partial differential equations or neural network parameterizations. A recent review of existing 3D point cloud parameterization methods was presented by Zhu et al. (2022).

In the case of parameterizing data points for fitting curves, some applications of neural networks for directly parameterizing point sequences have been proposed by Laube et al. (2018) and Scholz and Jüttler (2021).

In the particular context of fitting smooth curves or surfaces to unorganized point clouds, a common approach is to optimize the parameterization and the control points of the fitting surface in an alternating manner. Here, the parameters at each iteration step are computed as the foot points of the data points on the fitting surface. The specific choice of error functional for the control points leads to *point distance minimization (PDM)*, *tangent distance minimization (TDM)* and *squared distance minimization (SDM)* (cf. Wang et al., 2006). Alternatively, the parameters and the control points can also be optimized simultaneously, which for the curve case has been shown to be more robust and efficient than the alternating approach (cf. Zheng et al., 2012). Rios and Jüttler (2022) studied the simultaneous optimization of parameters and control points using artificial neural networks and stochastic gradient descent.

In this paper, we propose an overdetermined generalization of the method of Floater and Reimers (2001). Instead of finding only one equation per unknown parameter u_i and solving the linear system (2), we find several convex combinations for each parameter with respect to different parameters. This leads to an overdetermined system that we solve in the least squares sense. We find that if we choose these local

convex combinations in a way that preserves the exact local parameterizations, also the resulting global parameterization is able to do so. In order to find such local parameterizations, we combine non-linear optimization with a neural network that predicts an initial parameterization that is sufficiently close to the optimum. More precisely, our neural network parameterizes twelve given points in $\mathbb{R}^3$ by barycentric coordinates with respect to three other points. The output of our neural network is then further improved by fitting quadratic surfaces to the given points while either alternatingly or simultaneously optimizing the parameter values. To this end, we compare the use of PDM, TDM, SDM and directly using the Levenberg-Marquardt algorithm. Since our method produces barycentric coordinates, the result is a suitable choice of weights for an overdetermined global system analogous to (2), where each chosen neighborhood of a vertex consists of three vertices.

In Section 2, we summarize the parameterization method from Floater and Reimers (2001) and present our overdetermined generalization. We also derive conditions for the existence and uniqueness of the parameterization resulting from a specific choice of local convex combinations. In Section 3, we present our methods for finding optimal local parameterizations by neural network initialization and fitting quadratic surfaces using different methods for non-linear optimization. In the numerical experiments in Section 4 we demonstrate the effectiveness of our local parameterization algorithm. Moreover, we compare the performance of classical surface fitting methods with direct non-linear optimization. In the last section, we combine the local parameterizations using overdetermined barycentric parameterization and we present numerical experiments for the global surface parameterization, comparing our method to the standard methods.

2. Point Cloud Parameterization

In this section, we first describe a standard approach for parameterizing unstructured point clouds. We then present a new overdetermined generalization of the standard method as well as its theoretical analysis.

2.1. Meshless barycentric parameterization

In Floater and Reimers (2001), a method for parameterizing a three-dimensional point cloud over a convex planar domain is presented. We summarize this method here.

Let $X = (x_1, \dots x_N)$ with $x_i \in \mathbb{R}^3$ a point cloud and D a convex domain in $\mathbb{R}^2$. We assume that X is decomposed as $X = X_I \cup X_B$ into *interior points* $X_I = (x_1, \dots, x_n)$ and *boundary points* $X_B = (x_{n+1}, \dots, x_N)$. We denote by $I = \{1, \dots n\}$ and $B = \{n+1, \dots N\}$ the corresponding index sets. Moreover, we assume that a one-dimensional parameterization $U_B = (u_{n+1}, \dots, u_N) \subset \partial D$ of X_B over the boundary of D is given. We aim to find parameters $U_I = (u_1, \dots, u_n) \subset D$ that correspond to the interior points in X_I. To this end, we assume that each parameter $u_i \in U_I$ can be written as a convex combination

$$u_i = \sum_{j \in N_i} \lambda_{ij} u_j, \tag{3}$$

where $N_i \subset I \cup B$ is any subset of the index set and $\lambda_{ij} > 0$ of $j \in N_i$, $\lambda_{ij} = 0$ for $j \notin N_i$ and

$$\sum_{j \in N_i} \lambda_{ij} = 1.$$

Equation (3) corresponds to a linear system

$$AU_I = C, \tag{4}$$

where $A_{ij} = \delta_{ij} - \lambda_{ij}$ for $i, j = 1, \dots n$ and $c_i = \sum_{j=n+1}^{N} \lambda_{ij} u_j$.

If A is invertible, the solution of eq. (4) is the parameterization of X_I. The specific choice of the sets N_i and the convex combination weights λ_{ij} is open and determines the resulting parameterization.

2.2. Overdetermined meshless barycentric parameterization

In this work, we generalize the approach summarized in the previous section by allowing more than one equation per parameter in eq. (3). More precisely, for each $i \in I$, we determine a number of $L_i \in \mathbb{N}_+$ subsets

$$N_i^\ell \subset I \cup B, \quad \ell = 1, \dots, L_i$$

and weights λ_{ij}^ℓ with $\lambda_{ij}^\ell > 0$ for $j \in N_i^\ell$, $\lambda_{ij}^\ell = 0$ for $j \notin N_i^\ell$ such that

$$\sum_{j \in N_i^\ell} \lambda_{ij}^\ell = 1.$$

Then, we consider the equations

$$u_i = \sum_{j \in N_i^\ell} \lambda_{ij}^\ell u_j, \tag{5}$$

for $i = 1 \dots n$, $\ell = 1 \dots L_i$, which lead to the overdetermined system of equations

$$AU_I = C, \tag{6}$$

where

$$A_{k(i,\ell)j} = \delta_{ij} - \lambda_{ij}^\ell,$$
$$b_{k(i,\ell)} = \sum_{j=n+1}^{N} \lambda_{ij}^\ell u_j$$

and

$$k : \bigcup_{i \in I} (\{i\} \times \{1, \dots, L_i\}) \to \{1, \dots, \sum_{i \in I} L_i\}$$

is any ordering. If eq. (6) has maximum rank, we solve it in the least-squares sense, meaning that we find $U_I \in \mathbb{R}^{n \times 2}$, such that

$$\|AU_I - C\|_2^2 \tag{7}$$

is minimal.

2.3. Conditions for existence and uniqueness

In general, neither the linear system eq. (4) nor the least squares problem eq. (7) is guaranteed to have a unique solution. However, in both cases there are easy to check conditions on the neighborhoods N_i and N_i^ℓ, respectively. We use the notion of *weakly chained diagonally dominant* matrices: Recall that a row i of a matrix A is *weakly diagonally dominant* if $|A_{ii}| \geq \sum_{j \neq i} |A_{ij}|$. and that it is *strongly diagonally dominant* if $|A_{ii}| > \sum_{j \neq i} |A_{ij}|$.

Moreover, a matrix $A \in \mathbb{R}^{n \times n}$ is *weakly chained diagonally dominant* if every row of A is weakly diagonally dominant and from each row i there is a path to a strongly diagonally dominant row, i.e. a sequence $(i = j_1, j_2, \dots, j_k)$ such that the j_k-th row of A is strongly diagonally dominant and $A_{j_a j_{a+1}} > 0$ for $a = 1 \dots k - 1$.

Theorem 1 (Shivakumar and Chew, 1974)**.** *Every weakly chained diagonally dominant matrix has full rank.*

We use this fact to show that the matrix A in eq. (4) has full rank. The following theorem is equivalent to (Floater and Reimers, 2001, Proposition 3.3) where it was proven directly without invoking weakly chained diagonally dominant matrices. We consider the directed graph $G = (V, E)$ with

$$V = I \cup B$$
$$E = \{(i, j) : i \in I, j \in N_i\}.$$

A vertex $i \in V$ is called *boundary connected* if there is a path in G from i to a vertex $j \in B \subset V$.

Theorem 2 (Floater and Reimers, 2001)**.** *If every vertex in G is boundary connected, the matrix A in eq.* (4) *has full rank.*

Proof. We consider the extended matrix $\tilde{A} \in \mathbb{R}^{N\times N}$ with

$$\tilde{A}_{ij} = \delta_{ij} - \lambda_{ij},$$

where we set $\lambda_{ij} = 0$ if $i > n$. Thus, $\tilde{A}$ is a block matrix

$$\begin{pmatrix} A & -C \\ 0 & I \end{pmatrix}$$

Clearly, $\tilde{A}$ has full rank if and only if A has full rank.

The diagonal entries of $\tilde{A}$ are all 1. For $i > n$, the diagonal entry is the only entry of the i-row and therefore all these rows are strongly diagonally dominant. For $i \le n$, we have

$$\sum_{j\neq i} = \sum_{j\in N_i} \lambda_{ij} = 1$$

and therefore these rows are weakly diagonally dominant. Finally, a path in G from a vertex i to a vertex $j \in B \subset V$ corresponds to a path from the i-th row in $\tilde{A}$ to a row $j > n$. Since the rows $j > n$ are strongly diagonally dominant and every vertex is boundary connected in G, we conclude that $\tilde{A}$ is weakly chained diagonally dominant and therefore has full rank. By construction of $\tilde{A}$, this means that also A has full rank. □

Remark. In (Floater and Reimers, 2001, Proposition 3.3) it was additionally proven that the resulting interior parameters u_i for $i \in I$ lie in the convex hull of the boundary parameters U_B.

We now generalize this result to the case of the overdetermined least squares problem. As before, we consider a graph $G = (V, E)$ with

$$V = I \cup B \tag{8}$$

$$E = \{(i, j) : i \in I, j \in N_i^\ell \text{ for some } \ell \in \{1, \dots, L_i\}\} \tag{9}$$

and we call a vertex i of this graph *boundary connected* if there is a path in G from i to a boundary vertex.

Theorem 3. *If every vertex in G is boundary connected, the least squares problem*

$$\|AU_I - C\|_2^2$$

with $A \in \mathbb{R}^{(\sum_{i\in I} L_i)\times n}$ defined as in (6) *has a unique solution.*

Proof. The least squares problem has a unique solution if and only if A has a $n \times n$ submatrix, obtained by removing rows of A, that has full rank. We construct such a matrix by removing rows from A and removing the corresponding edges from G simultaneously.

Consider the vertex $1 \in I$ with neighborhoods N_1^ℓ for $\ell = 1 \dots L_1$. Then by our assumption, there is a path

$$1 = j_1 \dots j_k$$

in $\hat{G}$ with $j_k \in B \subset V$. From this follows that $j_1 \in N_1^\ell$ for at least one $\ell \le L_1$, say, without loss of generality, for $\ell = 1$. We find the matrix A_1 by removing all rows $k(1, \ell)$ for $\ell = 2 \dots L_i$ (if $L_1 > 1$). Moreover, we construct the graph G_1 by setting $L_1 = 1$ in (9). Now, every vertex i in G_1 is still boundary connected: If i had a path

$$i = q_1 \dots q_{m-1}(q_m = 1)q_{m+1} \dots q_r$$

in G with $q_r \in B$, then

$$i = q_1 \dots q_{m-1} j_1 \dots j_k$$

is a path in G_1 from i to a vertex in B. Moreover, A_1 still corresponds to a system in the form of (5) with the same sets N_i^ℓ that were used to generate G_1.

We continue this process for the vertices $2, \ldots, n$ to arrive at the $n \times n$-matrix

$$(A_n)_{ij} = \delta_{ij} - \lambda_{ij}^1,$$

where we assume without loss of generality that we kept the equation corresponding to $\ell = 1$ for each interior vertex. Moreover, the graph G_n obtained by this procedure is equal to the graph generated by the neighborhoods sets N_i^1. Since G_n is boundary connected, A_n and G_n fulfill the assumptions of Theorem 2 and thus A_n has full rank. Since A_n is a submatrix of A, the least squares problem has a unique solution. □

2.4. Choosing the neighborhoods and weights

What remains to be chosen for both the barycentric parameterization described in Section 2.1 and our new overdetermined barycentric parameterization described in Section 2.2 are the neighborhoods N_i^ℓ as well as the weights λ_{ij}^ℓ. Floater and Reimers (2001) considered three particular choices: *uniform parameterization*, *reciprocal distance parameterization* and *shape preserving parameterization*. In both the uniform and the reciprocal distance parameterization, neighborhoods for each vertex are chosen as the ball neighborhoods

$$N_i = \{j \in \{1, \ldots, N\} \backslash \{i\} : \|x_i - x_j\|_2 < r\} \tag{10}$$

for some fixed radius $r > 0$. Then, the uniform weights are defined as $\lambda_{ij} = \frac{1}{|N_i|}$ and the reciprocal distance weights are defined as

$$\lambda_{ij} = \frac{\|x_i - x_j\|_2^{-1}}{\sum_{k \in N_i} \|x_i - x_k\|_2^{-1}}.$$

For the shape preserving parameterization, the neighborhoods are found by projecting a ball neighborhood of each vertex i onto its best approximating plane and generating a Delaunay triangulation on that plane. The neighbors of i in the Delaunay triangulation form the neighborhood N_i and the weights are found by preserving the angle ratios around the vertex as detailed in Floater (1997).

In this work, we leverage the additional flexibility of the overdetermined barycentric parameterization to improve the performance of the parameterization algorithm. To this end, we need to find a number of neighborhoods N_i^ℓ and weights λ_{ij}^ℓ for each vertex. In the following section we describe our method to choose these combinations by finding locally optimal parameterizations for a fixed number of points. Each of these parameterizations will lead to a number of equations of the form (5).

3. Finding local parameterizations

In order to generate equations of the form (5), we develop an algorithm to parameterize a fixed small number of points by barycentric coordinates with respect to three points of this point cloud.

3.1. Overview of the algorithm

We take as input a point cloud of fifteen points

$$\{x_1, \ldots, x_{12}\} \cup \{x_{13}, x_{14}, x_{15}\}$$

in $\mathbb{R}^3$, where the first twelve points are considered as *interior points* and the last three points are considered as *vertex points*. We consider fifteen points for two reasons: On the one hand, the fifteen points uniquely determine a quadratic surface in general. On the other hand, the number is not too large and thus we are able to train a neural network that reproduces the parameters (see below). The output of our proposed algorithm are barycentric coordinates

$$u_i = \sum_{j=13}^{15} \lambda_{ij} u_j$$

for $i = 1 \dots 12$ such that

$$\lambda_{ij} > 0, \qquad \sum_{j=13}^{15} \lambda_{ij} = 1.$$

In principle, there are infinitely many such barycentric parameterizations. In order to obtain good parameters for approximation with polynomial surfaces, our algorithm attempts to reproduce parameterizations of quadratic Bézier surfaces

$$P(u, v, w) = \sum_{a+b+c=2} b_{abc} B^2_{abc}(u, v, w), \tag{11}$$

where B^2_{abc} are the Bernstein polynomials and $b_{abc} \in \mathbb{R}^3$. This means that if the vertex points are

$$x_{13} = P(1, 0, 0), \quad x_{14} = P(0, 1, 0), \quad x_{15} = P(0, 0, 1)$$

and the interior points were sampled i.e. for $i = 1, \dots 12$ as

$$x_i = P(\lambda_{i13}, \lambda_{i14}, \lambda_{i15}), \tag{12}$$

then our algorithm should result in the exact barycentric parameters $(\lambda_{i13}, \lambda_{i14}, \lambda_{i15})$ for all $i = 1, \dots, 12$. In the general case of data points that were not sampled from a quadratic surface, we aim to minimize the fitting error of the best approximation of the point cloud with a quadratic Bézier surface.

In order to make this task more feasible, we restrict the inputs to point clouds where the orthogonal projection of the interior points $\{x_1, \dots, x_{12}\}$ into the plane defined by the vertex points x_{13}, x_{14}, x_{15} lies inside of the triangle spanned by the vertex points.

Our algorithm consists of three steps:

1. We find the affine transformation that transforms the triangle with vertices x_{13}, x_{14}, x_{15} into a standard triangle. The transformation is applied to the 12 interior points. The details are described in Section 3.2.
2. We generate initial parameters by one of two possible methods. A simple way of choosing initial parameters is projection of the interior points onto the plane generated by the vertex points. In order to obtain better results, we train a neural network for this task and use its output as initial parameters, see Section 3.3.
3. We perform a non-linear optimization to find the optimal parameterization starting from our initial parameterization. We compare classical surface approximation methods that alternate between parameter correction and minimizing the *point distance minimization* (PDM), *tangent distance minimization* (TDM) or *squared distance minimization* (SDM) functional with simultaneously optimizing the parameters and the control points using the Levenberg Marquardt (LM) algorithm, see Section 3.5.

Figure 1 presents a graphical summary of the three steps.

3.2. Standardization

The standardization process is only required for the neural network evaluation, since training a network is more efficient on standardized data

We transform the given points into a standard configuration by mapping the three vertex points x_{13}, x_{14}, x_{15} to the standard triangle

$$\hat{x}_{13} = [0, 0, 0], \quad \hat{x}_{14} = [1, 0, 0], \quad \hat{x}_{15} = \left[\frac{1}{2}, \frac{\sqrt{3}}{2}, 0\right].$$

To do so, we find an affine transformation

$$\alpha(x) = v + Ax$$

where A is a 3×3 matrix and v is a vector in $\mathbb{R}^3$, see Fig. 2. To find α, we enforce the following conditions:

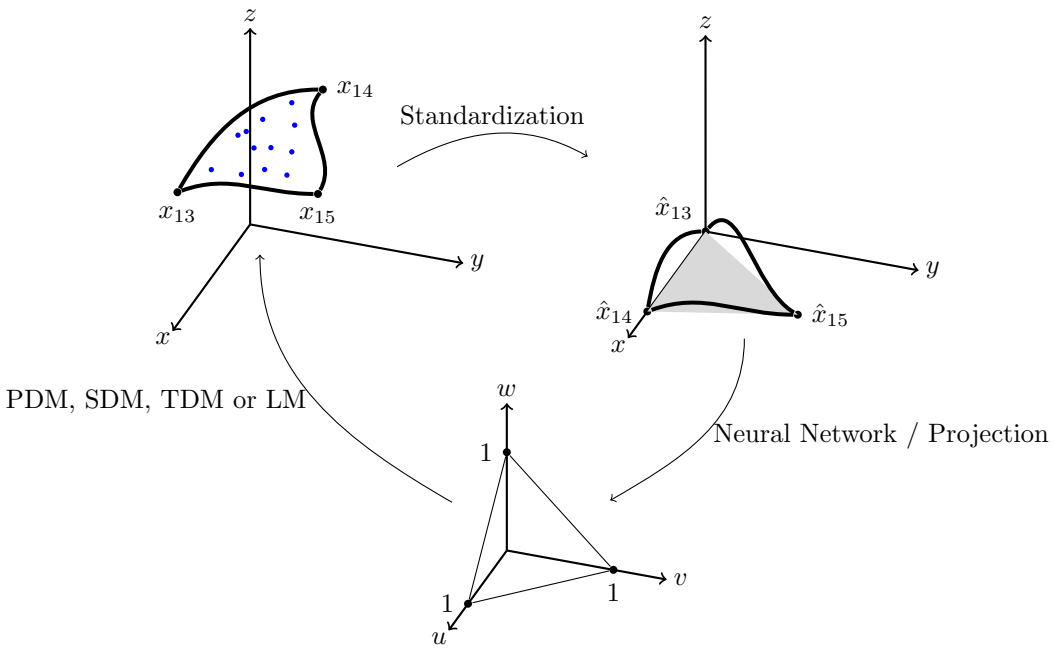


Figure 1: Steps followed by the parameterization process.

1. $\alpha(x_i) = \hat{x}_i$ for $i = 13, 14, 15$, and
2. $\alpha(b + Ln) = \hat{b} + \hat{L}\hat{n}$, where $\{b, n, L\}$, $\{\hat{b}, \hat{n}, \hat{L} = 3\}$ are the barycenters, normal vectors and perimeters of the triangles $x_{13}x_{14}x_{15}$ and $\hat{x}_{13}\hat{x}_{14}\hat{x}_{15}$ respectively.

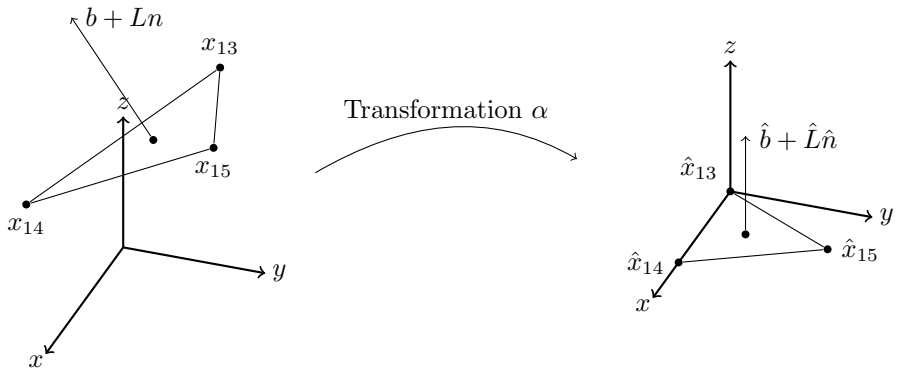


Figure 2: Standardization process.

The condition $\alpha(x_{13}) = \hat{x}_{13}$ leads us to $v = -Ax_{13}$ and the matrix A is found by solving a system of nine equations involving the remaining three conditions:

$$\begin{aligned}
\alpha(x_{14}) &= A(x_{14} - x_{13}) = \hat{x}_{14}, \\
\alpha(x_{15}) &= A(x_{15} - x_{13}) = \hat{x}_{15}, \\
\alpha(b + Ln) &= A(b + Ln - x_{13}) = \hat{b} + \hat{L}\hat{n}.
\end{aligned}$$

We transform the points x_i to their standard position $\hat{x}_i = \alpha(x_i)$. Since barycentric coordinates are preserved under affine transformations, this standardization does not affect the result.

3.3. Initial parameterization using an artificial neural network

Besides the straightforward initialization of the parameters via orthogonal projection, we propose another method to estimate the barycentric coordinates, which is based on residual neural networks. More precisely, we construct a network that takes as input 12 points $\hat{x}_1 \dots \hat{x}_{12}$ in the standard configuration described in Section 3.2 and gives as output a list of barycentric coordinates (u_i, v_i, w_i) with respect to the vertices $\hat{x}_{13}$, $\hat{x}_{14}$ and $\hat{x}_{15}$ for $i = 1, \dots, 12$ with $u_i, v_i, w_i \geq 0$ and $u_i + v_i + w_i = 1$.

3.3.1. Network architecture

To design our network, we use a residual neural network. Residual neural networks were initially introduced for classification problems on image data (He et al., 2016) and have later been adapted to regression problems (Chen et al., 2020). Recently they have been successfully applied to the problem of curve parameterization (Scholz and Jüttler, 2021). The key idea of residual neural networks is that they consist of several blocks that have an additive shortcut connection between their input and their output. This means that instead of approximating a function $h(x)$, each residual block approximates the residual $f(x) = h(x) - x$. In practice, the use of shortcut connections makes it possible to effectively train deeper neural networks.

Our network architecture is shown in Figure 3. On the left-hand side we show the layout of a single residual block, which consists of three densely connected layers with batch normalization and ReLu activation. The shortcut connection is the identity if the dimensions match, otherwise it consists of an additional densely connected layer with batch normalization.

On the right-hand side, we show the overall layout of the network. Since our input consists of 12 points in $\mathbb{R}^3$, the total input dimension is 36. After an input layer with ReLu activation function, we have four ResNet blocks and an output layer that transforms the data to the output dimension 36 (three barycentric coordinates u_i, v_i, w_i per input point). Since the barycentric coordinates need to lie between 0 and 1, we use the sigmoid function as our output activation function. Finally, we enforce the barycentric coordinates u_i, v_i, w_i for each point to sum up to one by dividing them by their sum.

3.3.2. Training data generation

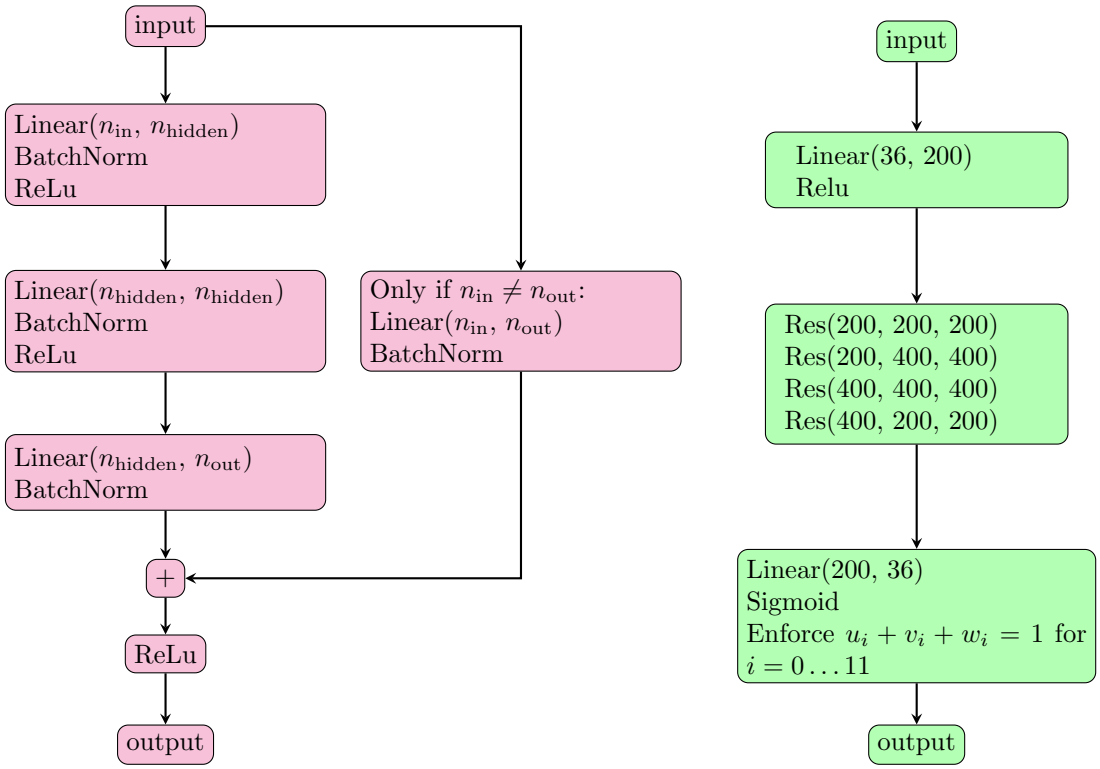
For our training data, we create 500,000 quadratic triangular Bézier surfaces in the standard position described in Section 3.2, which means that we set the control points b_{200}, b_{020} and b_{002} of the quadratic surface to the points $[0, 0, 0]$, $[1, 0, 0]$ and $[1/2, \sqrt{3}/2, 0]$, respectively. The remaining control points $b_{110}, b_{101}, b_{011}$ are chosen randomly in a box $[-0.4, 0.4] \times [-0.4, 0.4] \times [-1, 1]$ around the midpoints of the edges, see Figure 4. Afterwards, we generate 12 random barycentric coordinates (u_i, v_i, w_i) for $i = 0, 1, \dots, 11$ with $u_i, v_i, w_i \geqslant 0.05$, thereby ensuring that they are at a minimum distance from the boundary. The corresponding 3D points are found by evaluating the quadratic Bézier representation with the control points b_{200}, b_{020}, b_{002}, b_{110}, b_{101}, b_{011}.

3.4. Training

To train our network we use an unsupervised learning strategy, meaning that instead of using labeled data we use the output of the neural network as parameters (u_i, v_i, w_i) for $i = 1, \dots, 12$ to fit the best approximating quadratic triangular Bézier surface P to the input points. We then evaluate P at the parameters (u_i, v_i, w_i) and compute the mean squared loss with respect to the input points as

$$\frac{1}{12} \sum_{i=1}^{12} \|P(u_i, v_i, w_i) - x_i\|^2. \tag{13}$$

We implemented our method in Python using the PyTorch library (Paszke et al., 2019), which includes methods for solving linear least squares problems that provide the necessary gradients and that can be run on a GPU. The training process is carried out using the well known algorithm ADAM (A method for stochastic optimization, see Kingma and Ba, 2014), with a learning rate of 10^{-4} and batch size 16.



(a) Residual block Res(n_{in}, n_{hidden},n_{out})

(b) Neural network

Figure 3: The residual neural network for surface parameterization. The left diagram shows the layout of a residual block, while the right diagram shows the layout of the whole network.

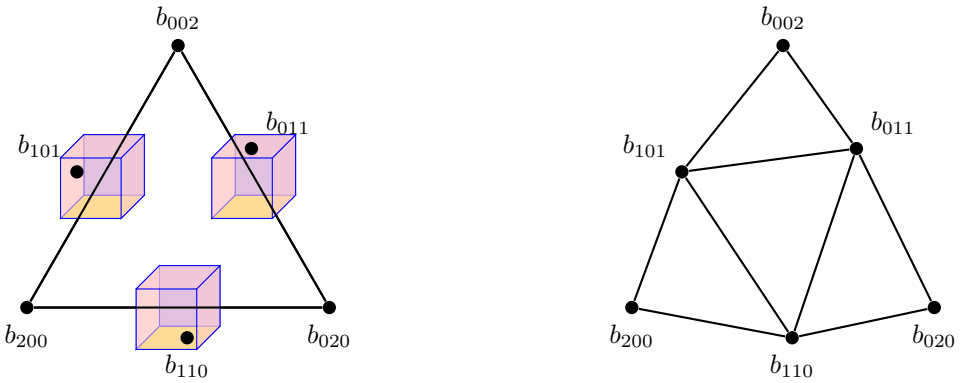


Figure 4: Training data generation. Left: Random control points chosen in a box around the edge midpoints. Right: Resulting control polygon of a quadratic Bézier surface.

3.5. Non-linear optimization of the local parameterization

After obtaining the initial parameters, we perform non-linear optimization to parameterize the 12 interior points $x_1, \dots x_{12}$ in the optimal way for fitting a quadratic triangular Bézier surface (11). This means that we minimize (13) by adjusting both the control points b_{ijk} of P and the parameters (u_i, v_i, w_i).

Since we assume that vertex points x_{13}, x_{14}, x_{15} are the vertices of the target surface, we fix the corre-

sponding control points as

$$b_{200} = x_{13}, \quad b_{020} = x_{14}, \quad b_{002} = x_{15} \ .$$

The rest of the control points b_{ijk} as well as the barycentric parameters (u_i, v_i, w_i) for $i = 1 \dots 12$ with respect to the vertices are to be optimized. The final output of our algorithm are then the optimized parameters

$$\lambda_{i13} = u_i, \quad \lambda_{i14} = v_j, \quad \lambda_{i15} = w_i$$

for all $i = 1, \dots 12$.

We present two different approaches for the optimization: Alternating surface fitting with parameter correction, and simultaneous surface fitting using the Levenberg Marquardt algorithm. A comparison of the two strategies is presented in the next section.

3.5.1. Alternating optimization of parameters and control points

Wang et al. (2006) present a full description of point distance minimization (PDM), tangent distance minimization (TDM) and squared distance minimization (SDM). They establish the equivalence of these schemes to well-known methods for nonlinear optimization: Variant of the steepest descent, Gauss-Newton method and Newton method, respectively.

In all these methods, the control points are first optimized such that a specific error functional is minimal. Depending on the method we consider the *PDM* functional

$$\frac{1}{2} \sum_{i=1}^{12} \big(P(u_i, v_i, w_i) - x_i\big)^2,$$

the *TDM* functional

$$\sum_{i=1}^{12} \big((P(u_i, v_i, w_i) - x_i) \cdot N(u_i, v_i, w_i)\big)^2,$$

and the *SDM* functional

$$\sum_{i=1}^{12} \max\left\{0, \frac{d}{d-\rho_1}\right\} \big((P(u_i, v_i, w_i) - x_i) \cdot T_1(u_i, v_i, w_i)\big)^2 + \max\left\{0, \frac{d}{d-\rho_2}\right\} \big((P(u_i, v_i, w_i) - x_i) \cdot T_2(u_i, v_i, w_i)\big)^2 + \big((P(u_i, v_i, w_i) - x_i) \cdot N(u_i, v_i, w_i)\big)^2,$$

where d is the signed distance to the foot point, N is the unit vector field, ρ_1, ρ_2 are the principal curvature radii and T_1, T_2 are the principal curvature directions. Each of these functionals results in a linear least squares problem, which can be solved using standard methods.

In the second step, the parameters are adjusted by a process called *parameter correction*.

Optimization of the control points and parameter correction is performed alternatingly until a stopping criterion is reached.

3.5.2. Simultaneous optimization of parameters and control points

As an alternative to the classical surface fitting methods described in the previous section, we propose to directly solve the non-linear problem

$$\frac{1}{2} \sum_{i=1}^{12} \big(P(u_i, v_i, w_i) - x_i\big)^2,$$

where the objective function is minimized for the parameters (u_i, v_i, w_i) and the control points b_{ijk} simultaneously. This results in a non-linear least squares problem, which we solve using the *Levenberg-Marquardt* algorithm (LM).

4. Numerical experiments for the local parameterization

Example 1 (Single surface)**.** We sample 12 points $x_1, \ldots, x_{12}$ on a quadratic triangular Bézier surface with control points

$$b_{200} = [2,1,1],\ b_{020} = [1,2,1],\ b_{002} = [1,1,2],\ b_{110} = [1.8,1.4,1.8],\ b_{101} = [1.1,0.9,1.7],\ b_{011} = [1.8,1.9,1.9]$$

where the control points $b_{200}, b_{020}, b_{002}$ where fixed representing an equilateral triangle, see Fig. 5.

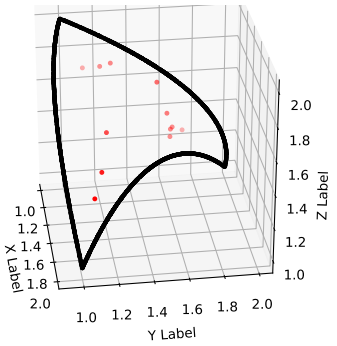


Figure 5: Twelve points on the quadratic triangular Bézier surface.

First of all we transform the surface into the standard position via the algorithm presented in Section 3.2, obtaining the points $\hat{x}_1, \ldots \hat{x}_{12}$. We evaluate the neural network at the Cartesian coordinates of the points $\hat{x}_1, \ldots \hat{x}_{12}$ and we obtain an estimation of the barycentric coordinates of the points $x_1, \ldots, x_{12}$ with respect to the vertices $b_{200}, b_{020}, b_{002}$. Then we optimize the parameterization using all the methods: PDM, TDM, SDM and LM. The resulting maximum and mean squared error of the parameters and the mean squared error of the fitted surfaces are presented in Table 1 for all the methods. We observe that the LM process converged towards the exact parameters that were used to sample the 12 points. Since the surface was not chosen to be in standard position, this confirms that the standardization process does not spoil the result. As can be observed in Table 1, the errors after applying PDM are the same as the ones after initialization since the method diverged.

	Initialization	PDM	TDM	SDM	LM-algorithm
Max parameter error	$9.5e-02$	$9.5e-02$	$1.1e-04$	$1.2e-05$	$5.6e-30$
Mean squared parameter error	$4.6e-03$	$4.6e-03$	$1.4e-08$	$1.4e-10$	$3.2e-15$
Mean squared fitting error	$1.9e-04$	$1.9e-04$	$3.8e-17$	$1.2e-19$	$1.0e-32$

Table 1: Errors for a single surface with 12 points.

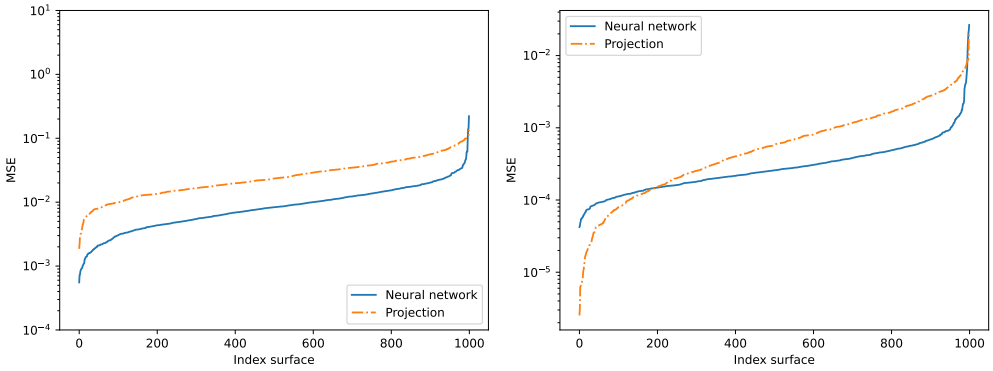
Example 2 (Standard least-squares fitting)**.** We compare the orthogonal projection and the output of the neural network on surfaces from the same class that was used for training the neural network. To this end, we generate $1,000$ quadratic triangular Bézier surfaces in standard position (see Section 3.2) with fixed vertex control points

$$b_{200} = [0,0,0], \quad b_{020} = [1,0,0], \quad b_{002} = \left[\frac{1}{2}, \frac{\sqrt{3}}{2}, 0\right]. \tag{14}$$

The remaining control points b_{110}, b_{101} and b_{011} are chosen randomly in a box $[-0.4, 0.4] \times [-0.4, 0.4] \times [-1, 1]$ around the midpoints of the triangle defined by b_{200}, b_{020} and b_{002}, see Figure 4. We call this class of surfaces *Standard Surfaces*.

We then sample from each surface 12 random interior points x_i for $i = 1, \ldots 12$ with barycentric coordinates (u_i, v_i, w_i) with $u_i, v_i, w_i \geq 0.05$.

To compare the methods, we obtain two sets of parameters from the points x_i for $i = 1, \ldots, 12$ via orthogonal projection and evaluation of the neural network, respectively. For each set of 12 barycentric coordinates we compute the mean square error with respect to the original parameters (u_i, v_i, w_i), see Fig. 6(a). Moreover, we fit a quadratic triangular Bézier surface by least squares to the points Q_i using the parameters and show the mean square errors for each surface in Figure 6(b).



(a) MSE parameters

(b) Mean squared fitting error

Figure 6: Standard surfaces (degree 2): MSE of the initial parameterizations and MSE of the fitted surfaces.

Note that in all plots the surfaces where sorted in ascending order individually with respect to the mean square errors to get a better understanding of the results.

We observe that the neural network predicts parameters that are closer to the original ones, compared to the orthogonal projection. Moreover, the fitting error resulting from the neural network is often smaller and it generally behaves in a more uniform way. For example, we count the number of surfaces with mean squared error under the threshold 10^{-3}, which is 654 for projection and 952 for the neural network.

Next, we compare the performance of orthogonal projection and the neural network in the non-zero residual case, i.e., on data that was not sampled from quadratic surfaces.

In order to generate such surfaces, we generate triangular Bézier parameterizations of degree 3 and 4 of the planar triangle with vertices as in (14). We then add to all control points except for the vertex control points a perturbation that is randomly sampled from the box $[-0.4, 0.4] \times [-0.4, 0.4] \times [-1, 1]$, see Figure 7.

In Figure 8, we show the resulting mean squared errors of the fitted surfaces, both in the cubic and the quartic case. In both cases, the means squared error resulting from the neural network is small compared to the ones resulting from orthogonal projection. Counting again the surfaces with mean squared error below the threshold 10^{-3}, we see that in the cubic cases projection results in 161 surfaces while the neural network results in 373 surfaces with errors below the threshold. In the quartic case, projection results in 71 surfaces and the neural network in 183 surfaces with mean squared errors below 10^{-3}.

Example 3 (Standard surfaces: Improving the parameterization via PDM, TDM, SDM and Levenberg–Marquardt algorithm)**.** We now revisit the previous example for data of degree 2,3 and 4 and additionally improve the parameterization via quadratic surface fitting as described in Section 3.5. In particular, we apply the PDM, TDM, SDM and LM algorithms until we reach an error of 10^{-15} or a maximum number of 20 iterations. In case the employed non-linear optimization method does not converge, we use the initial parameters obtained from projection or the neural network, respectively. Figure 9 shows the mean squared errors of the resulting fitting surfaces for all methods and degrees 2, 3, and 4.

We observe that in many cases PDM, TDM and SDM do not converge. For TDM and SDM, this means that the resulting error is the same as the initial errors shown in Figures 6 and 8. PDM slightly improves the error in all cases but still fails to converge. The Levenberg-Marquardt algorithm, however, converges in a significantly higher number of cases.

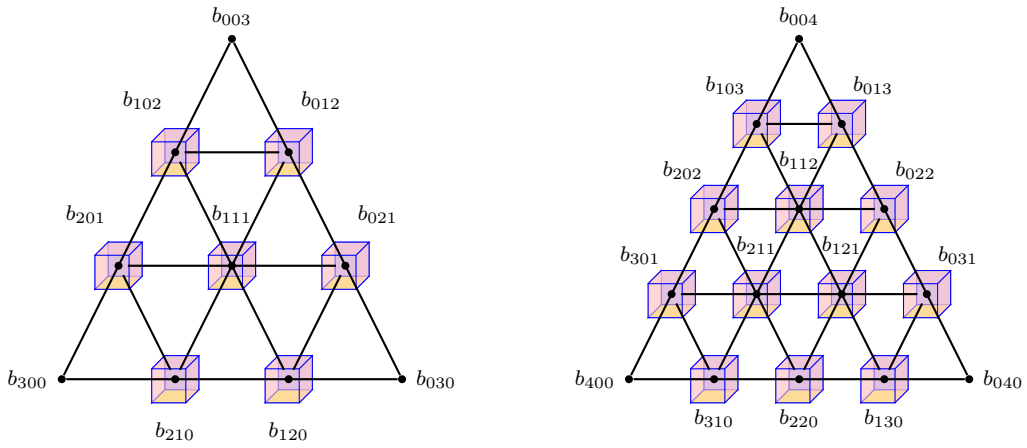
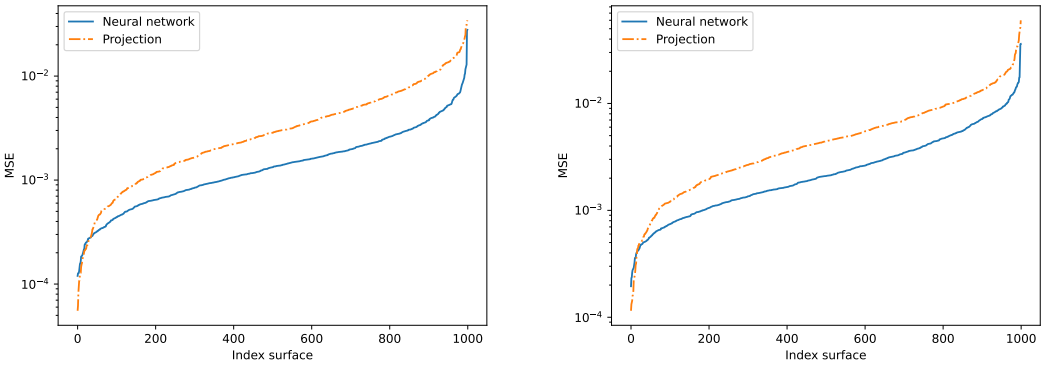


Figure 7: Control net of cubic (left) and quartic (right) Bézier surfaces with random non-vertex control points.



(a) Cubic case

(b) Quartic case

Figure 8: Surfaces of degree 3 and 4: MSE of the surfaces obtained by standard fitting.

For the quadratic case we report in Table 2 the ratio of surfaces for which the different methods converged. We observe a large advantage of the neural network initialization over the initialization by projection, especially for SDM and TDM.

Initialization	PDM	TDM	SDM	LM
Projection	0%	12.9%	9.1%	88.7%
Neural network	0%	42.6%	51.8%	95.2%

Table 2: Standard surfaces: Computation time and percentage of quadratic surfaces that converged when applying the different non-linear surface fitting methods.

In the cubic and quartic cases, TDM and SDM did not improve the result over the initial fitting and PDM improved slightly. LM, however, largely improves the resulting error. Due to its robustness it gives almost the same result for both initialization methods.

Example 4 (High curvature surfaces)**.** In the previous section, we observed that the initial fitting error when using the neural network initialization is significantly lower than when using initialization by projection. However, due to the robustness of LM, the resulting fitting error after the optimization with LM only differs slightly. For surfaces with high curvature, projection can lead to a suboptimal initialization since points that are far from each other on the original surface can be projected to planar parameters that are close to

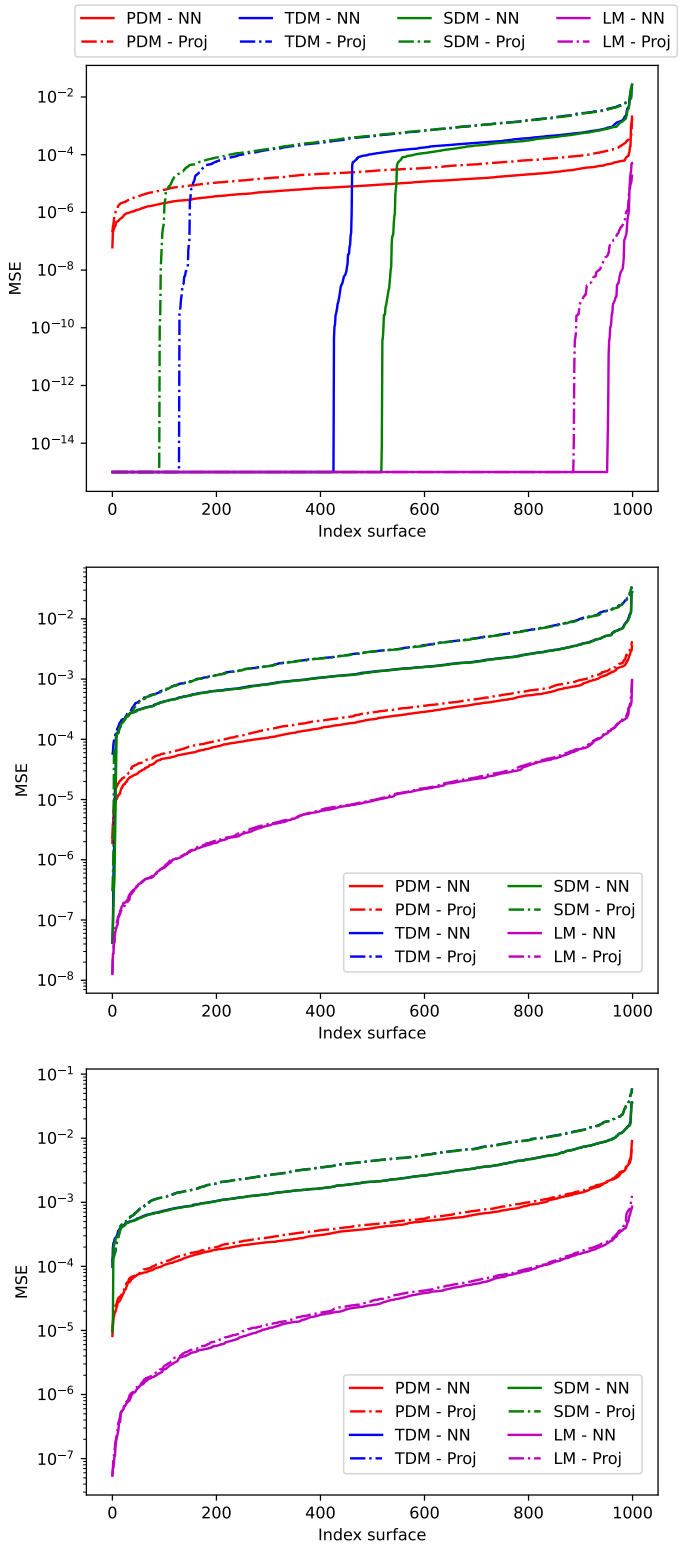


Figure 9: Surfaces of degree 2 (top), 3 (center) and 4 (bottom): MSE of the fitted surfaces after applying PDM, TDM, SDM and LM with initial parameters via projection and neural network evaluation.

each other. Moreover, the projected parameters can lie outside of the triangle spanned by the three vertices, leading to negative barycentric coordinates. In this section, we test the behavior of the neural network initialization as well as the initialization by projection on such surfaces.

To this end, we generate surfaces of degree 2, 3 and 4 in the same way as in the previous sections, with the modification that instead of varying the control points based on a uniform distribution inside a prescribed box centered at 0, we vary them in a way that favors large deformations. More precisely, we generate surfaces by perturbing the non-vertex control points (see Figures 4 and 7) of the parameterization of the planar triangle with values randomly sampled from $[0.3, 0.4] \times [0.3, 0.4] \times [0.7, 1]$. Therefore, the generated surfaces are in the same class as the ones from the previous section, but the expected value of the perturbation in each direction is larger than zero.

In Figure 10, we report the standard mean squared fitting errors that result from applying the neural network and projection as well as the mean squared fitting errors after applying PDM, TDM, SDM and LM. We observe that the neural network leads to a significantly better initial guess for degree 2, 3 and 4. This entails much smaller errors after applying the non-linear surface fitting methods. Moreover, we observe that LM recovers the exact parameterization in the zero-residual case for a large number of surfaces while the other methods fail to converge in many cases. In Table 3 we report the percentage of surfaces that converged for all methods in the zero-residual case. Even more, the computational time when applying the whole method with LM to 1000 quadratic high curvature surfaces is half when using the neural network initialization than when using projection, see Table 4.

Initialization	PDM	TDM	SDM	LM
Projection	0%	0.01%	0.03%	44.7%
Neural network	0%	20.0%	25.6%	78.2%

Table 3: High curvature surfaces: Computation time and percentage of quadratic surfaces that converged when applying the different non-linear surface fitting methods.

For the cubic and quartic case, one observes an improvement of more than one order of magnitude of the mean square errors of the fitted surfaces after LM. Moreover, the computational time is around 4 times faster when using neural network initialization, see Table 4.

	Degree 2	Degree 3	Degree 4
Projected Parameters	23s	57s	98s
Neural Network Parameters	11s	14s	14s

Table 4: Computational time in seconds for 1000 high curvature surfaces with degree 2,3 and 4 when applying LM.

5. Parameterization of general point clouds

In this section, we describe the process of choosing the neighborhoods N_i^ℓ and the weights λ_{ij} for the overdetermined parameterization method described in Section 2.2. First, for an interior point x_{i_1}, we consider the $r-$neighborhood Θ_i. We choose r large enough so that Θ_i contains at least 15 points.

Second, we construct the convex hull of Θ_i and we consider the list of triangles $\mathcal{L} = \{x_{i_{13}} x_{i_{14}} x_{i_{15}}\}$ where the vertices are chosen from the boundary of the convex hull. Afterwards, the list is sorted in decreasing order with respect to the ratio E/e, where E and e are the lengths of the longest and shortest edge of the triangle $x_{i_{13}} x_{i_{14}} x_{i_{15}}$, respectively. Next, we go over the list of triangles $\mathcal{L}$ looking for triangles such that the projection of 12 interior points including the point x_i onto the plane spanned by the triangle lies inside the triangle. We denote the set of indices of the interior points as $\Xi = \{i_1, i_2, \dots i_{12}\}$.

Based on the robustness and good results obtained in the last section for neural network initialization and the Levenberg-Marquardt algorithm, we use this approach to estimate the barycentric coordinates λ_{ij} for $i \in \Xi$ and $j = i_{13}, i_{14}, i_{15}$ of the interior points $\{x_{i_1}, \dots x_{i_{12}}\}$ with respect to the vertices $x_{i_{13}}$, $x_{i_{14}}$ and $x_{i_{15}}$. If the residual of LM is less than a fixed threshold, the following equations are added to the global

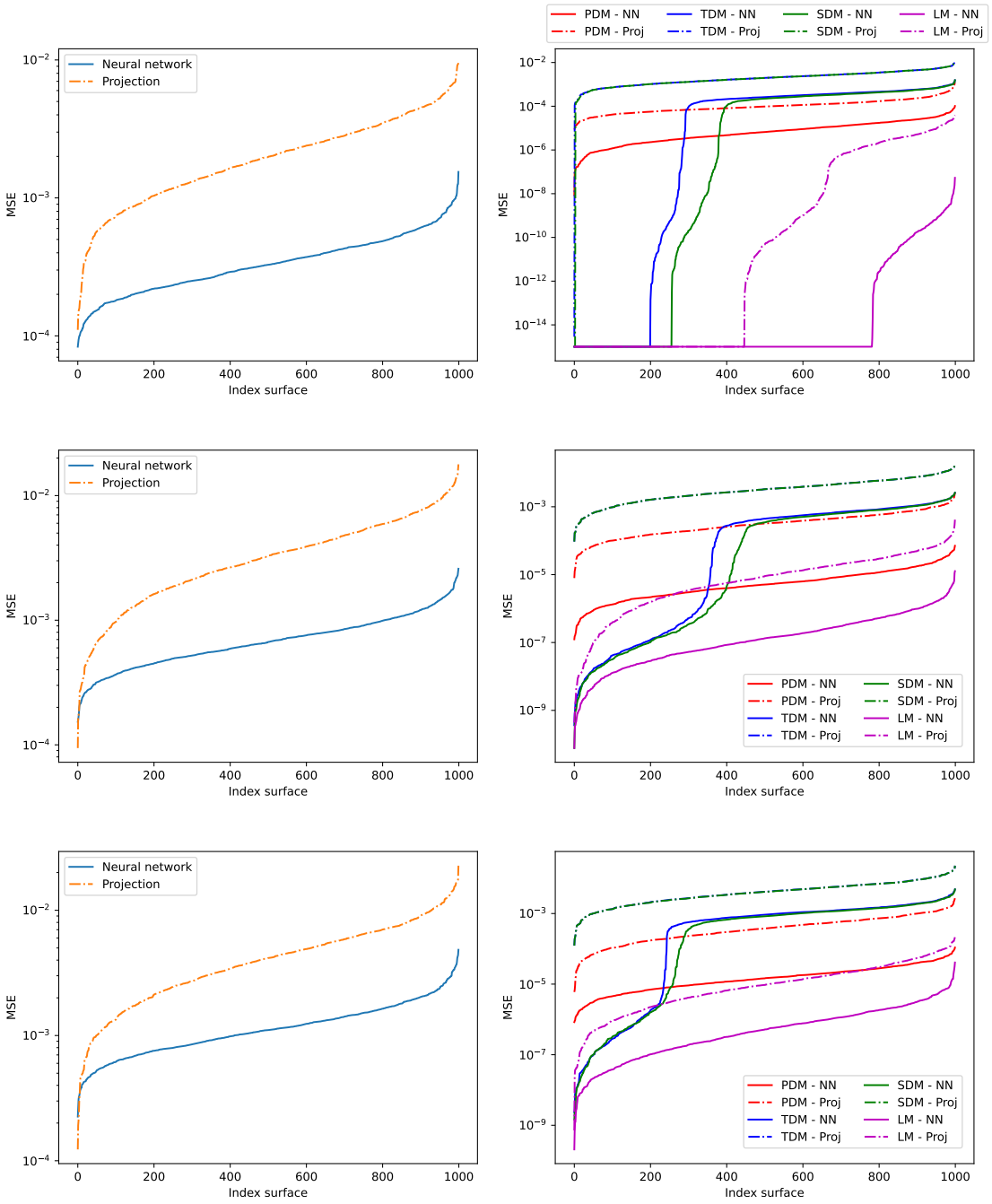


Figure 10: High curvature surfaces of degree 2 (top), 3 (center) and 4 (bottom): MSE of the fitted surfaces after applying PDM, TDM, SDM and LM with initial parameters via projection and neural network evaluation.

system of equations:

$$u_i = \sum_{j \in \{i_{13}, i_{14}, i_{15}\}} \lambda_{ij} u_j$$

for $i \in \Xi$. We repeat these steps until each interior point x_i in the point cloud has reached a minimum number of equations or until each interior point has been visited a prescribed number of times. Moreover, we ensure that the resulting overdetermined system is boundary connected such that Theorem 3 applies and we can solve the system in the least-squares sense.

In the rest of this section, we study the application of this method to data fitting. When the point cloud $X = X_I \cup X_B$ is sampled from a quadratic surface with corresponding parameters $U = U_I \cup U_B$, we aim to reproduce the exact parameters U_I (zero-residual case). If the degree is larger than 2 (non-zero-residual case), then we aim to minimize the mean squared error of the fitting surface.

We compare the results of our method with the three methods (uniform, reciprocal distance and shape-preserving) proposed by Floater and Reimers (2001) in three numerical examples.

Example 5 (Triangular domain - zero residual case)**.** We sample 500 interior points x_i for $i = 1, \ldots, 500$ from a quadratic triangular Bézier surface with corresponding barycentric coordinates $u_i, v_i, w_i > 0.03$ for $i = 1, \ldots, 500$. We also sample 75 boundary points x_i with corresponding parameters $u_i \in D$ for $i = 501, \ldots, 575$, see Figure 11.

To generate the surface we use the same method described in Section 3.3.2. To this end, we fix the control points b_{200}, b_{020} and b_{002} as

$$b_{200} = [0, -1, 0], \quad b_{020} = [-1, 3, 2], \quad b_{002} = [2, 2, 3].$$

The remaining control points b_{110}, b_{101} and b_{011} are chosen randomly in a box $[-0.4, 0.4] \times [-0.4, 0.4] \times [-3, 3]$ around the midpoints of the triangle defined by b_{200}, b_{020} and b_{002}, obtaining

$$b_{110} = [0.81, 0.87, 2.93], \quad b_{101} = [2.13, 0.19, 1.07], \quad b_{011} = [1.02, 2.44, 0.51].$$

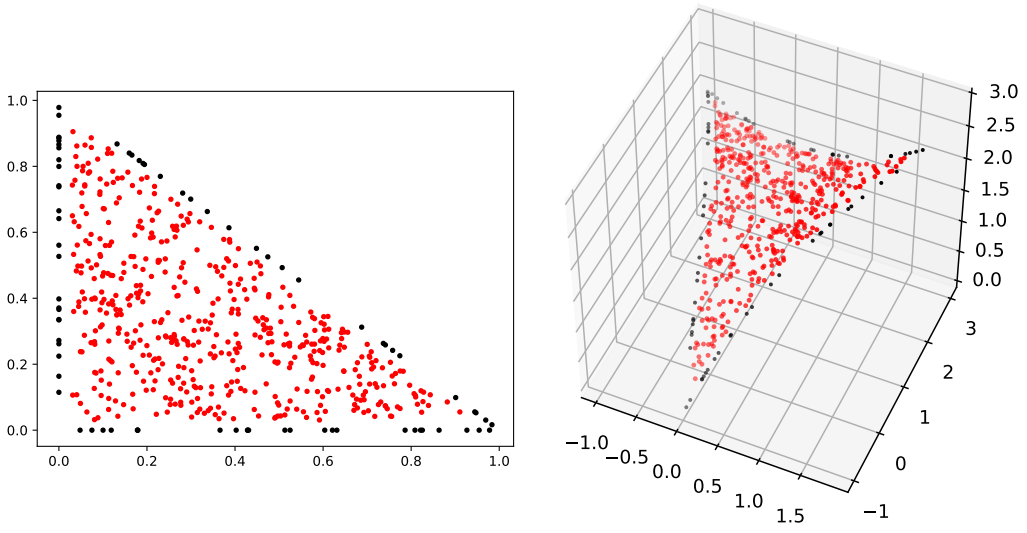


Figure 11: Triangular domain - zero residual case: Parameter values (left) and point cloud (right).

We perform a comparison with the three methods proposed by Floater and Reimers (2001), meaning the uniform, reciprocal distance and shape-preserving parameterizations. Since these methods do not give a specific way of choosing the radius r, we consider different values of the radius $r = 0.2, 0.4, 0.6, 0.8, 1$. Table

5 reports the mean squared error of the estimated parameters with respect to the original parameterization for each method and radius. We select for the final fitting the parameters corresponding with the smallest mean squared error.

Radius r	0.2	0.4	0.6	0.8	1
MSE uniform	$4.33e{-3}$	$7.79e{-3}$	$1.63e{-2}$	$2.64e{-2}$	$3.73e{-2}$
MSE reciprocal distance	$4.61e{-3}$	$6.77e{-3}$	$1.33e{-2}$	$2.10e{-2}$	$2.87e{-2}$
MSE shape-preserving	$5.11e{-4}$	$7.37e{-5}$	$7.48e{-5}$	$9.91e{-5}$	$8.65e{-5}$

Table 5: Triangular domain - zero residual case: MSE of the parameters for different choices of the radius r using Floater and Reimers' methods.

For our method we chose the threshold 10^{-20} for the local parameterizations and we go over the list of interior points until each point has at least one equation. This results in a system of equations with dimension 970×500 from 82 local parameterizations. Representations of the local parameterizations and the associated graph to the system of equations are presented in Figure 12. We note that all the interior points are boundary connected since they all belong to the maximally connected component. This implies the uniqueness of the solution and moreover, the estimated parameters are contained in the parameter domain.

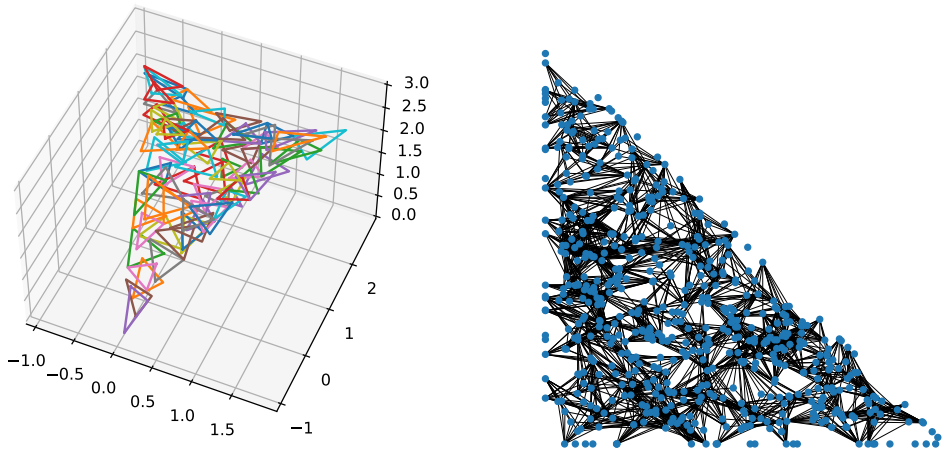


Figure 12: Triangular domain - zero residual case: Local parameterizations (left). Associated graph to the system of equations (right).

For all four methods we use the estimated parameters to fit quadratic triangular Bézier surfaces. The MSE of the estimated parameters and the fitted surfaces are reported in Table 6. Clearly, our global parameterization algorithm estimates the exact parameters, while the methods proposed by Floater and Reimers (2001) only estimate approximations of the parameters. This exact reproduction of the parameters for a quadratic surface gives a significant advantage when using our new approach compared to the standard methods, since we gain approximation power. A significant improvement of the shape-preserving method with respect to the uniform and reciprocal distance weights is observed.

	our method	uniform	reciprocal distance	shape-preserving
MSE parameters	$2.09e{-24}$	$4.33e{-3}$	$4.61e{-3}$	$7.37e{-5}$
MSE fitting	$3.74e{-24}$	$1.01e{-2}$	$1.06e{-2}$	$5.56e{-5}$

Table 6: Triangular domain - zero residual case: MSE of the parameters and MSE of the fitted surface for all the methods.

Example 6 (Quadrilateral domain - zero residual case)**.** We revisit the previous example and apply extrapolation to the quadratic triangular Bézier surface. Then, we sample 500 interior points and 100 boundary points over its rectangular domain $[0, 1] \times [0, 1]$, see Figure 13.

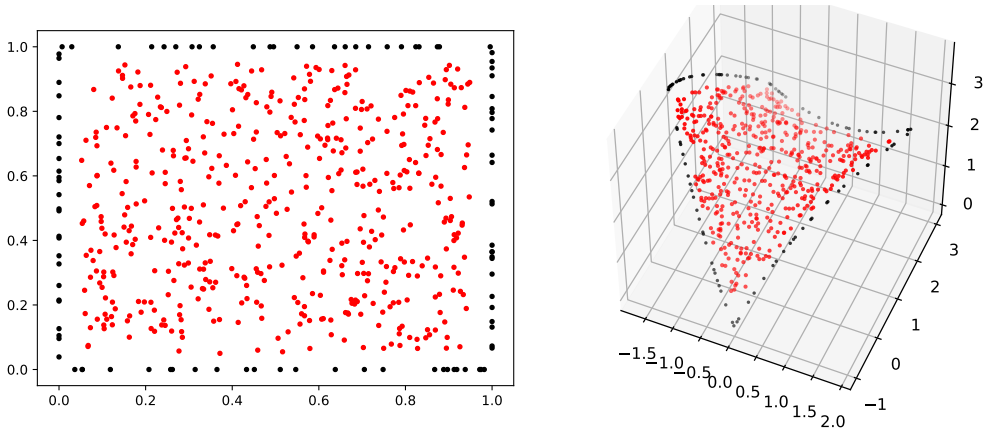


Figure 13: Quadrilateral domain - zero residual case: Parameter values (left) and point cloud (right).

Again, we consider different choices of the radius $r = 0.2, 0.4, 0.6, 0.8, 1$ and we compute the mean squared error of the estimated parameters with respect to the original parameterization. Once again we keep the parameters corresponding with the smallest mean squared error for each method.

Next, we use our global parameterization method with the same threshold and ensuring at least one equation per point. This results in a system of equations of dimension 987×500 from 83 local parameterizations. Representations of the local parameterizations and the associated graph to the system of equations are presented in Figure 14. Like in the previous example we notice that all the interior points are boundary connected and again the obtained parameters were contained inside the parameter domain.

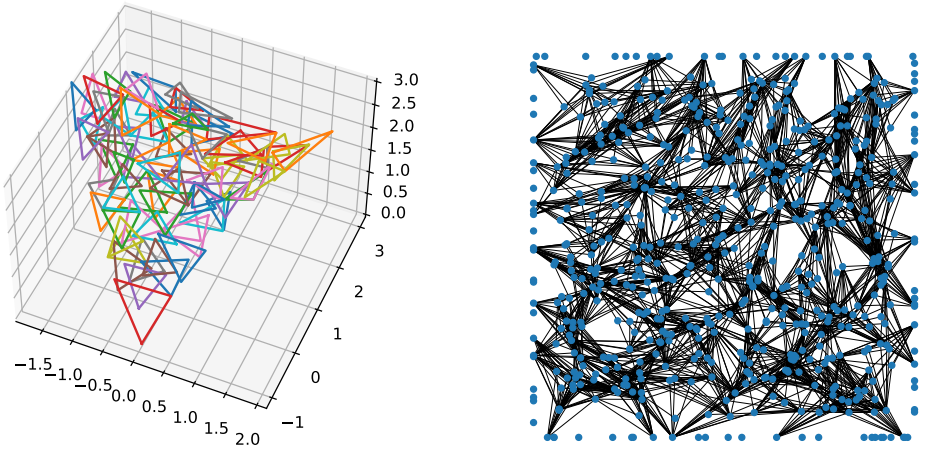


Figure 14: Quadrilateral domain - zero residual case: Local parameterizations (left). Associated graph to the system of equations (right).

Table 7 reports the MSE for both the parameters and the fitting surfaces for each method. We notice that our method estimates the exact parameters while Floaters methods only gives an approximation. More specifically, a significant improvement of the shape-preserving method with respect to the uniform and

reciprocal distance weights is observed, whereas our method outperforms all of the standard methods and it is able to recover the exact parameters.

	our method	uniform	reciprocal distance	shape-preserving
MSE parameters	$6.70e{-25}$	$1.44e{-2}$	$1.50e{-2}$	$1.11e{-3}$
MSE fitting	$1.50e{-24}$	$2.80e{-2}$	$2.30e{-2}$	$9.79e{-4}$

Table 7: Quadrilateral domain - zero residual case: MSE of the parameters and MSE of the fitted surface for all the methods.

Example 7 (High curvature - Set of bi-quadratic tensor-product Bézier surfaces)**.** In this example we would like to investigate the performance of our method for a specific class of surfaces. To this end, we generate 50 hyperbolic bi-quadratic tensor-product Bézier surfaces with high curvature. To do so, we create the surfaces in analogous way described in Section 3.3.2 by fixing $c_{00} = [0, 0, 0]$, $c_{20} = [1, 0, 0]$ and $c_{02} = [0, 1, 0]$ and choosing the remaining six control points randomly inside boxes of dimension $[-0.4, 0.4] \times [-0.4, 0.4] \times [2, 5]$ around the vertices of a regular 3×3 grid over $[0, 1]^2 \times \{0\}$.

In Figure 15 we report the MSE of the parameters and the MSE of the fitted surfaces. We observe a significant improvement in the parameterization obtained from our global parameterization algorithm with respect to the methods proposed by Floater and Reimers (2001), resulting in approximating surfaces with lower fitting error. This confirms that our method possesses a clear advantage over the standard methods from Floater and Reimers (2001).

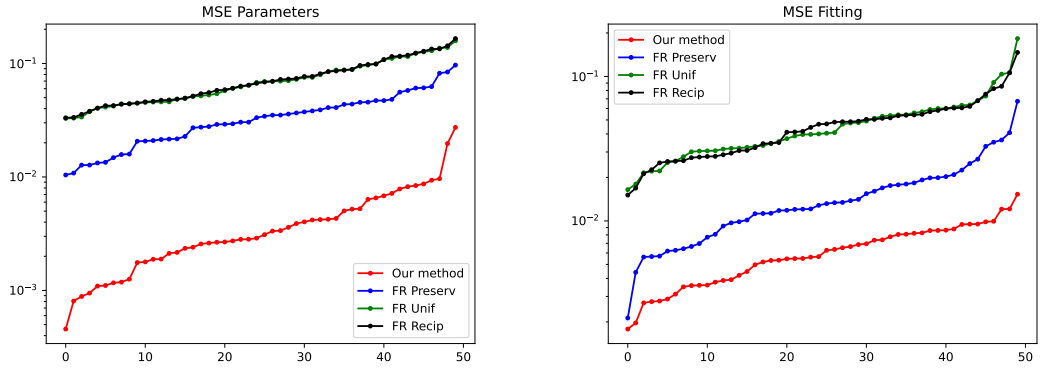


Figure 15: High curvature - Set of bi-quadratic tensor-product Bézier surfaces: MSE of the parameters (left) and MSE of the fitted surfaces (right). The label FR refers to the methods of Floater and Reimers (2001).

6. Conclusion

In this paper, we proposed an overdetermined generalization of a standard parameterization algorithm for unstructured point clouds. Moreover, in order to generate suitable local convex combinations, we proposed training a neural network for the parameterization of a point cloud of fixed size by barycentric coordinates with respect to three of them. We showed that in all considered cases the output of the neural network results in better fitting errors when fitting triangular Bézier surfaces, compared to the standard parameterization via projection. In particular for surfaces with high curvature, the neural network performs very robustly while projection leads to suboptimal results. We then studied the use of this parameterization as an initialization of non-linear methods and concluded that we can recover the original parameterization in the zero-residual case and find good parameters for fitting quadratic surfaces to data points in the non zero-residual case. Furthermore, we compared non-linear optimization for surface fitting using the standard methods point distance minimization (PDM), tangent distance minimization (TDM) and squared distance mininmization and the Levenberg-Marquardt algorithm (LM) and observed that LM outperforms the other methods in all

cases. Finally, we showed that good local parameterizations for small subsets of a large point cloud lead to good global parameterizations of the entire point cloud when they are used to generate equations for our overdetermined parameterization algorithm. In future work we plan to extend the method to higher polynomial degrees as well to surface approximations with triangular and tensor-product splines.

Acknowledgments

This work was supported by the Linz Institute of Technology (LIT) and the government of Upper Austria through the project LIT-2019-8-SEE-116.

References

Barhak, J., Fischer, A., 2001. Parameterization and reconstruction from 3D scattered points based on neural network and pde techniques. IEEE Transactions on Visualization and Computer Graphics 7, 1–16.

Chen, D., Hu, F., Nian, G., Yang, T., 2020. Deep residual learning for nonlinear regression. Entropy 22, 193.

Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., Stuetzle, W., 1995. Multiresolution analysis of arbitrary meshes, in: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, pp. 173–182.

Floater, M.S., 1997. Parametrization and smooth approximation of surface triangulations. Computer aided geometric design 14, 231–250.

Floater, M.S., Reimers, M., 2001. Meshless parameterization and surface reconstruction. Computer Aided Geometric Design 18, 77–92.

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

Greiner, G., Hormann, K., 1997. Interpolating and approximating scattered 3D-data with hierarchical tensor product B-splines, in: Surface Fitting and Multiresolution Methods, Vanderbilt University Press, Nashville. p. 163–172.

Groiss, L., Jüttler, B., Mokriš, D., 2021. 27 variants of Tutte's theorem for plane near-triangulations and an application to periodic spline surface fitting. Computer Aided Geometric Design 85, 101975.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778.

Hormann, K., Greiner, G., 2000. MIPS: An efficient global parametrization method, in: Laurent, P.J., Sablonnière, P., Schumaker, L.L. (Eds.), Curve and Surface Design: Saint-Malo 1999. Vanderbilt University Press, Nashville. Innovations in Applied Mathematics, pp. 153–162.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

Laube, P., Franz, M.O., Umlauf, G., 2018. Deep learning parametrization for B-spline curve approximation, in: 2018 International Conference on 3D Vision (3DV), IEEE. pp. 691–699.

Paszke, A., et al., 2019. Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024–8035.

Rios, D., Jüttler, B., 2022. LSPIA,, (stochastic) gradient descent, and parameter correction. Journal of Computational and Applied Mathematics 406, 113921.

Scholz, F., Jüttler, B., 2021. Parameterization for polynomial curve approximation via residual deep neural networks. Computer Aided Geometric Design 85, 101977.

Shivakumar, P., Chew, K.H., 1974. A sufficient condition for nonvanishing of determinants. Proceedings of the American mathematical society , 63–66.

Tutte, W.T., 1963. How to draw a graph. Proc. London Math. Soc. 3, 743–767.

Wang, W., Pottmann, H., Liu, Y., 2006. Fitting B-spline curves to point clouds by curvature-based squared distance minimization. ACM Transactions on Graphics (ToG) 25, 214–238.

Zheng, W., Bo, P., Liu, Y., Wang, W., 2012. Fast B-spline curve fitting by L-BFGS. Computer Aided Geometric Design 29, 448–462.

Zhu, Z., Iglesias, A., You, L., Zhang, J.J., 2022. A review of 3D point clouds parameterization methods, in: International Conference on Computational Science, Springer. pp. 690–703.