Deep Learning Parametrization for B-Spline Curve Approximation

Pascal Laube* Matthias O. Franz* Georg Umlauf*
*Institute for Optical Systems, University of Applied Sciences Konstanz, Germany
pascal.laube@gmail.com

Abstract

In this paper we present a method using deep learning to compute parametrizations for B-spline curve approximation. Existing methods consider the computation of parametric values and a knot vector as separate problems. We propose to train interdependent deep neural networks to predict parametric values and knots. We show that it is possible to include B-spline curve approximation directly into the neural network architecture. The resulting parametrizations yield tight approximations and are able to outperform state-of-the-art methods.

1. Introduction

In the parametrization of B-spline curve approximation, both parametric values and a suitable knot vector have to be computed. Since the approximation quality strongly depends on the parametrization this is a key problem for many applications. However, finding good parametrizations is a complex task and computationally difficult. A good parametrization may be defined as the one leading to minimal deviation between data points and the approximating curve. Typically, an error threshold needs to be satisfied. In addition, the approximation should result in as few control points as possible. Thus, the number of knots has to stay small.

Usually, recovering a point parametrization and computing a suitable knot vector are regarded as separate problems. Point parametrization often is only analyzed in terms of interpolation which leads to an implicit knot vector, whereas in the case of approximation, the knot vector usually is regarded as fixed. Other methods focus on knot vector computation while choosing a well-known point parametrization in a preprocessing step.

We propose a method that is able to simultaneously predict parametric values and knots using two interdependent deep neural networks (DNN): (1) a Point Parametrization Network (PPN) which assigns parametric values to point sequences; (2) a Knot Selection Network (KSN) which predicts new knot values for knot vector refinement. Our neural networks directly operate on point data without the need for computing intermediate features. We will show that it is possible to include B-spline curve approximation directly into the network architecture. When compared to state-of-the-art methods, our parametrization leads to smaller approximation errors. To our knowledge, we are the first to show the potential of neural networks for B-spline curve approximation.

2. Related works

Most knot placement methods require prior computation of parametric values. Besides classic methods like the uniform parametrization and chord-length parametrization the most prominent one is the centripetal-method [11]. Another well-known approach is the universal method proposed by Lim [13] which leads to affine invariance and is closely related to the uniform method. The hybrid method by Shamsuddin et al. [24] is a mixture of the chord-length and centripetal method which leads to slightly higher accuracies. Knot placement is usually an iterative process of inserting new knots until satisfaction of an error bound. In [12] a heuristic rule based on an angular measure is used to determine suitable knot values. An extensive analysis of the impact of different geometric features for knot vector computation is given by Razdan [22]. Piegl and Tiller [19] average parametric values to generate the knot vectors. A well-known refinement based method was introduced by Park and Lee [18] where the knot vector relies on the computation of dominant points which are points of special interest (e.g. high curvature). A machine learning approach using support vector machines for knot placement is described in [10] which produces approximations with slightly higher error rates than the method by Park and Lee [18]. Other methods use genetic algorithms for knot vector optimization [30, 29, 28] or meta-heuristics like a firefly algorithm driven approach [4].

Machine learning has become an important field in geometry processing and has been successfully applied to different problems regarding geometric modeling. Steinke et al. [26] use support vector regression for head reconstruction, outlier removal and hole filling. Lin et al. [14] propose DNNs for surface reconstruction based on 2D images. In [21] Qi et al. showed that DNNs can classify or segment point data without an intermediate representation. Another important research area is the application of machine learning in non-euclidean space, e.g., for shape-matching [17, 1] or shape-completion [15].

3. Preliminaries

Due to their design-properties, non-uniform B-spline curves are the standard curve representation in Computer Aided Design. Their application ranges from describing trajectories to solid modeling in the from of boundary representation models. Fitting B-spline curves to point data is a common task. B-spline curves are parametric curves, defined by linear combinations of basis functions, whose support is specified by a knot vector $\mathbf{u}$. Since they are parametric, sampled points $\mathbf{p}$ are accompanied by parametric values $\mathbf{t}$. As a consequence, approximation requires a set of parametric values as well as a knot vector. Since B-Spline curves are computed coordinate-wise they can represent any n-dimensional curve. While the following sections focus on 2D B-spline curves our approach is applicable to any n-dimensional curve. Following we give a short introduction to B-spline curve approximation and deep neural networks.

3.1. B-spline curve approximation

Suppose we have given a sequence of points $\mathbf{p} = (p_0, ..., p_m)$ with each point represented by coordinates $p_i = (x_i, y_i)$. Consider a B-spline curve

$$C(u) = \sum_{j=0}^{n} c_j \, N_j^k(u)$$

of degree k with control points c_j, B-spline functions $N_i^k(u)$, and a non-decreasing knot vector $\mathbf{u} = (u_0, \ldots, u_n)$ where the knots u_0 and u_n have multiplicity $k+1$ for end point interpolation. To compute the control points c_j of the B-spline curve C approximating $\mathbf{p}$, the least squares problem

$$\sum_{i=0}^{m} |p_i - C(t_i)|^2 \to \min$$

with precomputed parameters t_i, $i = 0, \ldots, m$ combined in the *parameter vector* $\mathbf{t} = (t_0, \ldots, t_m)$ and end points $C(t_0) = c_0 = p_0$ and $C(t_m) = c_n = p_m$ is solved. This yields the normal equation

$$(\mathbf{N}^T\mathbf{N})\mathbf{c} = \mathbf{q} \tag{1}$$

where $\mathbf{N}$ is the $(m-1) \times (n-1)$ matrix

$$\mathbf{N} = \begin{pmatrix} N_1^k(t_1) & \ldots & N_{n-1}^k(t_1) \\ \vdots & \ddots & \vdots \\ N_1^k(t_{m-1}) & \ldots & N_{n-1}^k(t_{m-1}) \end{pmatrix},$$

and $\mathbf{c}$ and $\mathbf{q}$ are the vectors defined as

$$\mathbf{c} = \begin{pmatrix} c_1 \\ \vdots \\ c_{n-1} \end{pmatrix}, \mathbf{q} = \begin{pmatrix} \sum_{i=1}^{m-1} N_1^k(t_i)q_i \\ \vdots \\ \sum_{i=1}^{m-1} N_{n-1}^k(t_i)q_i \end{pmatrix}$$

and

$$q_i = p_i - N_0^k(t_i)p_0 - N_n^k(t_i)p_m$$

for $i = 1, ..., m-1$. If there are no constraints for end point interpolation (1) reduces to

$$(\mathbf{N}^T\mathbf{N})\mathbf{c} = \mathbf{N}^T\mathbf{p}. \tag{2}$$

The control points c_j can be computed using (1), if

$$\sum_{l=1}^{m-1} N_i^k(t_l) N_j^k(t_l) \neq 0. \tag{3}$$

This is equivalent to the existence of a parameter $t_i \in [u_j, u_{j+1}]$ for $j = k, ..., n+1$, see e.g. [3]. For our experiments, we used $k = 3$.

3.2. Deep neural networks

In this work, we apply feed-forward DNNs to learning the parameter vector $\mathbf{t}$ for points $\mathbf{p}$ and a knot vector $\mathbf{u}$ to get tight B-spline curve approximations. DNNs are organized in layers of uniformly behaving artificial neurons. Some of these layers are trainable, i.e., their behavior is controlled by adjustable weights. Here, we use the classical multilayer perceptron (MLP) architecture (Figure 2) where each neuron is connected via weights to all neurons of the previous layer. Each neuron computes a weighted sum of its inputs and feeds it through a static nonlinearity as its output which again is connected to all neurons in the next layer. The weights are adapted during the training phase by gradient descent on a loss function that measures the performance of the DNN (Section 5.3). The reader is referred to Goodfellow et al. [6] for an in-depth discussion of deep learning.

In order to apply DNNs to our problem of parametrization, we have to address the following challenges:

CL1. Since there are no publicly available datasets for this problem we have to synthesize a sufficiently large training data set.

CL2. It must be ensured that the training data and the real data share the same characteristics.

CL3. A suitable loss function for a parametrization of a B-spline approximation must be defined.

CL4. Due to its architecture, a MLP requires input of a fixed size whereas point sequences for approximation are of variable size. The approach must be able to cope with this problem.

4. Learning Point Parametrization and Knot Placement

In this section we will describe our method for point parametrization and knot placement for arbitrary 2D point sequences $\mathbf{p}$. Since approximation itself requires point sets of known sequential order we consider $\mathbf{p}$ to satisfy this constraint. In our exposition we follow Figure 1 which gives an overview of the parametrization process.

P1. Segmentation The input point sequence $\mathbf{p}$ has a complexity given by its total curvature

$$\widehat{\kappa}(\mathbf{p}) = \sum_{i=0}^{m-1} \frac{(|\kappa_i| + |\kappa_{i+1}|)\|p_{i+1} - p_i\|_2}{2},$$

where κ_i is the curvature at point p_i. Throughout this work curvature is computed using osculating circles [27]. Using $\widehat{\kappa}(\mathbf{p})$ as a measure of complexity we can quantify the maximum complexity of the training data set and by that the maximum complexity the DNN is able to process. To handle CL2. we propose to split point sequences so that the resulting segments represent the complexity of the training data. For a point sequence $\mathbf{p}$ compute the total curvature $\widehat{c}(\mathbf{p})$ and split $\mathbf{p}$ into point sequence segments $\mathbf{p}^s, s = 1, ..., r$, at the median, if $\widehat{\kappa}(\mathbf{p}) > \widehat{\kappa}_t$ for a threshold $\widehat{\kappa}_t$. We set $\widehat{\kappa}_t$ to the 98th percentile of $\widehat{\kappa}(\cdot)$ of the training set (Section 5.3). We repeat this segmentation process $r-1$ times until each segment $\mathbf{p}^s$ satisfies $\widehat{\kappa}(\mathbf{p}^s) < \widehat{\kappa}_t$.

P2. Sub-/supersampling and normalization To process $\mathbf{p}^s, s = 1, ..., r$, by the PPN/KSN the number of points per segment has to match the DNN input size l (CL4.). Thus, the $\mathbf{p}^s$ are sub- or supersampled:

Subsampling: If the number of points in $\mathbf{p}^s$ is larger than l, draw points from $\mathbf{p}^s$ such that the drawn indices i are equally distributed and include the first and last point.

Supersampling: If the number of points in $\mathbf{p}^s$ is smaller than l, we linearly interpolate temporary points between consecutive points p_i^s and p_{i+1}^s from left to right. This is iterated until the number of points equals l. These temporary points are not permanent members of $\mathbf{p}$. They only exist to matching the network input size.

The sub-/supersampled segments are then normalized to $\bar{\mathbf{p}}^s$ consisting of the points

$$\bar{p}_i^s = \frac{p_i^s - \min(\mathbf{p}^s)}{\max(\mathbf{p}^s) - \min(\mathbf{p}^s)},$$

where $\min(\mathbf{p}^s)$ and $\max(\mathbf{p}^s)$ are the minimum and maximum coordinates of $\mathbf{p}^s$.

P3. Parametrization For each $\bar{\mathbf{p}}^s$ the PPN (see Section 5.1) generates a parametrization $\bar{\mathbf{t}}^s \subset [0, 1]$. This parametrization is rescaled to $[u_{s-1}, u_s]$ and adapted to the sub-/supersampling of $\bar{\mathbf{p}}^s$, yielding $\mathbf{t}^s$. For a point p_i that was removed from $\mathbf{p}^s$ in the subsampling, insert to $\bar{\mathbf{t}}^s$ a parameter

$$t_i = t_\alpha^s + (t_\omega^s - t_\alpha^s)\frac{\text{chordlen}(p_\alpha^s, p_i)}{\text{chordlen}(p_\alpha^s, p_\omega^s)},$$

where chordlen is the length of the polygon defined by a point sequence and p_α^s and p_β^s are the closest neighbors of p_i to the left and right in the subsampled segment with parameters t_α^s and t_β^s. Parameters t_i^s corresponding to temporary points are simply removed from $\bar{\mathbf{t}}^s$.

For the initialization of the parametrization step, an initial knot vector is required. First define $u_0 = 0$ and $u_n = 1$. Then, for each segment (except the last one), one knot u_i is added:

$$u_i = u_{i-1} + \frac{\text{chordlen}(\mathbf{p}^s)}{\text{chordlen}(\mathbf{p})}, \quad i = 1, ..., r-1.$$

This yields a start- and end-knot for every point sequence segment.

P4. Refinement In the refinement step, additional knots are added to point sequence segments with large approximation error. We determine the segment $\mathbf{p}^s$ with largest Hausdorff distance to the input data $\mathbf{p}$. For $\bar{\mathbf{p}}^s$ and $\bar{\mathbf{t}}^s$ the KSN (see Section 5.2) generates a new estimated knot $\bar{u}_s \in [0, 1]$. The new knot $\bar{u}_s$ is mapped to the actual knot value range $[u_{s-1}, u_s]$ by

$$\widetilde{u}_s = u_{s-1} + \bar{u}_s(u_s - u_{s-1}). \tag{4}$$

Then, instead of $\widetilde{u}_s$, the parameter value t_i closest to $\widetilde{u}_s$ is inserted to $\mathbf{u}$. Since the KSN operates on sub-/supersampled data, this correction of $\widetilde{u}_s$ is the simplest choice to ensure (3).

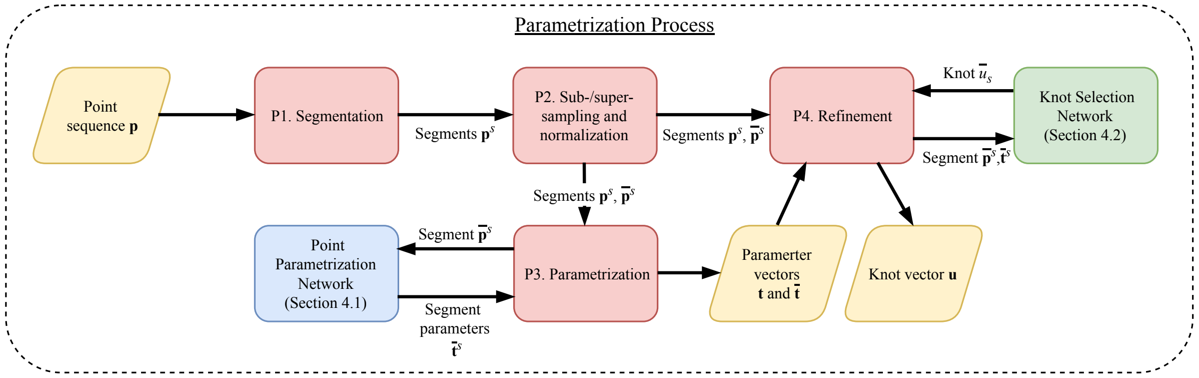
Parametrization Process
Point sequence **p**
P1. Segmentation
Segments $\mathbf{p}^s$
P2. Sub-/super-sampling and normalization
Segments $\mathbf{p}^s$, $\bar{\mathbf{p}}^s$
P4. Refinement
Knot $\tilde{u}_s$
Knot Selection Network (Section 4.2)
Segment $\bar{\mathbf{p}}^s$, $\bar{\mathbf{t}}^s$
Segments $\mathbf{p}^s$, $\bar{\mathbf{p}}^s$
Segment $\bar{\mathbf{p}}^s$
Point Parametrization Network (Section 4.1)
P3. Parametrization
Segment parameters $\bar{\mathbf{t}}^s$
Paramerter vectors **t** and $\bar{\mathbf{t}}$
Knot vector **u**

Figure 1: Overview of the parametrization process. The inputs/outputs are marked in yellow. Red boxes refer to sub-processes described successively in Section 4. The other colors refer to the sub-processes in Figure 2.

Remark 1 *Note, that $\tilde{u}_s$ could be inserted to* **u** *directly, as long as (3) is satisfied.*

We propose to further refine **u** until the desired curve approximation error threshold is satisfied.

5. DNN architectures

In this section we describe the deep neural network architectures for point parametrization (PPN) and knot selection (KSN) and their training. Figure 2 shows the network architectures of these networks. There exists an infinite number of parametrizations that will lead to identical approximation results. This absence of ground truth is the main reason for making approximation an active part of the neural network architecture. Since the networks take point sequence segments $\bar{\mathbf{p}}^s$ as input we will drop the upper index s and the over-bar for all variables in Sections 5.1 and 5.2 to simplify notation.

5.1. Point Parametrization Network

For a sequence of points $\mathbf{p}$ a parameter vector $\mathbf{t} = (t_i)_i$ is defined as $t_i = t_{i-1} + \Delta_{i-1}$. For classical methods Δ_i is computed based on geometric properties of $\mathbf{p}$, e.g. the centripetal parametrization $\|p_{i+1} - p_i\|^{\frac{1}{2}}$. We propose to estimate Δ_i using a pre-trained neural network called the Point Parametrization Network. Similarly, the input to the PPN consists of segments $\mathbf{p}$. It can be written in the form $\mathbf{p} = (x_0, ..., x_{l-1}, y_0, ..., y_{l-1})$, where the x and y are the coordinates of the points of $\mathbf{p}$. The parameter domain is defined as $u_0 = t_0 = 0$ and $u_n = t_{l-1} = 1$. Then, the task of the PPN is to predict missing values $\Delta = (\Delta_0, ..., \Delta_{l-2})$ with

$$\Delta_i > 0, i = 0, \dots, l-2, \tag{5}$$

such that $t_0 < t_1$ and $t_{l-2} < t_{l-1}$. We apply a MLP to the input data $\mathbf{p}$, yielding as output a distribution for parametrization $\Delta^{mlp} = (\Delta_0^{mlp}, ..., \Delta_{l-2}^{mlp})$ of size $l-1$.

PP1. Accumulation & Rescaling The output Δ^{mlp} is used to compute a parameter vector $\mathbf{t}^{mlp}$ with $t_0^{mlp} = 0$ and

$$t_i^{mlp} = \sum_{j=0}^{i-1} \Delta_j^{mlp}, i = 1, \dots, l-1.$$

Since, t_{l-1}^{mlp} is usually not 1, rescaling of $\mathbf{t}^{mlp}$ yields the final parameter vector $\mathbf{t}$ with

$$t_i = t_i^{mlp} / \max(\mathbf{t}^{mlp}).$$

To ensure (5), the MLP output Δ^{mlp} must be positive. This is achieved by using the softplus activation function

$$f(x) = \ln(1 + e^x)$$

for neurons of the MLP in the PPN.

PP2. Approximation To be able to define a network loss (CL3.) we include the B-spline curve approximation directly as a network layer. The input points $\mathbf{p}$ and their parameters $\mathbf{t}$ are used for an approximation with knot vector $\mathbf{u} = (0, 0, 0, 0, 1, 1, 1, 1)$ for $k = 3$. Since the PPN parametrizes curve segments and not the complete curve we approximate without endpoint interpolation (2). The approximation layer's output $\mathbf{p}^{app} = (p_0^{app}, \dots, p_{l-1}^{app})$ is the approximating B-spline curve evaluated at $\mathbf{t}$.

PP3. Euclidean Loss The loss for the PPN is

$$\frac{1}{l} \sum_{i=0}^{l-1} \|p_i - p_i^{app}\|_2. \tag{6}$$

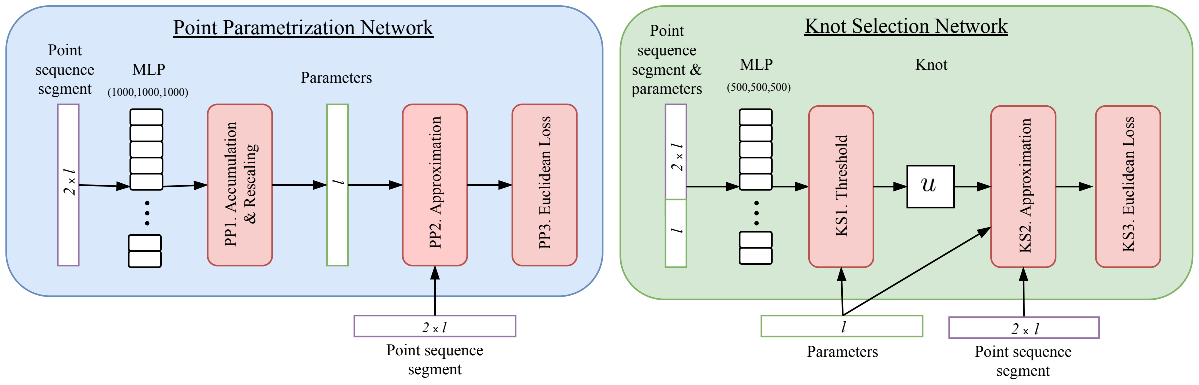

Figure 2: Network architectures for the Point Parametrization Network (PPN, left) and the Knot Selection Network (KSN, right). Red boxes refer to sub-processes described successively in Sections 5.1 and 5.2.

5.2. Knot Selection Network

The KSN predicts a new knot u to the interval $(0, 1)$ for a given segment $\mathbf{p}$ and parameters $\mathbf{t}$ (predicted by the PPN). Thus, the resulting network input size is $3l$.

We apply a MLP, which transforms the input to a single output value u^{mlp}. As network activation functions we use the RELU function [5] except for the output layer, where we use the Sigmoid function [7].

KS1. Threshold Layer The new knot u has to satisfy $u \in (0, 1)$, and $\mathbf{t} \cap [0, u] \neq \emptyset$ and $\mathbf{t} \cap [u, 1] \neq \emptyset$, to ensure (3). To satisfy these constraints, we use a threshold layer which maps u^{mlp} to

$$u = \begin{cases} \varepsilon & , \text{if } u^{mlp} \leq 0 \\ 1 - \varepsilon & , \text{if } u^{mlp} \geq 1 \\ u^{mlp} & , \text{otherwise.} \end{cases}$$

By introducing a small $\varepsilon = 1\text{e}{-5}$ we make sure that knot multiplicity at the end-knots stays equal to k. This choice of u corresponds to the more general approach mentioned in Remark 1. Also note that u corresponds to $\bar{u}$ in (4) where the knot is mapped to its actual value range.

KS2. Approximation The approximation in the KSN has the same form as the approximation of the PPN with the exception of the knot vector. Here, the knot vector is $\mathbf{u} = (0, 0, 0, 0, u, 1, 1, 1, 1)$. For backpropagation, the derivative of the B-spline basis functions with respect to u is required, see [20]. As for the PPN, the approximation layer's output $\mathbf{p}^{app} = (p_0^{app}, \dots, p_{l-1}^{app})$ is the approximating B-spline curve evaluated at $\mathbf{t}$.

KS3. Euclidean Loss The loss function for the KSN is the same as for the PPN, see (6).

5.3. Training set generation and training process

For the training of the PPN and the KSN sufficiently large training and test datasets are required. As for the input size we define $l = 100$. An ideal dataset would consist of diverse real-world point clouds, with a known sequential order of points. Since no such datasets are publicly available, we chose to synthesize the data using B-spline curves. We generate random control points c_i using a normal distribution with mean μ and variance σ to define B-spline curves of degree $k = 3$ with $(k + 1)$-fold end-knots and no interior knots. For the y-coordinates, we use $\sigma = 2$ and $\mu = 10$. For the x-coordinates, we use $\sigma = 1$ and $\mu = 10$ for the first control point and increase μ by $\Delta\mu = 1$ for all consecutive control points. Curves with self-intersections are discarded, because the sequential order of their sampled points is not unique and, in reverse engineering, such point sets are usually split into subsets at the self-intersection. Smaller σ for control point x-coordinates reduces the number of curves with self-intersections in the first place. Using this approach, we generate a dataset consisting of 150.000 curves. Then, we sample l points $\mathbf{p} = (p_0, \dots, p_{l-1})$ along each curve. Since these curves tend to have increasing x-coordinates from left to right we add index-flipped versions of the point sequences to the dataset resulting in 300.000 point sequences of which 20% are used as test data in the training process. In our experiments, this method leads to a very diverse set of curves containing sections with very little up to no curvature as well as sections with high curvature and even sharp features. While we use cubic B-spline curves for dataset generation, our approach is not limited to point clouds computed this way.

Since the KSN requires point parametrizations $\mathbf{t}$ as input, the PPN is trained first. After training, the layers PP2 and PP3 are discarded and PP1 becomes the network output layer. We compute parametric values $\mathbf{t}$

for the training dataset by applying the PPN and train the KSN on the combined input. For parametrization in Section 4, the layers KS2 and KS3 are discarded while KS1 becomes the network output layer. The MLPs of the PPN and KSN consist of three hidden layers with sizes $(1000, 1000, 1000)$ and $(500, 500, 500)$. We apply dropout [8] to MLP layers and train the networks using the Adam optimizer [9].

6. Results

In this section we present results of our parametrization method which we call PARNET. First, we discuss results of the point parametrizations computed by the PPN (Section 6.1). Then, we discuss knot selections computed by the KSN as well as the global approximation quality of our approach (Section 6.2).

For the evaluation we generated four evaluation sets:

- *Evaluation set 1* contains 500 curves computed as described in Section 5.3. We sample 500 equidistributed (in terms of arc length) points on each curve.
- *Evaluation set 2* contains the curves from *evaluation set 1* but sampled at random parameters.
- *Evaluation set 3* contains 500 curves computed as described in Section 5.3 but with random interior knots without multiplicities. We generate 3 to 8 random interior knots which results in a set of very diverse curves, some of high complexity. We sample 500 equidistributed points on each curve.
- *Evaluation set 4* contains the curves from *evaluation set 3* but sampled at random parameters.

We included *evaluation set 2* and *evaluation set 4* into our evaluation because many parametrization methods use noise filters before parametrization, e.g. [23]. These filters result in a smooth set of points (or smooth curvature) but also lead to an uneven distribution of points. Training, test and evaluation sets can be downloaded from (http://www.ios.htwg-konstanz.de/parnetdatasets).

6.1. Point Parametrization

In Table 1 we compare point parametrizations computed by the PPN and the uniform, chordal, and centripetal parametrization. For evaluation of curve approximation quality, Hausdorff distance is the de facto standard [25, 2]. We compare the methods for equidistributed as well as randomly sampled points in terms of the average Hausdorff distance over the complete evaluation set. Parametrizations computed by the

	Evaluation set 1	*Evaluation set 2*
PNN	0.0224	0.0992
Uniform	0.2097	0.2095
Chordal	0.2099	0.2001
Centripetal	0.2098	0.2030
PPN	0.0245	0.1088
uniform	0.2040	0.2102
chordal	0.2042	0.1955
centripetal	0.2040	0.2008

Table 1: Average Hausdorff distances for approximation without interior knots of the PPN compared to common parametrization methods.

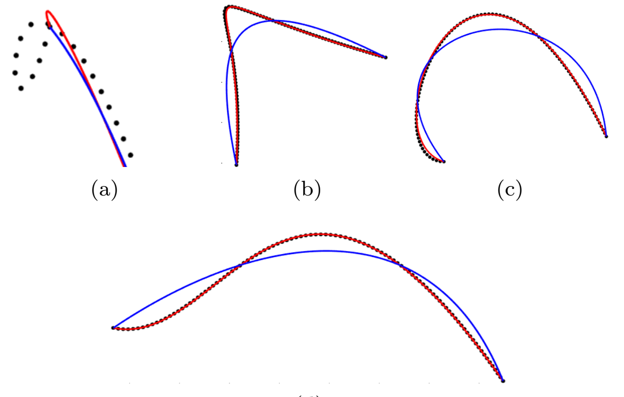

Figure 3: Results of parametrizations for examples of *evaluation set 1* by the centripetal method (blue) and by the PPN (red), approximated without interior knots.

PPN result in approximations with up to eight times smaller Hausdorff distance for *evaluation set 1*. For *evaluation set 2* the results are still two times smaller when compared to the other methods. Most methods for parametrization are based on geometric relations of points. It has been shown that high curvature is a strong indicator for a denser parametrization [16]. In [11], Lee introduces the general exponent method which also includes the centripetal parametrization. Here parameter values are based on changes in curvature. Assuming that regions of higher curvature are sampled more densely, this method fails for equidistributed points as can be seen from results in Table 1 and Figure 3. As can be seen in Figure 3 approximations using our method are able to follow the points very closely. Figure 3a shows a close-up on an example where the PPN results in larger Hausdorff distance when compared to the centripetal method. In Figure 4 we colored curves from *evaluation set 1* according the distribution of the parametrization Δ. It is obvious,

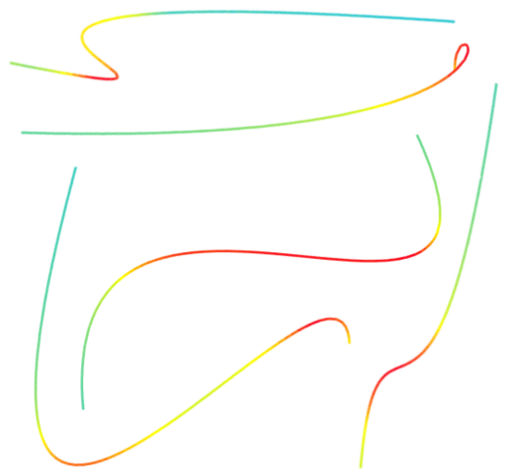

Figure 4: Heatmap-colored curves of *evaluation set 3* colored by parametrization value Δ. Blue corresponds to low values for Δ while red corresponds to large values for Δ.

that a large absolute curvature is a strong indicator for larger values in Δ. But also regions containing inflection points lead to large values in Δ. Since points in *evaluation set 2* are equidistributed, the PPN has learned to incorporate curvature into the parametrization process.

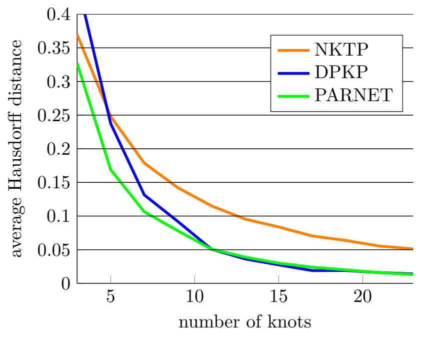

Figure 5: Average Hausdorff distance over *evaluation set 3* at different numbers of knots for PARNET, DPKP and NKTP

6.2. Knot Selection

We evaluate the effectiveness of our approach by comparing PARNET to two other methods for knot placement. One is a the well-known averaging method by Piegl and Tiller [20] which does not incorporate any geometric information in the process of knot placement (NKTP). Since we place knots using refinement we also compare PARNET to a well-known refinement-

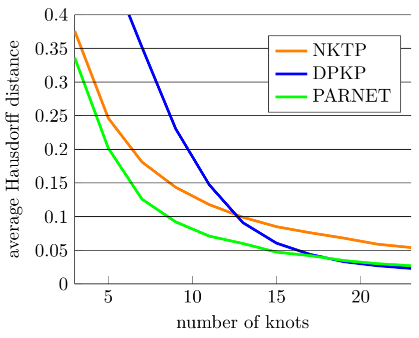

Figure 6: Average Hausdorff distance over *evaluation set 4* at different numbers of knots for PARNET, DPKP and NKTP

based method by Park et al. [18] which uses so-called dominant points for knot placement (DPKP). Again, methods are compared by average Haussdorf distance. Figures 5 and 6 show the results of knot placement in a range from 3 to 23 knots on the *evaluation sets 3* and *4*. On both sets, our method produces approximations of higher quality. Especially with fewer knots in the range from 3 to 12 knots, our method has significantly lower Haussdorf distance. With an increasing number of knots, results of DPKP and our method are very close with a small advantage for our method at 23 knots on *evaluation set 3* and for DPKP on *evaluation set 4*. Figure 7 shows an approximation by the different methods for one example from *evaluation set 3*. While NKTP produces very smooth results it fails in complex regions. The DPKP method is able to approximate regions of high curvature very well but may also lead to wiggles in these regions. Our method is able to approximate highly curved regions while also producing smooth approximations (see the supplementary material for more examples). In Figure 8, we present some curves of *evaluation set 1* and predicted knot positions for refinement by the KSN. Examining knot predictions made by the KSN we make several observations that agree with observations of other authors:

- Curvature plays an important role in knot placement [22, 31]. Regions of high curvature should be favored when placing knots (see Figures 8b, 8c, 8d, and 8e).
- If one has to refine segments with varying curvature directions placing the knot at an inflection point is beneficial [10, 18]. Examples of the KSN choosing to place the knot near an inflection point and not at high curvature regions can be seen in Figures 8f and 8h.
- If curvature is small or changes slowly it is beneficial to split segments so that resulting segments

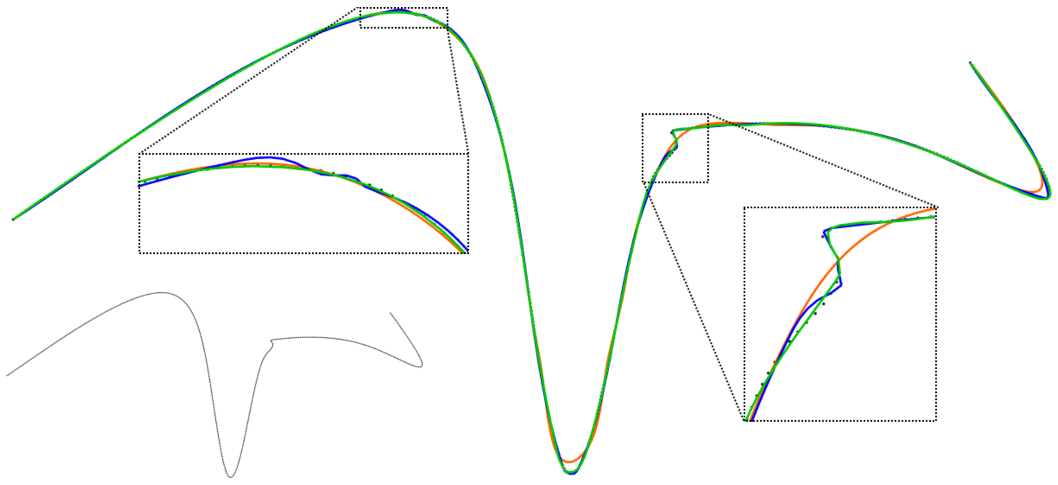

Figure 7: Approximation results with 23 knots for one example of *evaluation set 3* by methods NKTP (orange), DPK (blue) and PARNET (green). The original point sequence is shown in the bottom left corner while the framed boxes contain close-ups of critical regions.

are of equal complexity [18, 22]. This can be seen in Figures 8a and 8g where the KSN splits the point sets close to the median index but makes segments containing higher curvature smaller.

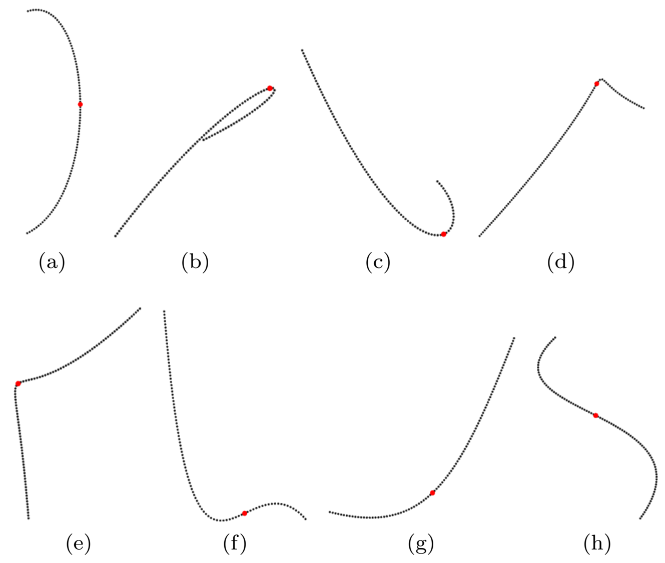

Figure 8: Single knot (red) selected by the KSN on samples of *evaluation set 1*.

7. Discussion

Our experiments show that neural networks are able to successfully predict simultaniously parametric values $\mathbf{t}$ and knots $\mathbf{u}$ for the problem of B-spline curve approximation. Our method results in tight approximations. It works well for unevenly spaced points although we trained on evenly spaced point sequences. This shows that our network is able to generalize well to previously unseen data with versatile characteristics. Since approximation to a certain error threshold requires adding knots in succession we decoupled point parametrization and knot placement in separate networks. When the number of knots is known upfront PPN and KSN could be merged in a single network. One aspect that limits our approach is the synthetic training data set. Real world data would be preferable. Since it is common to retrain networks for the purpose of specialisation our approach can be used as a pre-training method, which may be subsequently improved on additional data. Another drawback of our method is the need for segmentation as well as sub- and supersampling of point sequences together with MLPs of fixed input size. We deliberately chose to segment and sample in a very simple fashion to show that the approximation quality is not a result of preprocessing but of the parametrization by the networks. To be able to process point sequences of arbitrary size we plan to investigate the application of recurrent neural networks instead of simple MLPs. We also would like to apply our approach to surface approximation and other parametric representations like T-splines.
In our approach B-spline curve approximation is directly integrated into the network training loop. We hope that this will enable others to apply neural networks for approximation-related problems.

Acknowledgments

This research is funded by the Federal Ministry of Education and Research (BMBF) of Germany (pn 02P14A035).

References

[1] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 3189–3197, 2016.

[2] X.-D. Chen, W. Ma, and J.-C. Paul. Cubic b-spline curve approximation by curve unclamping. *Computer-Aided Design*, 42(6):523–534, 2010.

[3] G. E. Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann, 2002.

[4] A. Gálvez and A. Iglesias. Firefly algorithm for explicit b-spline curve fitting to data points. *Mathematical Problems in Engineering*, 2013, 2013.

[5] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[6] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[7] J. Han and C. Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, pages 195–201. Springer, 1995.

[8] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[9] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] P. Laube, M. O. Franz, and G. Umlauf. Learnt knot placement in b-spline curve approximation using support vector machines. *Computer Aided Geometric Design*, 2018.

[11] E. T. Lee. Choosing nodes in parametric curve interpolation. *Computer-Aided Design*, 21(6):363–370, 1989.

[12] W. Li, S. Xu, G. Zhao, and L. P. Goh. Adaptive knot placement in b-spline curve approximation. *Computer-Aided Design*, 37(8):791–797, 2005.

[13] C.-G. Lim. A universal parametrization in b-spline curve and surface interpolation. *Computer Aided Geometric Design*, 16(5):407–422, 1999.

[14] C.-T. Lin, W.-C. Cheng, and S.-F. Liang. Neural-network-based adaptive hybrid-reflectance model for 3-d surface reconstruction. *IEEE Transactions on Neural Networks*, 16(6):1601–1615, 2005.

[15] O. Litany, A. Bronstein, M. Bronstein, and A. Makadia. Deformable shape completion with graph convolutional autoencoders. *arXiv preprint arXiv:1712.00268*, 2017.

[16] W. Ma and J.-P. Kruth. Parameterization of randomly measured points for least squares fitting of b-spline curves and surfaces. *Computer-Aided Design*, 27(9):663–675, 1995.

[17] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015.

[18] H. Park and J.-H. Lee. B-spline curve fitting based on adaptive curve refinement using dominant points. *Computer-Aided Design*, 39(6):439–451, 2007.

[19] L. Piegl and W. Tiller. *The NURBS book*. Springer Science & Business Media, 2012.

[20] L. A. Piegl and W. Tiller. Computing the derivative of nurbs with respect to a knot. *Computer aided geometric design*, 15(9):925–934, 1998.

[21] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017.

[22] A. Razdan. Knot placement for b-spline curve approximation. *Tempe, AZ: Arizona State University*, 1999.

[23] O. Schall, A. Belyaev, and H.-P. Seidel. Robust filtering of noisy scattered point data. In *Eurographics/IEEE VGTC Symposium Proceedings Point-Based Graphics*, pages 71–144. IEEE, 2005.

[24] S. M. H. Shamsuddin and M. A. Ahmed. A hybrid parameterization method for nurbs. In *Computer Graphics, Imaging and Visualization, 2004. CGIV 2004. Proceedings. International Conference on*, pages 15–20. IEEE, 2004.

[25] J. Sklansky and V. Gonzalez. Fast polygonal approximation of digitized curves. *Pattern Recognition*, 12(5):327–331, 1980.

[26] F. Steinke, B. Schölkopf, and V. Blanz. Support vector machines for 3d shape processing. *Computer Graphics Forum*, 24(3):285–294, 2005.

[27] G. Taubin. Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 13(11):1115–1138, 1991.

[28] V. Tongur and E. Ülker. B-spline curve knot estimation by using niched pareto genetic algorithm (npga). In *Intelligent and Evolutionary Systems*, pages 305–316. Springer, 2016.

[29] E. Ülker. B-spline curve approximation using pareto envelope-based selection algorithm-pesa. *International Journal of Computer and Communication Engineering*, 2(1):60, 2013.

[30] O. Valenzuela, B. Delgado-Marquez, and M. Pasadas. Evolutionary computation for optimal knots allocation in smoothing splines. *Applied Mathematical Modelling*, 37(8):5851–5863, 2013.

[31] Y. Yuan, N. Chen, and S. Zhou. Adaptive b-spline knot selection using multi-resolution basis set. *IIE Transactions*, 45(12):1263–1277, 2013.