Highlights

Scattered Points Interpolation with Globally Smooth B-Spline Surface using Iterative Knot Insertion

Xin Jiang,Bolun Wang,Guanying Huo,Cheng Su,Dong-Ming Yan,Zhiming Zheng

- We introduce a new method to interpolate scattered 3D data points with a single B-spline surface patch, whose knot vectors are generated according to the parametrization of the scattered data points by iteratively inserting knots to the knot vectors. The generated knot vectors guarantee the existence of interpolation surfaces, while often using fewer control points.
- Our method introduces a novel parameter δ to improve the stability of control point solving. By increasing δ, the interpolation errors will be reduced.
- Since our knot vector construction algorithm guarantees the existence of the interpolation surface, the control points can be solved by a global fairing energy minimization procedure with strict interpolation constraints. Thus, a globally smooth B-spline surface can be generated.

Scattered Points Interpolation with Globally Smooth B-Spline Surface using Iterative Knot Insertion

Xin Jiang[a,e,f], Bolun Wang[a,b,e,*], Guanying Huo[a,e,f], Cheng Su[a,e,f], Dong-Ming Yan[c,d] and Zhiming Zheng[a,e,f]

[a]Key Laboratory of Mathematics, Informatics and Behavioral Semantics (LMIB), Institute of Artificial Intelligence, Beihang University, Beijing, China
[b]King Abdullah University of Science and Technology, Saudi Arabia
[c]National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, 100190, China
[d]State Key Laboratory of Hydro-Science and Engineering, Tsinghua University, Beijing 100084, China
[e]Peng Cheng Laboratory, Shenzhen, Guangdong, China
[f]Zhengzhou Aerotropolis Institute of Artificial Intelligence, Zhengzhou, Henan, China

ARTICLE INFO

Keywords:
B-spline surface
Scattered data points interpolation
Knot vector construction
Fairing energy minimization

ABSTRACT

We introduce a new method to interpolate scattered 3D data points with a single B-spline surface patch which is globally smooth. Given a set of scattered 3D data points and their corresponding parametrization, our method first constructs a set α of B-spline bases using a weighted strategy, and inserts knots to the knot vectors based on α. Then, the knot insertion procedure is iterated until a set β of B-spline bases exists, which indicates the existence of the interpolation surface. Finally, by applying the fairing energy minimizing with interpolation constraints, a globally smooth B-spline surface which interpolates the data points can be produced. Experimental results demonstrate that the generated B-spline surfaces often have fewer control points than those of traditional methods, while keeping the scattered data points interpolated accurately. The implementation of our algorithm and the scripts to reproduce all the results are available at https://github.com/wangbolun300/sparse_data.

1. Introduction

Non-Uniform Rational B-splines (NURBS) have become an industry standard in Computer-Aided Design (CAD), Computer-Aided Engineering (CAE) and Computer-Aided Manufacturing (CAM) [14, 25]. A B-spline surface is often used to approximate/interpolate a 3D point set acquired from real objects in reverse engineering [29]. B-spline surface interpolation methods generate surfaces passing through the data points, while approximation methods allow deviation of the surface from the data points. However, most of the B-spline surface interpolation algorithms assume that the data points are in grids or in rows [25, 11, 4, 34, 30, 32], which cannot always be satisfied by various data sources (for example, laser scanners), or require a large number of control points if the data points are scattered [15, 25]. In this paper, we focus on reconstructing a B-spline interpolation surface from scattered 3D data points.

Generally, given a point set, computing a B-spline surface which fits the points often contains the following steps: parametrization, knot vector generation and control point solving. A lot of works have been done on 3D data parametrization. For data points in rows or in grids, the parameters can be calculated through chordal parameterization or centripetal parametrization [25]. For scattered data such as physical quantities collected in geology, meteorology and oceanography, experimental results produced in chemistry and engineering, and, computation values for scattered inputs, the data points usually can be transformed into height-fields, and the u-v parameters can be directly taken from the x-y coordinates [15]. Especially, in Computer Graphics community, there are numerous works on parametrization for polygonal meshes [6, 16, 19] and point clouds [21, 2, 38]. Knot vector generation is crucial to B-spline surface fitting, since it decides the number and the distribution of control points. After constructing the knot vectors, the surface can be obtained by control point solving. In this paper, we focus on the latter two steps: knot vector generation and control point solving. The parametrization of the data points can be obtained from works mentioned above, such as harmonic parametrization [6].

More control points lead to a strong ability of shape approximation to the data points. To generate appropriate surfaces for different applications, approximation methods often experimentally construct an initial surface, then progressively update knot vectors and control points to achieve a better approximation. During this process, fairness terms such as thin-plate energy are usually used to achieve better quality in the smoothness of the fitted surfaces. However, one often needs to choose proper parameters to balance smoothness and accuracy, which is not an easy task [14].

Interpolation surfaces naturally keep a high accuracy in fitting the data points, but the existence of the interpolation surfaces needs to be guaranteed by properly initializing the knot vectors before control point solving. For data points in grids, it is relatively easy to set up the knot vectors by distributing control points evenly to the data points. For randomly distributed data, traditional methods such as [25, 11, 4, 34] conservatively insert too many knots into knot vectors, which lead to enormous quantity of control points, large computation time and machine storage usage. In recent

*Corresponding author
wangbolun@buaa.edu.cn (B. Wang)

years, Wang et al. [30, 32] introduce methods for B-spline lofting surface generation which can significantly reduce the number of control points. However, these methods are designed to deal with data points given in rows. For scattered data points, it is still not obvious how to reduce the control points while ensuring the existence of the interpolation surfaces.

In this paper, we introduce a novel method for scattered data points interpolation with B-spline surfaces. Inspired by [32], our method, using an iterative knot insertion algorithm, constructs knot vectors directly from the parametrization of the data points while ensuring the existence of the interpolation surface. Then we solve the control points by minimizing a global fairing term to obtain a smooth surface. Our main contributions include:

- **Existence of Interpolation Surface.** Our method directly generates knot vectors according to the parametrization of the scattered data points by iteratively inserting knots to the knot vectors. The generated knot vectors can guarantee the existence of interpolation surfaces, while often using fewer control points.
- **Numerical stability.** Our method introduces a novel parameter δ to improve the stability of control point solving. By increasing δ, the interpolation errors will be reduced.
- **Global Smoothness.** Since our knot vector construction algorithm guarantees the existence of the interpolation surface, the control points can be solved by a global fairing energy minimization procedure with strict interpolation constraints. Thus, a globally smooth B-spline surface can be generated.

2. Related Work

In this section, we provide a brief overview of previous B-spline surface fitting techniques, including knot vector generation and fairing.

2.1. Knot vector generation

It is crucial to properly construct knot vectors when interpolating the data points. If the assumption that the data points are distributed in grids is true, the knot vectors $\mathbf{U}$ and $\mathbf{V}$ can be calculated by averaging the u and v parameters [25, 4, 14]. The averaging method takes the parameters of the gridded points as inputs, and the generated knot vectors distribute one control point to each data point, thus the strict interpolation constraints can be met.

As mentioned above, the assumption cannot always be satisfied due to the various sources of data points. To guarantee the existence of the interpolation surface, one option is directly applying the averaging method [25] to the scattered data points. But it may lead to a great number of control points since a large number of them might be redundant. Lee et al. [15] provide an easy method to update knot vectors by double the number of knots in each iteration. A condition is also provided in [15] to determine if the generated knot vectors are sufficient for interpolating the scattered points. However, the number of required control points may be arbitrary large if the parameters of two neighbouring data points are close to each other. We compare these two methods with our method in Section 4.

To construct surfaces in more flexible formats, approximation methods such as [15, 20, 14, 33, 39, 18, 17, 5] can fit the input data points with fewer control points, while keeping high quality in the smoothness. In [15], the knot vectors are simply refined by increasing the knot density by 4× larger than the previous iteration to increase the approximation accuracy. Weiss et al. [33] design a shape dependent knot insertion strategy to detect more sensible knot intervals, and optimize the placement of the new knots within the intervals. To improve the surface quality for approximating unevenly distributed data points, Zhang et al. [39] proposed to iteratively insert and optimize knots based on the geometric features. Approximation methods can be used to remove high-frequency noise of the input data, and construct surfaces with fewer control points if the data points are too many. But the higher fitting accuracy is, the more iterations are required, thus taking longer runtime.

An orthogonal methodology is lofting. B-spline lofting algorithms are designed to fit data points given in rows. Approximation algorithms [23, 27, 24, 28] can be used to construct smooth surfaces with fewer control points. But the higher approximation accuracy always leads to more iterations, which can be time consuming. Woodward [34] and Piegl and Tiller [26] interpolate each row of points using a common knot vector, and try to reduce as many as knots when constructing the knot vectors. But neither of the two methods can guarantee the existence of the interpolation curves. To further reduce the number of control points, while guaranteeing the existence of the interpolation surface, Wang et al. [32] give the necessary and sufficient condition to construct a full-rank interpolation matrix for each contour curve. Their algorithm also gives proper flexibility for selecting knots to common knot vectors to avoid numerical stability problems. This method can effectively reduce the number of control points, and guarantees that the data points are exactly interpolated [32, 30]. However, if directly apply the lofting method to interpolate randomly distributed data points, the generated surface can exactly interpolate the data points, but could be extremely folded even if it's globally smooth, due to the unbalanced lengths of knot vectors. We demonstrate some examples in Figure 1 and Figure 7.

2.2. Surface fairing

For both interpolation and approximation algorithms, unhandled degrees of freedom may cause un-smoothness of the generated surface. Thus fairing functions are widely used to handle the degrees of freedom and improve the smoothness of the surfaces [37, 10, 13, 7, 31, 22, 33, 36, 35, 9, 14]. By combining the fairing terms with approximation/interpolation conditions, the noise or wiggles of the surface can be removed, while keeping the shape of the sur-

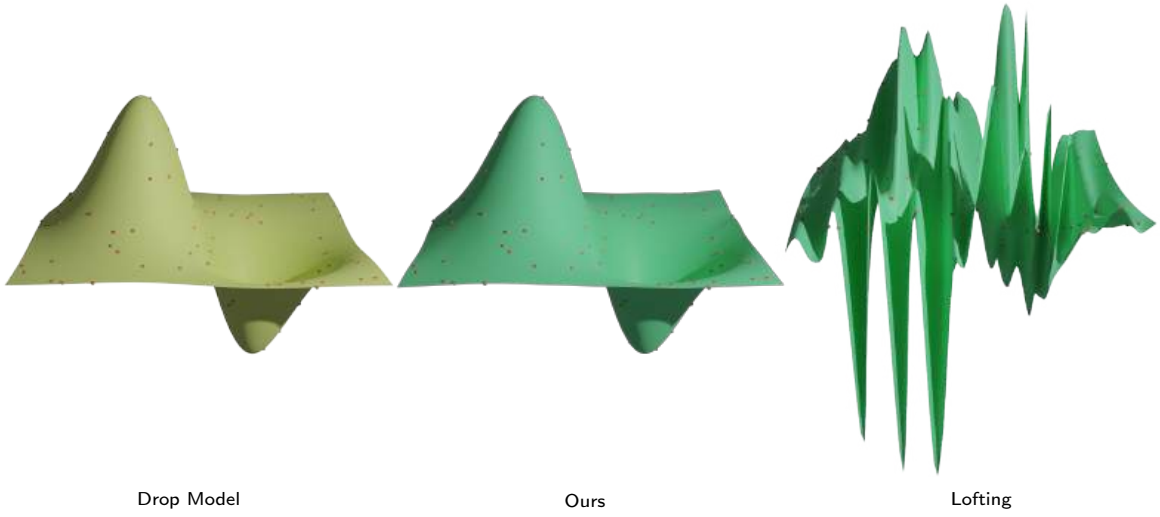

Figure 1: An example (right) of directly applying the lofting method [32] to generate knot vectors when interpolating 100 scatter points (red dots) randomly sampled from the Drop model (left). The number of control points for the interpolation surface is 100×4 with the input parameter $per = 0.5$. The results show that the lofting surface is extremely folded and spiky even if the control points are solved through global fairing energy minimizing (Formula 6), while our method generates an interpolation surface (middle) using 18×18 control points, and has higher quality.

face constrained by the data points. In this paper, we apply thin-plate energy which is the most commonly used in global surface fairing [7].

Employing thin-plate energy will lead to loss of sharp features since it only consider global smoothness. Ye et al. [36] fit B-spline surface with normal constraint, by minimizing an energy composed of position approximation error and normal approximation error parts. In this way, the reconstructed surfaces approximate the input data, and also preserve the normal or tangent information. Yamaura et al. [35] recover the height-field B-spline surfaces from its surface normal vectors. Instead of using the position of the 3D data points, Yamaura et al. [35] combine the thin-plate energy as well as normal vectors when solving the control points. More recently, Kawasaki et al. [13] introduce an image processing based normal field fairing method to preserve the surface features.

It should be noticed that, for approximation algorithms, post-processing fairing methods [10, 9, 13, 14] can be employed since these algorithms allow deviation of the surface from the data points. Post-processing fairing terms remove (or preserve) sharp parts by directly modifying the positions of the control points of the approximation surface, which may enlarge the approximation error. Our method requires strict interpolation conditions, which means we can only fair the surface when solving control points, but not move the control points locally to improve the local smoothness.

In this paper, our knot vectors $\mathbf{U}$ and $\mathbf{V}$ will be generated directly from the parametrization of the scattered data points, with the existence of the interpolation surface is guaranteed (Equation 3). Then, we employ thin-plate energy to generate a smooth surface which can accurately interpolate the given scattered data points.

3. Methodology

To make the paper self-contained, we first introduce the basic concepts of B-spline curves and surfaces. A B-spline curve is defined as:

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i, \tag{1}$$

where $N_{i,p}(u)$ are the basis functions with degree p defined over knot vector $\mathbf{U} = \{u_0, u_1, \dots, u_{n+p+1}\}$, $P_i, i = 0. \dots, n$ are the control points.

A B-spline surface with degree $p \times q$ can be denoted as:

$$S(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) P_{i,j}, \tag{2}$$

where $N_{i,p}(u)$ and $N_{j,q}(v)$ are the basis functions with degree p and q, defined over knot vectors $\mathbf{U} = \{u_0, u_1, \dots, u_{n+p+1}\}$ and $\mathbf{V} = \{v_0, v_1, \dots, v_{m+q+1}\}$, respectively. $P_{i,j}, i = 0 \dots n, j = 0, \dots, m$ are the control points.

Given a point set $\mathbf{Q} = \{Q_0, Q_1, \dots, Q_s\}$ and its parametrization $\bar{\mathbf{U}} = \{(\bar{u}_0, \bar{v}_0), (\bar{u}_1, \bar{v}_1), \dots, (\bar{u}_s, \bar{v}_s)\}$, the B-spline interpolation problem can be described as: finding suitable knot vectors $\mathbf{U}$, $\mathbf{V}$ and the corresponding control points $P_{i,j}$, such that,

$$\begin{aligned} &S(\bar{u}_r, \bar{v}_r) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(\bar{u}_r) N_{j,q}(\bar{v}_r) P_{i,j} = Q_r, \\ &\forall r \in \{0, 1, \dots, s\}. \end{aligned} \tag{3}$$

In this paper, the input data points $\mathbf{Q}$ are scattered, and its parametrization $\bar{\mathbf{U}}$ can be obtained from the x-y coordinates of $\mathbf{Q}$ if it is a height-field, or parametrization methods such as [6] if the data points are the vertices of a polygon mesh. Our algorithm is composed of 3 stages (Figure 2):

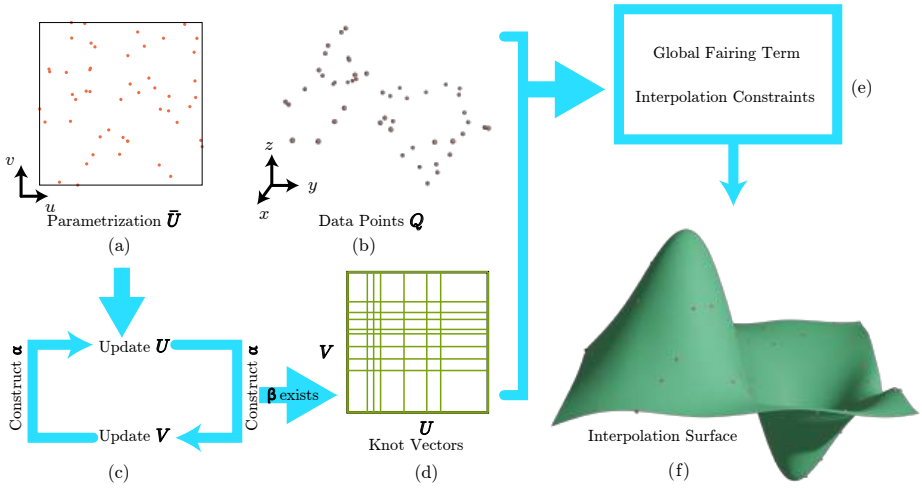

Figure 2: Overview of our algorithm: we take the point set $\mathbf{Q}$ (b) and its parametrization $\bar{\mathbf{U}}$ (a) as inputs. The knot vectors $\mathbf{U}$ and $\mathbf{V}$ (d) are generated by constructing α and iteratively inserting knots (c), until β exists (Section 3.1.2). Then, by applying a global fairing term with equality constraints (e) to solve the control points (Section 3.3), the interpolation surface (f) can be generated.

- constructing a set α of B-spline bases, and inserting knots into knot vectors based on α (Figure 2, a, c),
- iterating Stage 1 to update knot vectors $\mathbf{U}$ and $\mathbf{V}$ (Figure 2, d) alternatively, until a set β of B-spline basis exists, and,
- solving control points by minimizing a fairing energy term with interpolation constraints (Figure 2, e) to generate the interpolation surface (Figure 2, f).

The first and second stage generates knot vectors $\mathbf{U}$ and $\mathbf{V}$ directly from the parametrization $\bar{\mathbf{U}}$, with the existence of interpolation surface guaranteed. Similar to [32], we use a parameter δ to improve numerical stability. Then stage 3 takes the knot vectors $\mathbf{U}$ and $\mathbf{V}$ obtained from Stage 2 and solves the control points by minimizing the thin-plate energy with equality constraints. After the third stage, the output surface interpolates the scattered data points accurately with fewer control points compared with traditional methods [25, 15], which lead to less computation time (Section 4).

3.1. Knot Vector Generation

We assume that the point set $\mathbf{Q}$ with parameterization $\bar{\mathbf{U}}$ can be interpolated by a B-spline surface $S(u, v)$ with knot vectors

$$\mathbf{U} = \{u_0, u_1, \dots, u_{n+p+1}\}, \quad \mathbf{V} = \{v_0, v_1, \dots, v_{m+q+1}\},$$

which means the interpolation constraints (Equation 3) are satisfied. From Equation 3, it can be further derived that:

$$\begin{aligned} Q_r = S(\bar{u}_r, \bar{v}_r) &= \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(\bar{u}_r)N_{j,q}(\bar{v}_r)P_{i,j} \\ &= \sum_{i=0}^{n}\Big(\sum_{j=0}^{m} N_{j,q}(\bar{v}_r)P_{i,j}\Big)N_{i,p}(\bar{u}_r) = \sum_{i=0}^{n} N_{i,p}(\bar{u}_r)C_i(\bar{v}_r), \end{aligned} \tag{4}$$

where $r = 0, 1, \dots, s$. $\{C_i(\bar{v}_r), i = 0, \dots, n\}$ are the $n+1$ control points of the curve which interpolates data points with their v parameters equal to $\bar{v}_r$. We can denote the interpolation problem as:

$$\mathbf{Q} = \mathbf{HP}, \tag{5}$$

where

$$\mathbf{Q} = [Q_0, \dots, Q_s]^\top,$$
$$\mathbf{P} = [P_{0,0}, \dots, P_{0,m}, \dots, P_{n,0}, \dots, P_{n,m}]^\top,$$

and $\mathbf{H} = \{h_{i,j}\}, h(i, i(n+1)+j) = N_{i,p}(\bar{u}_r)N_{j,q}(\bar{v}_r)$. The $s+1$ data points $\mathbf{Q}$ can be interpolated, if the matrix $\mathbf{H}$ is row full rank. In Section 3.1.1 we provide a sufficient condition for the existence of interpolation surfaces, so our knot vector generation algorithm can be built upon it.

3.1.1. *Sufficient Condition for Surface Interpolation*

Intuitively, the data points $\mathbf{Q}$ can be interpolated, if there exists an injection $i : \mathbf{Q} \to \mathbf{P}$, which maps each data point to one of its corresponding control points. In this case, for each data point, there is at least one control point distributed to it, thus there are enough degrees of freedom on the right part of Equation 5 to construct a surface passing through the $s+1$ data points, and the interpolation conditions can be satisfied. To generate knot vectors that guarantee the existence of the interpolation surface, we first introduce the following two conditions:

Condition 1. *There exists a set* $\alpha = \{N_{k_i,p}(\bar{u}_i) \neq 0\}_{i=0}^{s}$, $k_i \in \{0, \dots, n\}$, *which satisfies that* $\forall i, j \in \{0, \dots, s\}$, *if* $\bar{v}_i = \bar{v}_j, \bar{u}_i < \bar{u}_j$, *then* $k_i < k_j$.

Condition 2. *There exists a set* $\beta = \{N_{r_i,q}(\bar{v}_i) \neq 0\}_{i=0}^{s}, r_i \in \{0, \dots, m\}$, *which satisfies that* $\forall i, j \in \{0, \dots, s\}$, *if* $N_{k_i,p}(\bar{u}_i), N_{k_j,p}(\bar{u}_j) \in \alpha, k_i = k_j, \bar{v}_i < \bar{v}_j$, *then* $r_i < r_j$.

Then, we can have the following proposition as a sufficient condition of the existence of an interpolation surface:

Proposition 1. *A point set* $\mathbf{Q} = \{Q_i | i = 0, \dots, s\}$ *with its parametrization* $\bar{\mathbf{U}} = \{(\bar{u}_i, \bar{v}_i) | i = 0, \dots, s\}$ *can be interpolated by a B-spline surface (Equation 2), if the two conditions Condition 1 and Condition 2 are true.*

The proof of Proposition 1 is given in Appendix A. Based on Proposition 1, we introduce an iterative knot insertion algorithm which can generate knot vectors which interpolate the given scattered data points.

3.1.2. Iterative Knot Insertion

Essentially, if α and β exist, at least one control point will be assigned to each data point, which means the injection i exists, thus the interpolation surface exists. Next, we introduce an iterative knot insertion algorithm to generate knot vectors $\mathbf{U}$ and $\mathbf{V}$ which make sure Proposition 1 is satisfied. We construct α explicitly, then iteratively update $\mathbf{U}$ and $\mathbf{V}$ until β exists. Built on the sufficient condition, our method guarantees the existence of the interpolation surface, while often using fewer control points when interpolating scattered data points.

We assume that there are $l+1$ different v parameter values $\bar{v}_0, \dots, \bar{v}_l$ in the parameterization $\bar{\mathbf{U}}$. To make sure there exists a set α that satisfies Condition 1, we need to construct a knot vector $\mathbf{U}$ to make sure that all the data points can be interpolated by the $l+1$ iso-v curves

$$C_r(u) = \sum_{i=0}^{n} N_{i,p}(u) C_i(\bar{v}_r),$$

where $r = 0, \dots, l$. To obtain that, we can adopt the method in [32] to iteratively update the knot vector $\mathbf{U}$ until α exists, as an initialization of $\mathbf{U}$. Then, we use a weighted strategy to construct α explicitly, and then update $\mathbf{V}$ to make sure β exists. The generated knot vectors $\mathbf{U}$ and $\mathbf{V}$ can be directly applied to the control point solving stage described in Section 3.3 so the interpolation surface can be constructed. Analogously, we can also initialize $\mathbf{V}$ first, then update $\mathbf{U}$ to make sure β exists. In our algorithm, we iteratively insert knots to $\mathbf{U}$ and $\mathbf{V}$ alternatively to avoid unbalance of the lengths of the two knot vectors.

We describe our knot vector generation algorithm in Algorithm 1. The inputs of our algorithm are the parametrization $\bar{\mathbf{U}}$ of the data points $\mathbf{Q}$, the degrees p, q of the surface, a parameter *per* to control the flexibility of inserting knots using [32]'s method, and a parameter δ for selecting proper elements to avoid numerical problems when constructing α in Condition 1. The parameter δ is explained in Section 3.2. For convenience, we assume $\mathbf{U}$ is fixed after initialization, and then describe how to insert knots to $\mathbf{V}$ until β exists.

Initialization of U. Assume that there are $b+1$ data points $Q_0, \dots, Q_b$ whose parameters $\bar{v}_0 = \dots \bar{v}_b = \bar{v}_r, \bar{u}_i < \bar{u}_{i+1}$, and given a knot vector $\mathbf{U} = \{u_0, \dots, u_g\}$, the knot vector may need to be updated to interpolate these points. Denote an intermediate knot vector as $\mathbf{U_r} = \{u_{r,0}, u_{r,1}, \dots\}$.

Algorithm 1 $\mathbf{U}, \mathbf{V}$ Knot Vectors Generation

function ITERATIVEKNOTINSERTION($\bar{\mathbf{U}}, per, \delta, p, q$)
  $\bar{u}, \bar{v} \leftarrow \bar{\mathbf{U}}$ ▷ The u and v parameters of the points
  $\mathbf{U}, \mathbf{V}, \mathbf{V}_1, \mathbf{V}_2, \bar{\mathbf{V}}_1, \bar{\mathbf{V}}_2 \leftarrow \emptyset, d_1, d_2 \leftarrow 0$
  $\mathbf{U} \leftarrow$ INIT($\bar{\mathbf{U}}, \mathbf{U}, per$), $\mathbf{V} \leftarrow$ INIT($\bar{\mathbf{V}}, \mathbf{V}, per$) ▷ Initialize the knot vectors using [32]'s method
  updateV$\leftarrow$SIZE($\mathbf{U}$)> SIZE($\mathbf{V}$)
  updated$\leftarrow$ *true*
  while updated **do**
    if updateV **then**
      $\mathbf{V}_1 \leftarrow \mathbf{U}, \mathbf{V}_2 \leftarrow \mathbf{V}, \bar{\mathbf{V}}_1 \leftarrow \bar{u}, \bar{\mathbf{V}}_2 \leftarrow \bar{v}$
      $d_1 \leftarrow p, d_2 \leftarrow q$
    else
      $\mathbf{V}_1 \leftarrow \mathbf{V}, \mathbf{V}_2 \leftarrow \mathbf{U}, \bar{\mathbf{V}}_1 \leftarrow \bar{v}, \bar{\mathbf{V}}_2 \leftarrow \bar{u}$
      $d_1 \leftarrow q, d_2 \leftarrow p$
    $\bar{v}_0, \dots, \bar{v}_l \leftarrow \bar{\mathbf{V}}_2$ ▷ There are $l+1$ different values in $\bar{\mathbf{V}}_2$
    for $r = 0, \dots, l$ **do**
      $\bar{u}_0, \dots, \bar{u}_b \leftarrow \bar{\mathbf{V}}_1$ ▷ $\bar{v}_0 = \dots = \bar{v}_b = \bar{v}_r$
      $\mathcal{L}_r \leftarrow$LISTCONSTRUCTION($\bar{u}_0, \dots, \bar{u}_b, d_1, \mathbf{V}_1, \delta$)
    $\mathcal{L} \leftarrow \mathcal{L}_0, \dots, \mathcal{L}_l$
    $w \leftarrow$WEIGHT($\mathcal{L}$) ▷ Weight Calculation
    $\alpha \leftarrow \alpha$ GENERATION($\mathcal{L}, w$)
    $s_1 \leftarrow$ SIZE($\mathbf{V}_1$), $s_2 \leftarrow$ SIZE($\mathbf{V}_2$)
    $N \leftarrow$MAX($2, s_1 - s_2$)
    $\mathbf{V}_3 \leftarrow$INSERTKNOTS($\mathbf{V}_2, \alpha, N$) ▷ Updating the knot vector using [32]'s method
    $s_3 \leftarrow$SIZE($\mathbf{V}_3$)
    if $s_3 > s_2$ **then**
      $updated \leftarrow true$, updateV$\leftarrow$ NOT(updateV)
    else
      $update \leftarrow false$
    $\mathbf{V}_2 \leftarrow \mathbf{V}_3$
    $\mathbf{U}, \mathbf{V} \leftarrow \mathbf{V}_1, \mathbf{V}_2$
  return $\mathbf{U}, \mathbf{V}$

We can summarize the knot vector generation method in [32] as the following:

First, calculate the intermediate $\mathbf{U}' = \{u'_0, \dots, u'_{b+p+1}\}$ based on the averaging method [25, 4]:

$$u'_0 = \dots u'_p = 0, \quad u'_{b+1} = \dots u'_{b+p+1} = 1,$$
$$u'_{p+k} = \frac{1}{p} \sum_{i=k}^{k+p-1} \bar{u}_i, k = 1, \dots, b-p.$$

and for $i = 0, \dots, p$, set $u_{r,i} = u'_i$. Then, iterate i from $p+1$ to b, for each i, construct an interval $[a_i, b_i] \subseteq (\bar{u}_{r,i-p-1}, \bar{u}_{r,i})$, and check if there is a $u_{g(i)} \in [a_i, b_i]$, where $g(p) \geq i, g(i) > g(i-1)$ for $i > p$. If true, then set $u_{r,i} = u_{g(i)}$. Otherwise, $u_{r,i} = u'_i$. Finally, the knot vector $\mathbf{U}$ can be generated by merging $\mathbf{U}$ and $\mathbf{U_r}$. In [32]'s method, the interval $[a_i, b_i]$ helps to improve the solving stability by avoiding the selected parameter too close to $\bar{u}_{r,i-p-1}$ or $\bar{u}_{r,i}$. The generated knot vector $\mathbf{U}$ can guarantee that the data points $Q_0, \dots, Q_b$ can be interpolated by the curve $C_r(u)$.

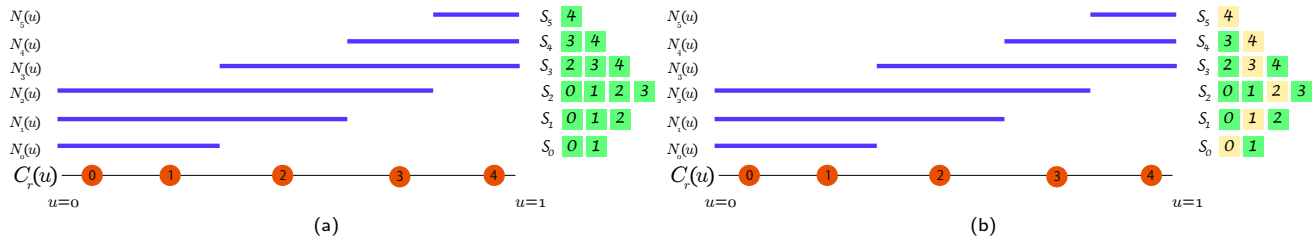

Figure 3: An example of building a list L_r for curve $C_r(u)$. The red dots are the u parameters of the five input data points, where the corresponding parameters $\bar{v}_0 = \cdots = \bar{v}_4 = \bar{v}_r$. After initialization, the data points can be interpolated by $C_r(u)$. The blue lines indicate the support regions of the B-spline basis functions. (a) We build six lists $\mathcal{S}_0, \ldots, \mathcal{S}_5$ (the green boxes) for the B-spline basis functions, and each of the lists contains the indices of the data points supported by the basis function. (2) Using Algorithm 2, the list $\mathcal{L}_r$ can be constructed through selecting one element (marked as yellow boxes) from each of $\mathcal{S}_0, \ldots, \mathcal{S}_5$. The generated $\mathcal{L}_r = \{0, 1, 2, 3, 4, 4\}$

With [32]'s method, we can iterate over different curves $C_r(u)$ to update $\mathbf{U}$ (line 4 in Algorithm 1) as an initialization. The generated knot vector $\mathbf{U}$ can guarantee that all the $s+1$ data points can be interpolated by the $l+1$ iso-v curves, which means Condition 1 can be satisfied. Figure 21 shows that if the data points are distributed evenly in grids, the initialized knot vector $\mathbf{U}$ and $\mathbf{V}$ can be directly used to generate the interpolation surfaces, which means Proposition 1 is satisfied, thus no knot needs to be inserted into the knot vectors. In Figure 22, we also show that if the data points are given in a clear row-structure, the control points of the interpolation surfaces will also be in rows, since the initialization stage (line 4) is directly borrowed from [32], which is a lofting algorithm.

Next, we show how to select proper elements from the B-spline bases of $C_r(u), r = 0, \ldots, l$ to construct the α set in Condition 1.

Construction of α candidates $\mathcal{L}$. According to the local support property of B-splines, for a u parameter $\bar{u} \in [u_i, u_{i+1})$, if $\bar{u} \neq u_i$, then there are $p+1$ non-zero B-spline bases $N_{i-p,p}(\bar{u}), \ldots, N_{i,p}(\bar{u})$. If $\bar{u} = u_i$, there are p non-zero bases $N_{i-p,p}(\bar{u}), \ldots, N_{i-1,p}(\bar{u})$. The elements in α are selected from these non-zero bases. For each data point, there are multiple corresponding bases; for each B-spline basis, there could also be multiple corresponding data points. If there are two data points parametrized as $\{\bar{u}_{i_1}, \bar{v}_i\}$ and $\{\bar{u}_{i_2}, \bar{v}_i\}$ selecting one unique B-spline basis as their corresponding element in α, then α will not satisfy Condition 1. Thus, we first build a group of lists $\mathcal{L}$ which contains the candidates of α. For each data point, there are multiple corresponding elements in $\mathcal{L}$, but for each element in $\mathcal{L}$, there is at most one corresponding data point. In this way, for each data point, if we select any one of its corresponding elements in $\mathcal{L}$ to construct α, the Condition 1 can be satisfied. We first build $n+1$ lists $\mathcal{S}_0, \ldots, \mathcal{S}_n$ for the $n+1$ basis functions of curve $C_r(u)$ to make sure that $k_i < k_j$ when $\bar{v}_i = \bar{v}_j, \bar{u}_i < \bar{u}_j,$. Iterating over i from 0 to b, for each k_i that satisfies $N_{k_i,p}(\bar{u}_i) \neq 0$, we push the index i to the top of $\mathcal{S}_{k_i}$. Denoting a list with $n+1$ elements as $\mathcal{L}_r$, we hope to build a projection from indices of the data points to the elements of $\mathcal{L}_r$: for $\forall i, j \in \{0, \ldots, b\}, \exists k_i, k_j, s.t. \mathcal{L}_{k_i} = i, \mathcal{L}_{k_j} = j$, and if $\bar{u}_i < \bar{u}_j$, then $k_i < k_j$. If the $l+1$ lists $\mathcal{L} = \{\mathcal{L}_i\}_{i=0}^{l}$ can be constructed (line 18), then α can be constructed by choosing any $N_{k_i,p}(\bar{u}_i)$ such that $\exists r, \mathcal{L}_r(k_i) = i$. After constructing α, we update knot vector $\mathbf{V}$ using [32]'s method (line 23) to

Algorithm 2 Construction of list $\mathcal{L}_r$

```
1:  function ListConstruction(ū_0, ..., ū_b, p, U, δ)
2:      S_0, ..., S_n ← ∅
3:      for i = 0, ..., n do
4:          for j = 0, ..., b do
5:              [a, b] = GetInterval(i, U, δ)                ▷ Section 3.2
6:              if ū_j ∈ [a, b] then
7:                  S_i push(j)
8:      c ← ∅
9:      for i = 1, ..., n do
10:         if S_i = ∅ then
11:             c ← ∅
12:         if size(S_i) = 1 then
13:             c = S_i(0)
14:         if size(S_i) > 1 then
15:             if c = ∅ then
16:                 c = S_i(0)
17:             else
18:                 c = Next(c, S_i)
19:         L_r(i) ← c
20:     return L_r
21: function Next(c, S_i)
22:     w ← ∅, t ← Top(S_i)
23:     if t = c then
24:         return t
25:     while S_i ≠ ∅ do
26:         t ← Pop(S_i)
27:         if S_i ≠ ∅ then
28:             w ← Top(S_i)
29:             if w = c then
30:                 return t
31:         else
32:             return t
```

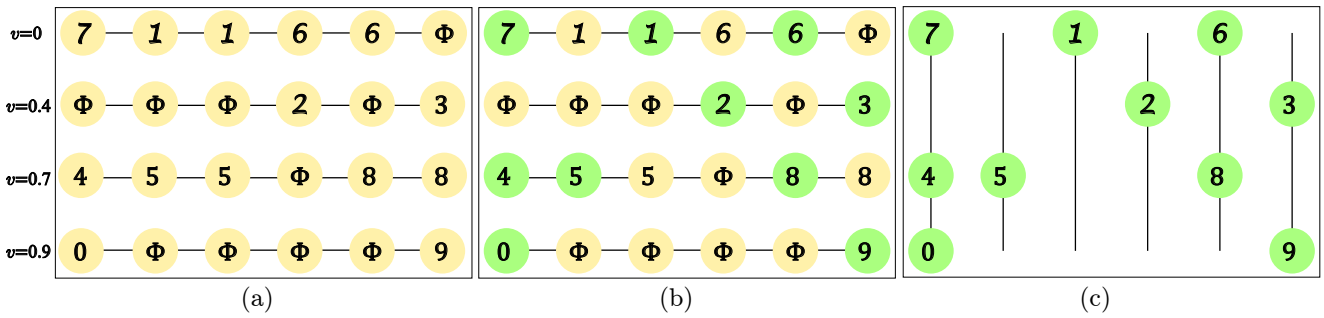

Figure 4: An example of constructing α and updating $\mathbf{V}$ to make sure Proposition 1 is satisfied. (a) The four rows of yellow dots represent four lists $\mathcal{L}$. The v parameter is marked on the left of the three figures. (b) Using our weighted method, the α is selected and marked as green dots. For each of the 10 data points, there is only one corresponding element in α. (c) Iterate over each column of the green dots and update $\mathbf{V}$, until the generated $\mathbf{V}$ makes sure that the data points in each row can be interpolated. Consequently, Proposition 1 is satisfied.

make sure the curve

$$C_{k_i}(v) = \sum_{j=0}^{m} N_{j,q}(v) P_{k_i,j}$$

interpolates points with parameters $\bar{v}_i$, which satisfy that

$$N_{k_i,p}(\bar{u}_i) \in \alpha.$$

Then, the knot vectors $\mathbf{U}, \mathbf{V}$ that guarantee to interpolate $\mathbf{Q}$ are obtained. We provide the following procedure (Algorithm 2) to construct $\mathcal{L}_r$. Figure 3 is an illustration of building list $\mathcal{L}_r$ for curve $C_r(u)$ whose degree $p = 2$. Using Algorithm 2, each of the data points (red dots) has at least one corresponding element in list $\mathcal{L}_r$

$\mathcal{L}$ can be generated through iterating over different $\bar{v}$ parameters using Algorithm 2. It can be seen that for each $\bar{u}_i, \bar{v}_i = \bar{v}_r$, there can be one or more $k_i, s.t.\ \mathcal{L}_r(k_i) = i$, and for each $r \in 0, \dots, l$, $i \in 0, \dots, n$, $\mathcal{L}_r(i)$ has either zero or one element. If $\mathbf{V}$ is updated based on $\mathcal{L}$ instead of selecting α out of $\mathcal{L}$, the knot vectors $\mathbf{U}, \mathbf{V}$ can still guarantee to interpolate $\mathbf{Q}$. However, as shown in Table 3 (*No Selecting* strategy), the surface will be too conservative, which means there are too many redundant control points. Thus, we provide a method of constructing a proper α by assigning weights to the elements of $\mathcal{L}$, and alternatively updating $\mathbf{V}$ and $\mathbf{U}$.

Weighted method for α construction. We observe that,

1. For a given k_i, if there are more $r, s.t.\ \mathcal{L}_r(k_i) \neq \emptyset$, then more knots need to be inserted to $\mathbf{V}$.
2. If $\mathbf{U}$ has more knots, then fewer knots are needed to be inserted into $\mathbf{V}$ to construct the interpolation surface.

We assign weights $\mathbf{W} = \{w(r,i) | r = 0, \dots, l, i = 0, \dots, n\}$ to the elements in $\mathcal{L}_r$ (line 19 of Algorithm 1), so that for $\bar{u}_i$ which satisfy that $\bar{v}_i = \bar{v}_r$, for $\forall k_i, s.t.\ \mathcal{L}_r(k_i) = i$, we select $N_{k_i,p}(\bar{u}_i)$ where $\mathcal{L}_r(k_i)$ has the highest weight to construct α. Based on the 1st observation, to reduce the number of control points, *i.e.,* the number of knots inserted to $\mathbf{U}$ or $\mathbf{V}$, for each k_i, there should not be too many h such that $N_{k_i,p}(\bar{u}_h) \in \alpha$. We define the weights as:

$$w(r,i) = w_1(r,i) \cdot w_2(r,i) \cdot w_3(r,i),$$

where

$$w_1(r,i) = 1 / \sum_{k=0}^{l} n_a^2(k,i),$$
$$w_2(r,i) = 1 / n_b(r,i),$$
$$w_3(r,i) = n_c(r,i).$$

$n_a(k,i)$ is the number of control points that $\mathcal{L}_r(i)$ shares with $\mathcal{L}_k(i)$ in curve $C_i(v)$. The more control points that $\mathcal{L}_r(i)$ shares with other $\mathcal{L}_k(i) (k = 0, \dots, l)$, the smaller $w_1(r,i)$ will be. $n_b(r,i)$ is the number of $h, s.t. \mathcal{L}_h(i) \neq \emptyset, h \leq r$. And $n_c(r,i)$ is the number of $h, s.t. \mathcal{L}_r(h) = \mathcal{L}_r(i), h \leq i$. Since the knot vectors are updated alternatively, (we will describe it later), as $\mathbf{U}$ get updated, the number of knots needed to be inserted to $\mathbf{V}$ will be less and less. $n_b(r,i)$ and $n_c(r,i)$ are designed to avoid the selected elements of $\mathcal{L}$ gathering in the curves $C_{k_i}(v)$ who has smaller k_i. From Observation 2, if a larger k_i is selected to construct α, which means it may not need to insert a knot into $\mathbf{V}$ especially for $N_{k_i,p}(\bar{u}_i) \in \alpha$ because it may be dealt with $\mathbf{U}$ knot vector insertion, which is triggered by the length difference of the two knot vectors, as undermentioned. We compare our weighted method in Table 3, with a non-weighted strategy–*Naive Selecting* strategy, which simply select the smallest k_i for $\mathcal{L}_r(k_i) = i$ to construct α. The results show that weights w can significantly reduce the number of control points of the interpolation surface.

After computing the weights $\mathbf{W}$, the elements in $\mathcal{L}$ are selected to construct α. If $\mathcal{L}_r(k_i) = i$ is selected, then for $\forall k_j \neq k_i, \mathcal{L}_r(k_j) = i$, we update $\mathcal{L}$ by setting $\mathcal{L}_r(k_j) = \emptyset$. It can be seen that as the elements in $\mathcal{L}$ are selected, the structure of $\mathcal{L}$ is changed, so the weights should also be updated. Thus, each time we select a few elements of $\mathcal{L}$ into α, then recalculate weights $\mathbf{W}$. Repeat until α is constructed. In our implementation, we set the number of elements selected in each iteration no more than 5. Based on α, we insert knots into $\mathbf{V}$ using [32]. The knot vector generation algorithm terminates when there is no knot can be inserted into $\mathbf{V}$ using [32]'s method. In this case, the β exists, implying the existence of the interpolation surface. Figure 4 shows an ex-

ample of constructing α from lists $\mathcal{L}$ to interpolate 10 points. After generating α and inserting knots, the generated $\mathbf{U}$ and $\mathbf{V}$ guarantee the existence of the interpolation surface.

Alternatively updating of U and V. As assumed, the points $\mathbf{Q}$ are not distributed in grids or in rows, $\mathbf{U}$ could contain very few knots after initialization (line 4 in Algorithm 1). The size of $\mathbf{V}$ could be significantly larger than that of $\mathbf{U}$, just like the Lofting method shown in Figure 1 and Figure 7, which leads to the interpolation surface "folding" in one direction and is highly unfair. To avoid that, we first construct α for iso-v curves, and insert a few knots to $\mathbf{V}$. Then, we construct α for iso-u curves, and insert a few knots to $\mathbf{U}$ (line 23 in Algorithm 1). Repeating this procedure, $\mathbf{U}$ and $\mathbf{V}$ will be progressively updated until Condition 2 is satisfied. The knot vectors will have a similar number of knots, thus the resulting surface can be more fair and smooth. In our implementation, we limit the number of knots inserted to $\mathbf{V}$ no larger than $N = \max(2, |size(\mathbf{U}) - size(\mathbf{V})|)$ (line 22 in Algorithm 1). If the number of knots inserted to $\mathbf{V}$ is larger than N, the knot insertion algorithm will be terminated, and we insert no more than N knots to $\mathbf{U}$, with the same method of constructing α and knot insertion algorithm aforementioned.

The computation complexity of our knot vector generation algorithm is problem-specific. For data points distributed in grids or in rows, the knot vector generation algorithm will be terminated immediately after the initialization stage. For scattered data points, it will require extra iterations to update the knot vectors (as shown in Figure 21 and Figure 22). In general, the computation complexities of our knot vector initialization, weight calculation, α construction and knot insertion are all $\mathcal{O}(s)$, where $s+1$ is the number of data points. However, since our algorithm alternatively updates $\mathbf{U}$ and $\mathbf{V}$, and the α is reconstructed each time we switch the knot vector to be updated, the computation complexity of the weight calculation, α construction and knot insertion could be $\mathcal{O}(n^2)$ for the worst case. Even if so, our knot vector generation method is still efficient compared with the control point solving procedure undermentioned which requires to solve a large scale linear system (Figure 17).

The advantages of our knot vector generation method. We note that, apart from Proposition 1, there are other sufficient conditions for the existence of interpolation surface, such as [25] and [15]. We compare our method with them in Section 4. As a lofting method, [32] can also generate knot vectors that guarantee the existence of the interpolation surface. In Figure 7, we show the interpolation results of [32] by adopting their knot vector generation algorithm, and use the control point solving method described in Section 3.3 to handle the degrees of freedom. The comparisons show that our method generates interpolation surfaces with fewer control points when interpolating scattered data points, and have a better balance in efficiency, smoothness and interpolation accuracy.

Using our knot vector generation method, the number of the required control points highly depends on the parametrization $\bar{\mathbf{U}}$ of the $s+1$ input data points $\mathbf{Q} = \{Q_0, \dots, Q_s\}$. In Figure 23, by interpolating a non-uniformly randomly sampled data set, we show that, for high-resolution areas with a high concentration of points, our method generates more control points compared with those low-resolution areas which have considerably fewer number of data points. The lower bound of the number of control points could be exactly the same as the number of data points $s+1$. It happens when the data points are distributed in grids. In this case, the knot vectors will be generated directly from the initialization stage (line 4) which uses averaging algorithm. In another word, our method uses the same number of control points as those of [25] and [15] when interpolating data points distributed in grids (we show a few examples in Section 4, Figure 21), and performs better when interpolating scattered data points (Section 4).

3.2. Improved Numerical Stability

In Section 3.1.2, we provide a method to generate α and generate knot vectors $\mathbf{U}$ and $\mathbf{V}$ out of it to guarantee the existence of interpolation surface. Theoretically, any B-spline basis that satisfy Condition 1 can be selected to construct α. However, in practice, if $N_{k_i,p}(\bar{u}_i)$ is too close to 0, numerical problems may happen when solving the control points. We can use the notations in the proof of Proposition 1 (in Appendix A) to give a brief explanation: Equation 5 can be converted into $\mathbf{Q} = \begin{bmatrix}\mathbf{E_5}\mathbf{N_6} & 0\end{bmatrix}\mathbf{RP}$, where $\mathbf{N_6}$ and $\mathbf{R}$ are invertible matrices. The interpolation condition can be satisfied, if $\mathbf{E_5}$ is nonsingular. According to [3, 32], $\mathbf{E_5}$ is nonsingular if and only if there exists an elementary column transformation $\mathbf{L}'$, $\mathbf{E}'_\mathbf{5} = \mathbf{E_5}\mathbf{L}'$, such that the values on the diagonal of $\mathbf{E}'_\mathbf{5}$ are non-zero. It is easy to see that since the columns in $\mathbf{E_5}$ are selected out of matrix $\mathbf{E}$ based on α, if the elements in α are non-zero, $\mathbf{E_5}$ is nonsingular. Although α containing zero does not imply that $\mathbf{E_5}$ is singular, it is a necessary condition of the singularity of $\mathbf{E_5}$. It means that if α contains zero, the matrix $\mathbf{E_5}$ might be singular. Sequentially, if the elements in α are too close to zero, the matrix $\mathbf{E_5}$ is more likely to be nearly singular, thus numerical stability problems might be encountered.

[32] provides a parameter *per* to improve numerical stability when inserting knots to knot vectors of interpolation curves, which can guarantee that there exists a set α whose elements are not too close to zero. However, [32]'s method only guarantees that α exists, but cannot help us to select B-spline bases to construct α, which is crucial to our algorithm. Next, we provide a method to filter out the bases that are too close to 0.

For a parameter $\bar{u} \in [u_i, u_{i+1})$, if $\bar{u}$ is too close to u_i, then $N_{i,p}(\bar{u}) \to 0$; if $\bar{u}$ is too close to u_{i+1}, then $N_{i-p,p}(\bar{u}) \to 0$. We define an interval $[a, b]$ as following:

$$[a, b] = [u_{i+1} + t_1(u_i - u_{i+1}), \quad u_i + t_2(u_{i+1} - u_i)],$$

where,

$$t_1 = (1-g_1)(1-\delta)+g_1,$$
$$t_2 = (1-g_2)(1-\delta)+g_2,$$
$$g_1 = \frac{u_{i+p}-(u_i+u_{i+1})/2}{u_{i+p}-u_i},$$
$$g_2 = \frac{(u_i+u_{i+1})/2-u_{i-p+1}}{u_{i+1}-u_{i-p+1}},$$

and $\delta \in [0,1]$ is a parameter that controls the flexibility of filtering. If the parameter $\bar{u}$ is in $[u_i, a)$, we filter out $N_{i,p}(\bar{u})$. If the parameter $\bar{u}$ is in (b, u_{i+1}), we filter out $N_{i-p,p}(\bar{u})$. It can be seen that, when $\delta = 0$, then $t_1 = t_2 = 1$, and $[a,b] = [u_i, u_{i+1}]$, the filter will not work. Since $g_1, g_2 \in (\frac{1}{2}, 1)$, if $\delta \in (0,1]$, then $t_1, t_2 \in (\frac{1}{2}, 1)$, $u_i < a < (u_i+u_{i+1})/2 < b < u_{i+1}$. After filtering out the B-spline bases that are too close to 0, the lists $\mathcal{S}_0, \ldots, \mathcal{S}_n$ for each curve and $\mathcal{L}$ can be constructed, thus α will not contain elements that are too close to 0, which may cause numerical instability.

Note that, the larger δ is, the tighter the interval $[a,b]$ will be, which means more B-spline bases may be filtered out. To avoid numerical problems, we prefer a larger δ. However, if δ is too large, the remaining bases may not enough for constructing α: B-spline bases accepted by *per* are filtered out by δ. This problem can be detected by checking if there $\exists \bar{u}_i, i \notin \mathcal{L}$. Reducing δ can solve the problem. In our implementation, we detect if the problem is encountered, and automatically reduce δ until α is guaranteed to be constructed. In Section 4, we test how the two parameters *per* and δ affect our algorithm. The results show that, as *per* decreases, more knots are inserted to $\mathbf{U}$ and $\mathbf{V}$. Accroding to [32], a smaller *per*, not being obvious though, might help to reduce the numerical errors in lofting methods. However, in our experiments, the size of the solving matrix $\mathbf{H}$ in Equation 5 increases as *per* shrinks, leading to decreased solving precision (Figure 9) and local non-smoothness (Figure 10). A larger δ helps improve the interpolation accuracy. Consequently, by default, we set $per = 0.5$ and $\delta = 0.9$.

3.3. Control Point Solving

After the knot vectors $\mathbf{U}, \mathbf{V}$ are generated, the B-spline surface $S(u,v)$ that interpolates the scatter points $\mathbf{Q}$ can be constructed through solving the control points $P_{i,j}$, $i = 0, \ldots, n$, $j = 0, \ldots, m$. To obtain a smooth surface, we adopt thin-plate energy E_s as a fairing term. The control points can be obtained by

$$\begin{aligned} &\min E_s = \iint (S_{uu}^2 + 2S_{uv} + S_{vv}^2) du dv \\ &subject\ to\ S(\bar{u}_i, \bar{v}_i) = Q_i, i = 0, \ldots, s, \end{aligned} \tag{6}$$

which can be solved using Lagrange multiplier method. Since the knot vectors $\mathbf{U}, \mathbf{V}$ guarantees the existence of the interpolation surface, the equality constraints in Formula 6 can be satisfied. By applying the global fairing term, the interpolation surfaces have better quality in smoothness compared with the methods without fairing terms (for instance, MBI method [15] shown in Figure 5 and Figure 6). However, local non-smoothness may happen if the knots are non-uniformly distributed (Figure 9, Figure 12). In Section 4 we show that, for further improvement of surface smoothness, our method can also be adapted to a simple iterative local energy reduction procedure: in each iteration, we calculate the thin-plate energy of each sub-domain $E_{i,j} = \iint (S_{uu}^2 + 2S_{uv} + S_{vv}^2) du dv, u \in (u_i, u_{i+1}), v \in (v_j, v_{j+1})$, and insert knots to the sub-domains which has higher energy: if $\iint S_{uu}^2 du dv > \iint S_{vv}^2 du dv$, then insert $(v_j + v_{j+1})/2$ into $\mathbf{V}$; otherwise, insert $(u_i + u_{i+1})/2$ into $\mathbf{U}$. Then solve the control points using Formula 6. Figure 12 and Figure 14 show that, as the number of inserted knots grows, the smoothness of the surface get improved, while the scattered data points being always interpolated accurately.

The computation of our control point solving requires to solve a large scale linear system. Using Lagrange multiplier method, the size of the coefficient matrix could be $(n \times m + s) \times (n \times m + s)$, where $(m+1) \times (n+1)$ is the number of control points, and $s+1$ is the number of data points. Using LU decomposition, the computation complexity of this stage is $\mathcal{O}((n \times m + s)^3)$, which is the most time consuming part of the whole algorithm (Figure 16). In this sense, to improve the efficiency, it is crucial to reduce the number of control points of the interpolation surfaces, which is one of the main contributions of our knot vector generation method described in Section 3.1.2.

Methods	Models	t	$\#CP$	err
Averaging	Drope	20.86	50×50	2.60×10^{-10}
	Hyperboloid	20.67	50×50	5.66×10^{-11}
	Snail	21.23	50×50	1.26×10^{-10}
MBI	Drope	24.87	2005×2005	6.30×10^{-9}
	Hyperboloid	1.74	259×259	1.87×10^{-23}
	Snail	1.94	259×259	4.49×10^{-22}
Ours	Drope	0.40	10×12	1.18×10^{-12}
	Hyperboloid	0.15	10×9	8.98×10^{-13}
	Snail	0.27	12×13	3.81×10^{-12}

Table 1
The runtime t in s, the number of control points $\#CP$ and the maximal interpolation error err of the interpolation surfaces of Averaging, MBI and our method in Figure 5. The 50 data points for each interpolation surface are randomly sampled from 3 parametric surface models.

4. Results

Our algorithm is implemented in C++, and uses Eigen [8] for linear algebra operations. We run the results on a 2.8GHz Intel Core i7-7700HQ CPU. The implementation and the scripts to reproduce all the results will be released as open-source projects.

We test our algorithm on reconstructing cubic B-spline surfaces with data points randomly sampled from a series of parametric surfaces. We compare with 2 methods which are able to interpolate scattered data points in Figure 5 and Figure 6: *Averaging* method [25] which averages the parameters $\bar{\mathbf{U}}$ to generate knot vectors, and *MBI* [15] which uses multilevel B-spline surfaces to progressively reduce the in-

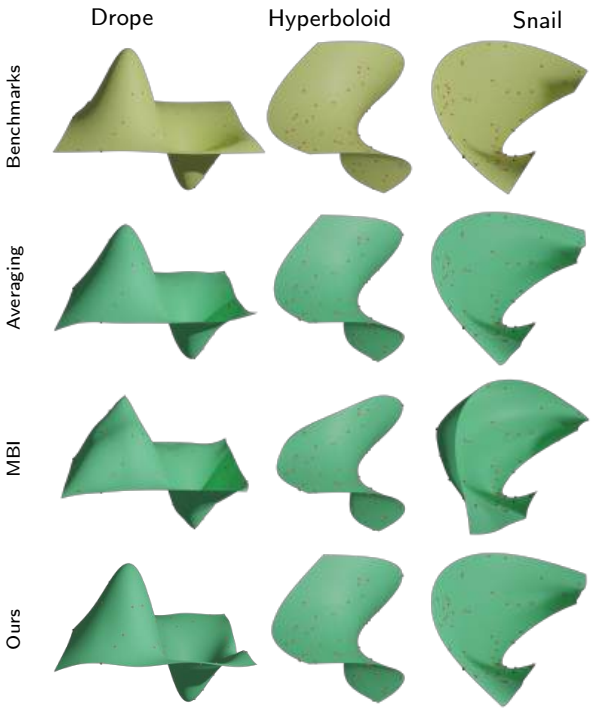

Figure 5: The interpolation results of Averaging (the second row), MBI (the third row) and our method (the last row) on 3 benchmark models (the first row). The red dots are the 50 data points randomly sampled from the 3 benchmark models.

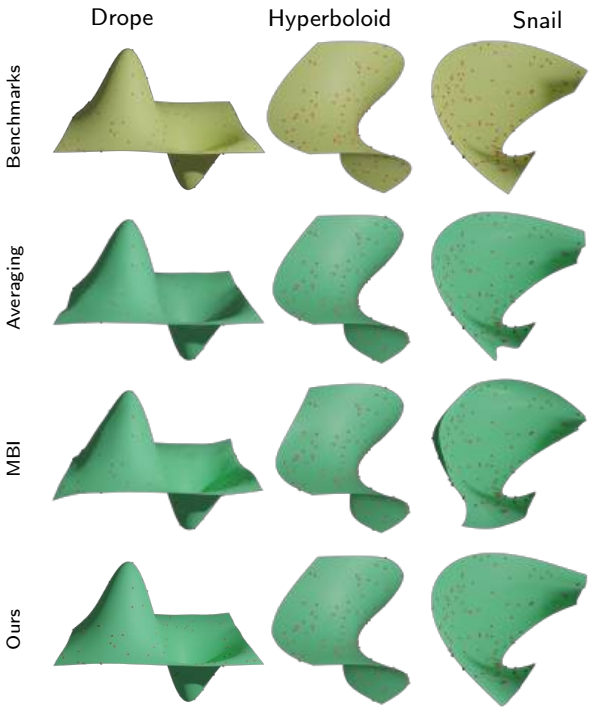

Figure 6: The interpolation results of Averaging (the second row), MBI (the third row) and our method (the last row) on 3 benchmark models (the first row). The red dots are the 100 data points randomly sampled from the 3 benchmark models.

terpolation errors. To conduct a fair comparison, we apply thin-plate energy to handle the degrees of freedom of Averaging method when solving control points. Since the termination of the original MBI method highly depends on the parametrization, which may cause unbounded number of control points, we terminate MBI when the maximal interpolation error err=max $\|S(\bar{u}_i, \bar{v}_i) - Q_i\|$ $(i = 0, \ldots, s)$ is less than 1×10^{-8}. The comparison results are listed in Table 1 and Table 2. It can be seen that, as the number of data points grows, it takes a longer runtime for the 3 methods to interpolate the surfaces. Our algorithm generates interpolation surfaces with fewer control points, thus leading to less computation time compared with the other two methods. MBI method has a very high interpolation accuracy, while being efficient on some of the data sets (Hyperboloid and Snail, 50 data points), but it requires an extremely large number of control points to reach the target interpolation accuracy. And Figure 5 and Figure 6 show that, the surfaces generated by MBI method are less smooth and contains more wiggles compared with the other two methods.

In Figure 1 and Figure 7, we also show the interpolation results of the 100 data points sampled from Drop, Hyperboloid and Snail using [32]'s knot vector generation. As a lofting method, [32] requires the data points distributed in rows, and may not meet the requirements of scattered data points interpolation, thus we apply the control point solving method described in Section 3.3 to handle the degrees of freedom. Since the length of two knot vectors are unbalanced, some of the surfaces generated by [32] are highly non-smooth and folded. The results in Figure 7 show that the data points are interpolated accurately, but the smoothness is worse compared with our results shown in Figure 5 and Figure 6 (except for the Hyperboloid model). The number of control points, the maximal interpolation error and the runtime for each interpolation surface are listed in Table 2

Figure 7: The 100 data points (red dots) sampled from Hyperboloid model (left) and Snail model (right) and the corresponding interpolation surfaces (green surfaces) using the knot vector generation algorithm of [32]. The detailed information of the interpolation surfaces is listed in Table 2

Compared with interpolation methods, approximation methods can reconstruct surfaces out of scattered data points while having a better quality in smoothness. Kineri et al. [14] introduced an iterative geometric approximation (IGA) algorithm which can approximate scattered 3D data points. Figure 8 show the three approximation results of 100 data points sampled from Drop, Hyperboloid and Snail model us-

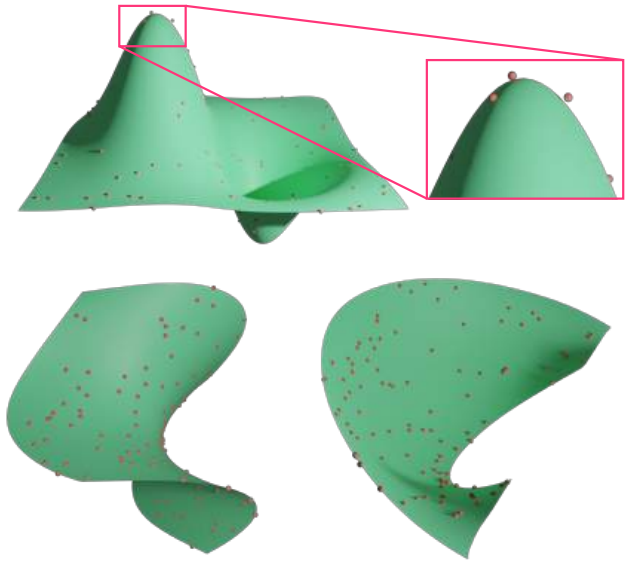

Figure 8: The 100 data points (red dots) sampled from Drop (top), Hyperboloid (bottom left) and Snail (bottom right), and the corresponding approximation surfaces (green surfaces) using IGA [14]. The closeup shows that, IGA allows deviation of the surface from the data points. The detailed information of the approximation surfaces is listed in Table 2.

Methods	Models	t	$\#CP$	err
Averaging	Drope	39.28	100×100	6.16×10^{-10}
	Hyperboloid	36.91	100×100	2.96×10^{-9}
	Snail	37.50	100×100	1.07×10^{-7}
MBI	Drope	54.16	2051×2051	2.69×10^{-11}
	Hyperboloid	18.41	1027×1027	2.03×10^{-11}
	Snail	8.81	515×515	3.75×10^{-23}
Lofting	Drope	0.66	100×4	3.64×10^{-9}
	Hyperboloid	0.64	98×4	3.38×10^{-9}
	Snail	0.65	100×4	1.33×10^{-8}
Ours	Drope	0.63	18×18	1.69×10^{-11}
	Hyperboloid	0.58	18×17	3.79×10^{-12}
	Snail	0.52	16×18	1.69×10^{-11}
IGA	Drope	0.90	10×10	0.055
	Hyperboloid	0.89	10×10	2.0×10^{-3}
	Snail	0.86	10×10	3.1×10^{-3}

Table 2
The runtime t in s, the number of control points $\#CP$ and the maximal fitting error err of the fitting results of the four interpolation methods: Averaging, MBI, Lofting and our method, and one approximation method IGA. The 100 data points for each fitting surface are randomly sampled from the 3 parametric surface models.

ing [14]. The number of iterations for each surface is 200, and we set the number of control points $\#CP = 10 \times 10$. The closeup of The Drop model shows that as an approximation method, IGA allows deviation of the surface from the data points, while being able to use fewer control points and having a good quality in smoothness. The more detailed information of the three approximation surfaces is listed in Table 2. In this sense, it reveals one of our advantages that our method can generate a globally smooth surface while guaranteeing the data points being interpolated accurately.

By applying our weight calculation method when selecting bases to construct α, the number of control points of the interpolation surfaces and the runtime can be effectively reduced. With the same sampling points and their parametrizations as Figure 5 and Figure 6, we test two non-weighted strategies in Table 3 other than our weight calculation–α construction method shown in Table 1 and Table 2: *No Selecting* strategy conservatively takes all the k_i which satisfies that $\mathcal{L}_r(k_i) = i, \forall i \in \{0, \dots, s\}$ to construct α, instead of selecting one k_i for each i. *Naive Selecting* strategy simply select the smallest k_i where $\mathcal{L}_r(k_i) = i$ to construct α. The results in Table 3 show that, both of the two non-weighted strategies produce more control points and take a longer runtime compared with our method, with no obvious differences in maximal interpolation errors.

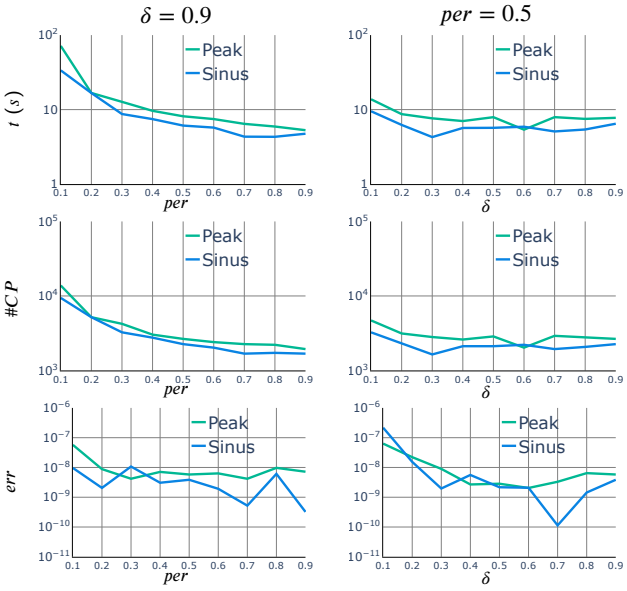

Figure 9: The log plots of runtime in s (top), the number of control points (middle) and the maximal interpolation errors (bottom) for different *per* (left, $\delta = 0.9$) and different δ (right, $per = 0.5$). The 500 data points for each interpolation surface are randomly sampled from 2 models: Peak and Sinus.

Our algorithm has two user-controlled parameters *per* and δ to control the flexibility of knot insertion and to improve the numerical stability of control point solving. The effects of the two parameters *per* and δ on runtime, the number of control points and the maximal interpolation errors are presented in Figure 9. Figure 10 and Figure 11 show some of the interpolation surfaces with different parameter settings. The 2 benchmark parametric surfaces we use to sample the 500 data points are very complicated and have many local minima.

per is a parameter provided by [32]'s method to control the flexibility of knot insertion. As *per* decrease, the number of control points of the surface increases. Although a smaller *per* avoids the elements in α being too close to 0, the increased number of control points leads to larger linear systems for solving control points, which reduces the solving precision: the *err* increases from 7.20×10^{-9} and 5.73×10^{-8} to 3.22×10^{-10} and 9.63×10^{-9} for Peak and Sinus, respectively, as *per* decrease from 0.1 to 0.9 (Figure 9, row 3, col-

		50 data points			100 data points		
Strategies	Models	t	$\#CP$	err	t	$\#CP$	err
No Selecting	Drope	1.38	23×24	2.98×10^{-11}	3.00	35×35	1.39×10^{-10}
	Hyperboloid	1.24	22×22	8.13×10^{-12}	3.74	40×40	1.17×10^{-10}
	Snail	1.56	27×25	8.99×10^{-11}	3.59	38×40	1.28×10^{-10}
Naive Selecting	Drope	0.86	19×20	1.73×10^{-11}	2.63	34×33	3.49×10^{-10}
	Hyperboloid	0.38	15×14	6.42×10^{-12}	1.52	26×28	4.99×10^{-11}
	Snail	1.00	21×23	3.80×10^{-10}	1.93	29×30	6.10×10^{-10}

Table 3
The runtime t in s, the number of control points $\#CP$ and the maximal interpolation error err of the two other strategies of α generation. Compared with our method in Table 1 and Table 2, both of these two simple strategies produce more control points and require a longer runtime.

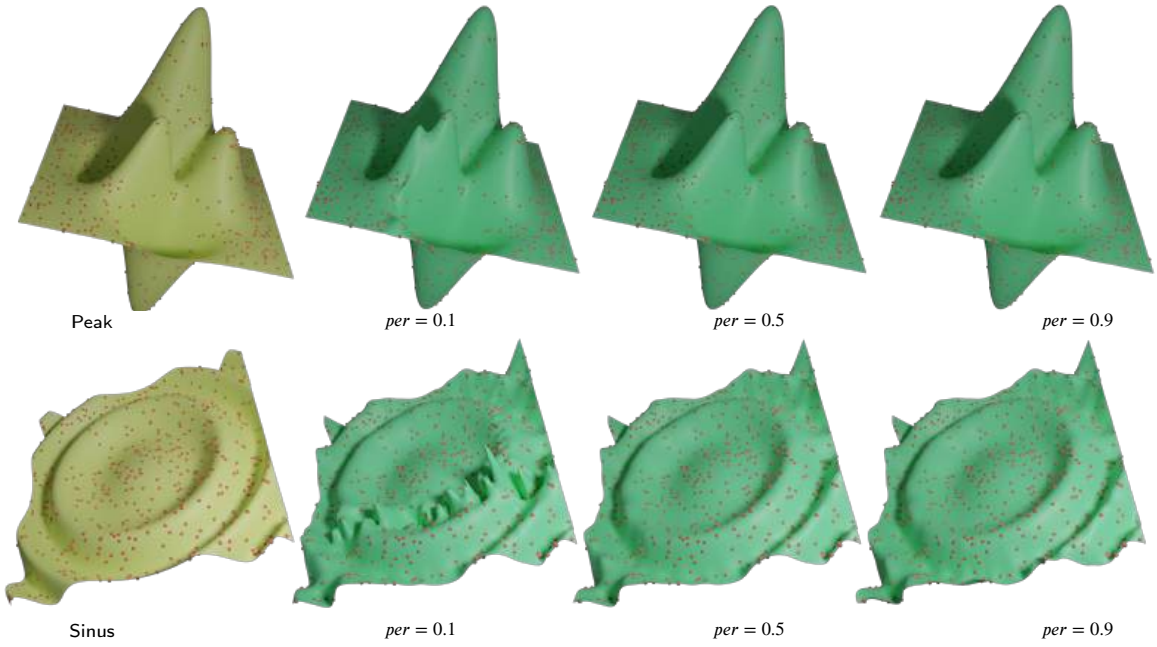

Figure 10: The interpolation surfaces for $per = 0.1$, $per = 0.5$ and $per = 0.9$. $\delta = 0.9$. The 500 data points are sampled from 2 benchmark models: Peak (top left) and Sinus (bottom left). It can be seen that, when interpolating Sinus, $per = 0.1, \delta = 0.9$ (the second row, the second column), the surface has "spikes" which implies less local smoothness.

umn 1). By default, we set $per = 0.5$.

δ is a parameter to discard B-spline bases that are too close to 0 when constructing α. If δ is set to 0, then no B-spline bases will be discarded. With a larger δ, there will be less risk of encountering numerical problems. Thus a larger δ is preferred. Figure 9 shows that, as δ increases from 0.1 to 0.9, the err decreases from 6.33×10^{-8} and 2.17×10^{-7} to 5.81×10^{-9} and 3.86×10^{-9} for Peak and Sinus, respectively. Although a large δ may cause situations that α cannot be generated (aforementioned in Section 3.2), our implementation can initially take a large δ as input (by default 0.9), and automatically reduce δ if α cannot be constructed.

Moreover, from Figure 10, it can be observed that when per is too small, such as $per = 0.1$, some regions of the Sinus interpolation surface are highly non-smooth and have "spikes". The reason why it happens is that there are too many knots inserted into knot vectors, and locally there are too many knots gathering together (*e.g.,* too many $\mathbf{U}$ knots gathering in a small interval $[0.4, 0.45]$). In this case, since our algorithm applies a global fairing term, the local fairing energy $E_{i,j} = \iint (S_{uu}^2 + 2S_{uv} + S_{vv}^2) du dv, u \in [u_i, u_{i+1}], v \in [v_j, v_{j+1}]$, could be extremely large even if the global fairing energy reaches minimum. We reproduce this problem with a simple example (Figure 12, top left): we directly take the surface in Figure 11, row 2, column 4 (500 Sinus data points with $per = 0.5, \delta = 0.9$), which is smooth and has no spiky features, and add 20 knots between 0.4 and 0.41 into its $\mathbf{V}$ knot vector. It can be observed that non-smooth parts appear, which implies higher local fairing energy.

As aforementioned, since our method uses thin-plate energy as a global fairing term, the local fairing energy could be large thus leads to non-smoothness. Increasing per can help to avoid this problem, since the knots will be fewer and sparser (Figure 10). For further improvement, we show in Figure 14 and Figure 12, that our method can be adapted to iterations of knot insertion to reduce local energy: by iteratively inserting knots to intervals $(u_i, u_{i+1})\times(u_j, u_{j+1})$ which have largest local thin-plate energy $\mathrm{ME} := \max_{i,j} E_{i,j}$ and

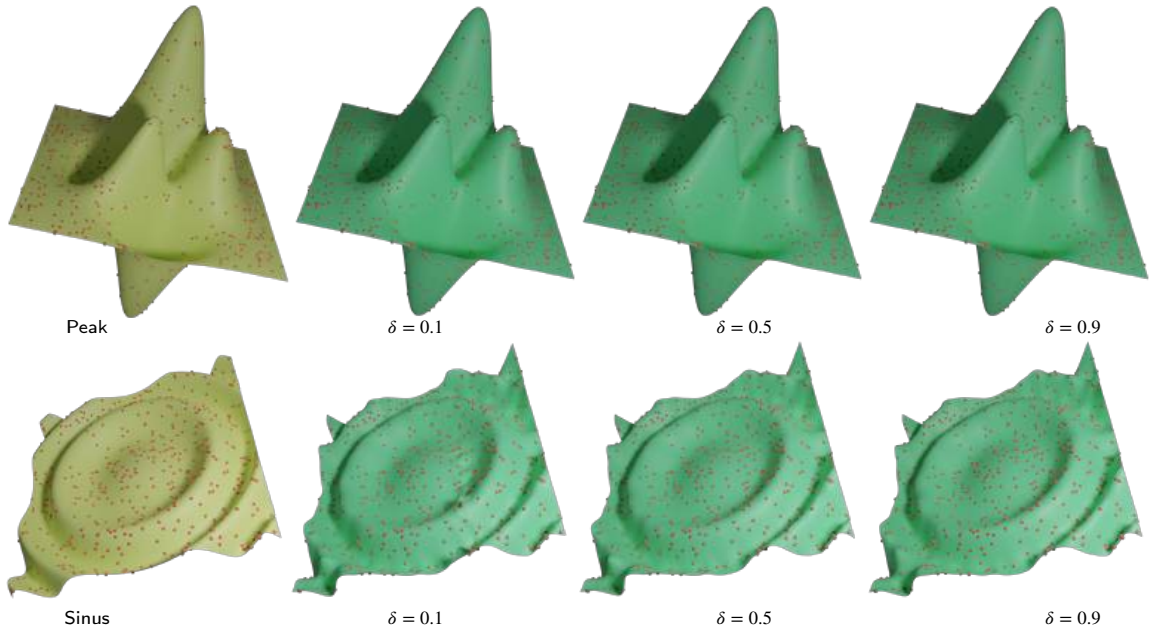

Figure 11: The interpolation surfaces for $\delta = 0.1$, $\delta = 0.5$ and $\delta = 0.9$. $per = 0.5$. The 500 data points are sampled from 2 benchmark models: Peak (top left) and Sinus (bottom left).

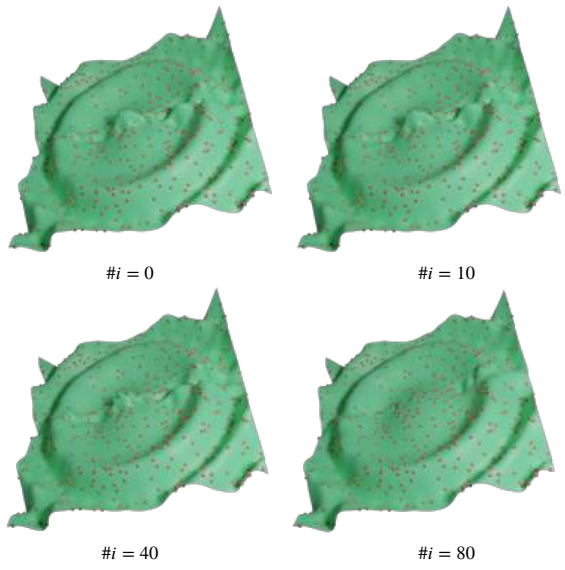

Figure 12: After adding 20 extra knots into $\mathbf{V}$, the surface in Figure 11, row 2, column 4 will have spiky features (top left). By iteratively inserting knots into regions which have higher local thin-plate energy, the spiky features can be progressively smoothed. $\#i$ is the number of knots inserted into the knot vectors.

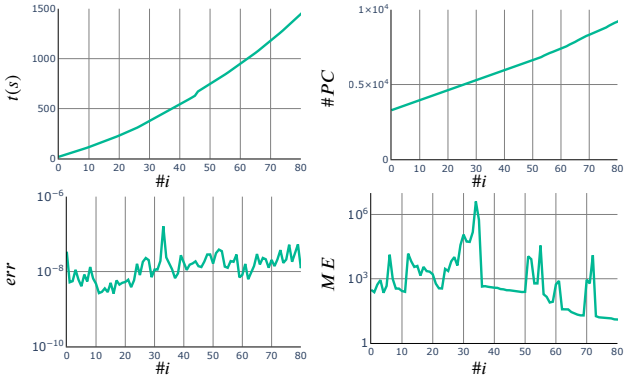

Figure 13: The runtime in s (top left), the number of control points (top right), the maximal interpolation error (bottom right) and the maximal local energy (bottom left) for different number of knots inserted into the knot vectors of the B-spline surface (Figure 12, top left) which has spiky areas.

solve the control points through Formula 6, the surface can be progressively smoothed, while always keeping the data points interpolated. We remark that, this can be done only if the interpolation surface exists, which is guaranteed by our knot vector generation algorithm. In Figure 12 and Figure 13, we show that the smoothness of the spiky regions is improved as the number of inserted knots $\#i$ increase from 0 to 80, and the maximal local energy ME decreases from 303.10 to 12.61. In Figure 14, the 30 data points are sampled from Sinus, and the surface (top left) is extremely spiky because the number of data points is too few to bound the shape of the interpolation surface. By inserting knots, the surface becomes more flatter and smooth, with ME decreasing from 3.36×10^5 to 4.25×10^2 (Figure 15).

Figure 16 shows the performance of our algorithm when interpolating different number of data points. `Time Knot` is the runtime for generating knot vectors, `Time Solve` is the runtime of solving control points, and, `Time All` is the total runtime which is the sum of `Time Solve` and `Time Knot`. Figure 16 shows that, as the number of data points grows, the number of control points and the runtime also increases. The most time consuming part of our algorithm is solving the control points, which requires constructing and solving

a large scale linear system. For the 10 different numbers of data points (from 50 to 500) we test in Figure 16, the ratio of `Time Solve` to `Time All` varies from 89.5% (50 data points) to 97.4% (300 data points), with the average 93.1%. The shape of the surfaces converges to the shape of the original models (Figure 17) as the number of data points grows. As the scale of the linear system enlarged with the number of data points, under floating point calculation, the interpolation accuracy will decrease. In our experiments (Figure 16), *err* increases from 6.78×10^{-13} (50 data points) to 3.86×10^{-9} (500 data points).

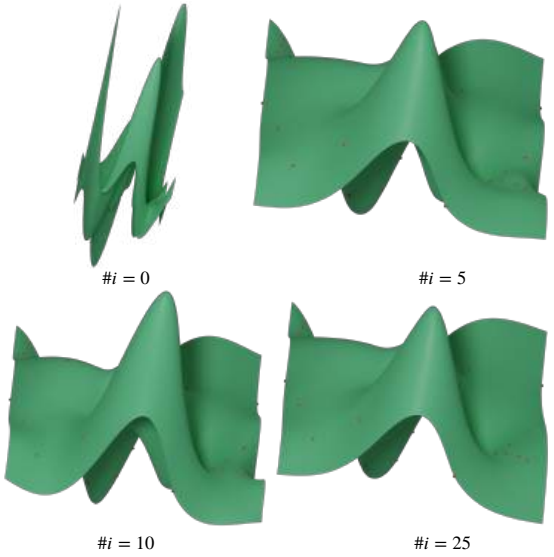

Figure 14: Example of interpolating 30 data points sampled from Peak. The generated surface is highly non-smooth (top left). As the number of inserted knots #i increases from 0 to 25 (top right, bottom left and bottom right), the maximal local energy decreases from 3.36×10^5 to 4.25×10^2, thus leading to improvement of surface smoothness.

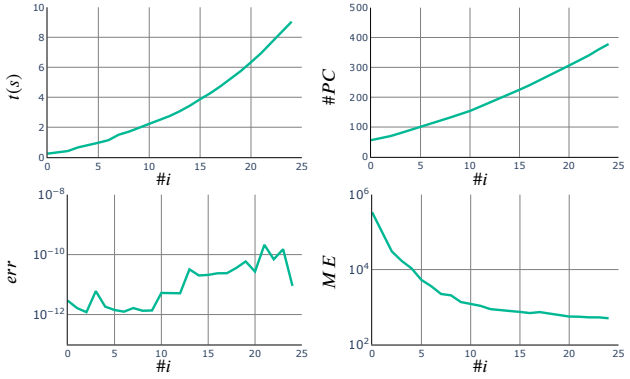

Figure 15: The runtime in s (top left), the number of control points (top right), the maximal interpolation error (bottom left) and the maximal local energy (bottom right) for different number of knots inserted into the knot vectors of the B-spline surface (Figure 14, first) which interpolates 30 data points sampled from Peak.

Unlike approximation algorithms which can filter out high-frequency noise of the input data [14, 32], our method is sensitive to noise. Especially, our algorithm tends to interpolate the data points with as fewer control points as possible. We add random noise to the normal direction of the Drop model, and randomly sample 100 data points from the model. Figure 18 (first row) shows the noisy Drop model viewed in two different views. The scale of the noise is up to 0.03, which is about 0.5% of the diagonal length of the axis-aligned bounding box of the Drop model. Figure 18 (second row) shows that the interpolation surface has wiggles, especially in the regions where the sample points are gathered, and the data points are interpolated accurately (with $\#\boldsymbol{CP} = 21 \times 20$ and $err = 2.08 \times 10^{-9}$). We can improve the smoothness of the interpolation surface through knot insertion. Figure 19 shows the result after inserting 15 knots into the knot vectors of the surface in Figure 18 (second row). There are fewer wiggles after knot insertion, with the $\#\boldsymbol{CP} = 25 \times 32$ and $err = 2.55 \times 10^{-9}$. The closeup in Figure 19 shows a non-smooth region on the surface, where there are 3 sample points very close to each other, and the added noise caused the fierce variation of the surface shape. Similar to Figure 18, in Figure 20 we add random noise (up to 0.03) to the Drop and Hyperboloid model, but keep the sampled parameters not too close to each other (the maximal difference between any two u parameters or v parameters is no more than 0.03). Compared with the results in Figure 18, the smoothness of the interpolation results in Figure 20 get improved since the randomly noisy data points are not too close to each other, thus no large variation happens in small surface regions.

Figure 21 shows examples of using our method to interpolate data points which are distributed in grids. There are 11×11 sampled points in Figure 21 (left), and 9×30 sampled points in Figure 21 (right). The numbers of required control points are also 11×11 and 9×30, respectively, reaching the lower bounds of the numbers of the control points required. For these two examples, Averaging method [25] and Lofting method [32] generate exactly the same surfaces as using our method, with no obvious differences in performance (All the three methods take around 0.20 seconds and 0.50 seconds to interpolate the data points in Figure 21 (left) and Figure 21 (right), respectively).

Figure 22 shows two examples of interpolating data points in rows sampled from Peak model and Snail model. In Figure 22 (left), the data points are sampled in 9 rows, and in each row, we randomly sample 30 points. In Figure 22 (right), there are 6 rows, and in each row there are 20 randomly sampled data. The interpolation surfaces contain 9×66 and 6×36 control points, respectively, and the $err = 3.39 \times 10^{-10}$ and 7.51×10^{-11}, respectively.

Figure 23 shows an example of interpolating a dataset non-uniformly randomly sampled from the Drop model. The interpolation result and the sampled points are shown in Figure 23 (left). The parameter domain is $[0, 1] \times [0, 1]$. In region $[0.3, 0.5] \times [0.3, 0.5]$, we randomly sample 100 data points, and randomly sample 150 data points outside this region. The parametrization of the sampled points is shown in

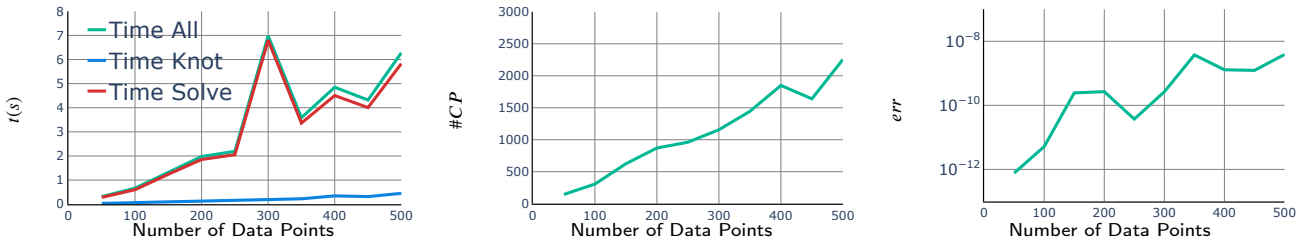

Figure 16: The runtime of constructing knot vectors (`Time Knot`), solving control points (`Time Solve`) and total runtime (`Time All`) for interpolating different numbers of data points are presented in the left plot. The middle and the right plots are the corresponding numbers of control points and the maximal interpolation errors. The data points are sampled from Sinus model.

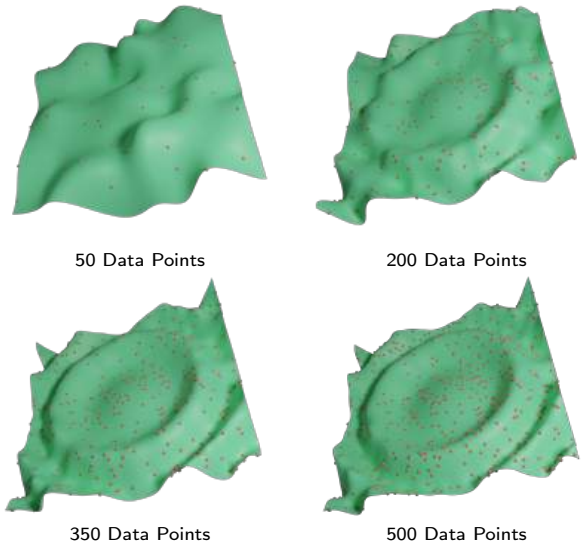

Figure 17: The B-spline surfaces for interpolating different numbers of data points sampled from Sinus model.

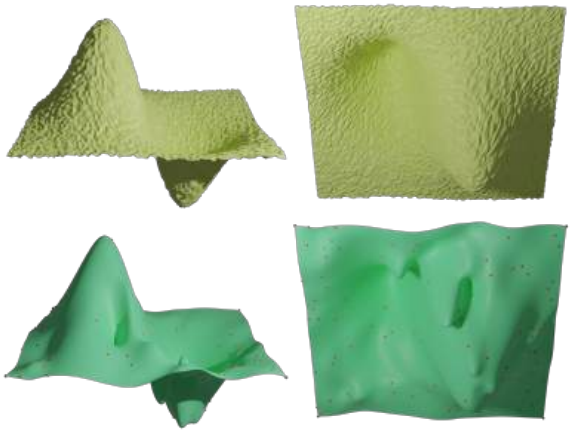

Figure 18: The randomly noisy Drop model in two views (first row) and the corresponding interpolation surface (second row). The scale of the noise is up to 0.5% of the length of the axis-aligned bounding box of the Drop model.

Figure 23 (middle). The green lines in Figure 23 (right) are the inserted knots. It is clear to see that the knots are denser in high-resolution areas, which implies the surface has a better ability of shape control in such areas.

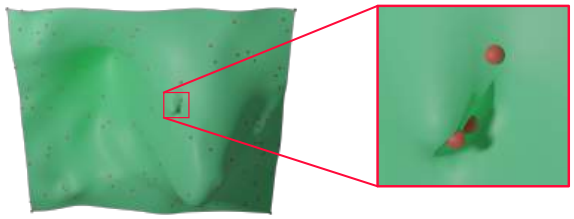

Figure 19: After inserting 15 knots into the interpolation surface in Figure 18 (second row), for most of the parts the smoothness is improved. The closeup shows a non-smooth part where there are 3 sampled data points close to each other.

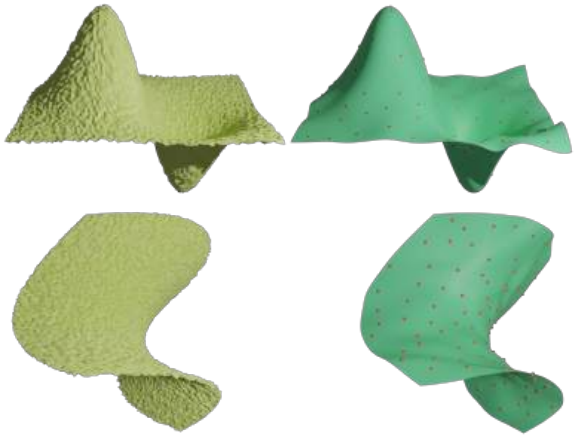

Figure 20: The randomly noisy Drop model (top first), Hyperboloid model (bottom first) and the surfaces (top right, bottom right) interpolating the 100 sampled data points. The *err* are 1.13×10^{-11} and 6.97×10^{-12} and the $\#CP$ are both 20×20.

We apply our method to reconstructing B-spline surface patches from triangular mesh models, each of which is topologically equivalent to a disk. (Figure 24 and Figure 25). The data points for B-spline surface interpolation are the vertices of the meshes, and the parametrization of the vertices of each model is obtained by *Harmonic parametrization* [6] implemented in Libigl [12]. We note that currently our method cannot be applied to those triangular mesh models of arbitrary topology, such as closed surfaces. Using multiple B-spline surface patches to construct models with more com-

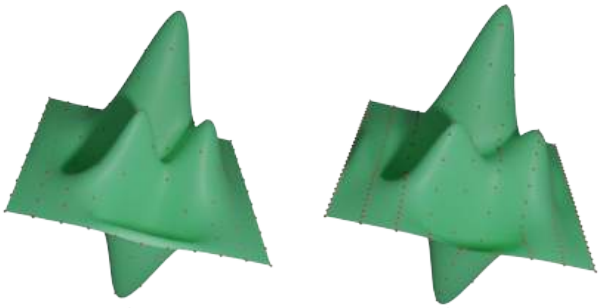

Figure 21: The number of gridded data points (red dots) are 11×11 (left) and 9×30 (right), respectively. The number of control points of the interpolation surfaces (green surfaces) are also 11×11 (left) and 9×30 (right), which reaches the lower bound of the number of control points required.

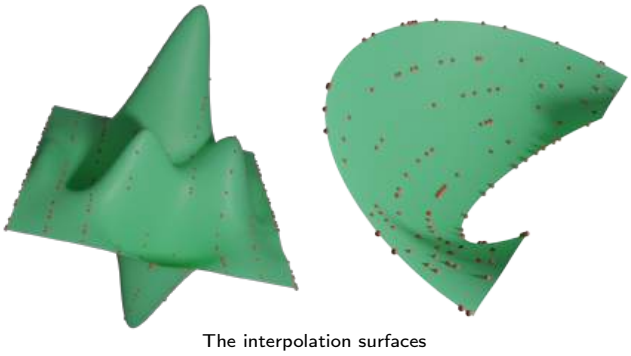

Figure 22: The data points (red dots) sampled from Peak model (left) and Snail model (right) and the corresponding interpolation surfaces (green surfaces). The data points sampled from Peak model are in 9 rows, and in each row the 30 data points are randomly sampled. The data points sampled from Snail model are in 6 rows, and in each row there are 20 randomly sampled data. The $\#CP$ of the two surfaces are 9×66 and 6×36, respectively.

plicated topology while ensuring the continuity is a major challenge, thus we would leave this as a future work. The Mask model (Figure 24, left) contains 1590 vertices. Using our method, the runtime for interpolating is $29.53s$, and the B-spline surface (Figure 24, middle) contains 93×93 control points, with maximal interpolation error 6.30×10^{-9}. We also calculate the Hausdorff distance H_d [1] (Figure 24, right) from the interpolation surface to the original mesh. The maximal H_d is 0.0388. The Tiger model (Figure 25, left) contains 785 vertices. The runtime, the number of control points and the maximal interpolation error of the interpolation surface (Figure 25, middle) are $10.87s$, 62×62 and 1.71×10^{-8}, respectively. The Hausdorff distance H_d from the interpolation surface to the Tiger model is shown in Figure 25, right. The maximal H_d is 1.20. It is clear to see that the H_d of Tiger model is higher than that of Mask. This is because the Tiger model has fewer vertices, while the scale of the mesh being larger than Mask (the lengths of the diagonals of the axis-aligned bounding boxes of Tiger and Mask are 102.16 and 12.27, respectively). It should also be noticed that, as an interpolation algorithm, our method often requires larger quantity of control points than that of the input data points, especially when the data points are scattered and have no obvious gridded structures. For the Mask and Tiger model, the ratios of number of control points to the number of data points are around 5.

Approximation algorithms allow reconstructing surfaces out of scattered data points using fewer data points. In Figure 26, we use the approximation algorithm IGA [14] to fit the vertices of Mask and Tiger. We set the number of control points close to the number of vertices for each surface (40×40 and 30×30 control points for Mask and Tiger which have 1590 and 785 vertices, respectively). With fewer control points than our interpolation results shown in Figure 24 and Figure 25, the approximation surfaces have good quality in smoothness, while the trade-off is the deviation of the surfaces from the input data points: By setting the number of iterations as 200, the runtime for the two surfaces are 33.14 and 10.44 seconds, which are comparable with our results. The maximal approximation errors of the two surface on the mesh vertices (0.053 and 1.20) are significantly larger than the maximal interpolation errors of our results ($\sim 1 \times 10^{-8}$), which reveals the advantage of our interpolation algorithm.

5. Conclusion

In this paper, we introduce a new method to interpolate scattered 3D data points with a single B-spline surface patch which is globally smooth. We validate our algorithm with parametric surface data points interpolation and also demonstrate the possibility of applying our method on constructing B-spline surfaces from triangular meshes. The proposed method guarantees the existence of the interpolation surfaces, while often using fewer control points when interpolating scattered data points, and being efficient compared with traditional methods. We also introduce a user-specified parameter δ to improve the numerical stability.

Since our algorithm uses a global fairing term, local non-smoothness may happen when knots are highly non-uniform. Although our simple energy detection–knot insertion procedure can improve the local smoothness while always keeping the data points interpolated accurately, the calculation is somehow time consuming, and the increased size of linear system may lead to decreased interpolation accuracy. This issue might be solved by employing superior fairing energy, or proper post-processing steps. Our method constructs a single B-spline patch to interpolate the scattered data points, thus cannot be applied to those triangular mesh models of arbitrary topology. We would like to leave these problems as future work.

Acknowledgements

This work was partially supported by the National Key Research and Development Program of China (2020YFA0713700), the National Natural Science Foundation of China (12171023, 12001028 and 62172415), and

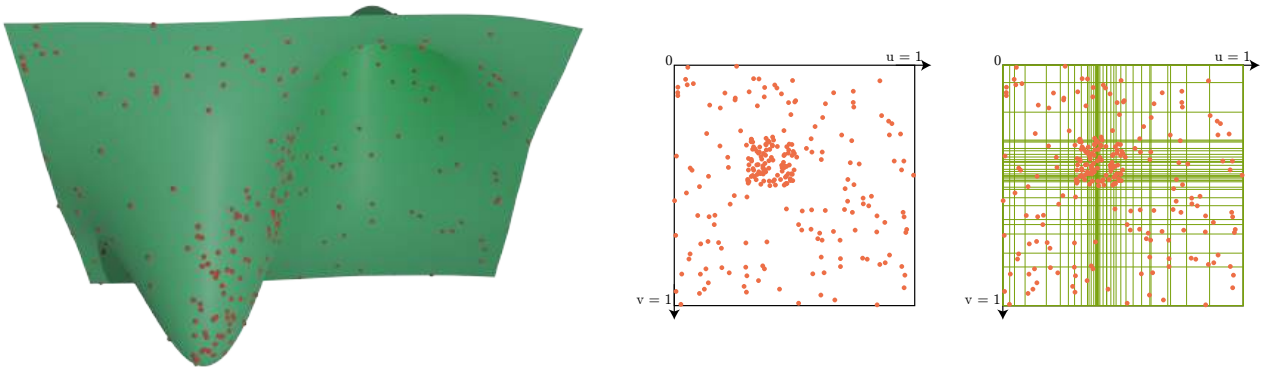

Figure 23: Left, the data points (red dots) non-uniformly sampled from Drop and the interpolation result (green surface). Middle, the parametrization of the sampled points. Right, the generated knots. The maximal interpolation error $err = 4.33 \times 10^{-9}$, the runtime $t = 3.10s$, and the number of control points $\#CP = 37 \times 36$.

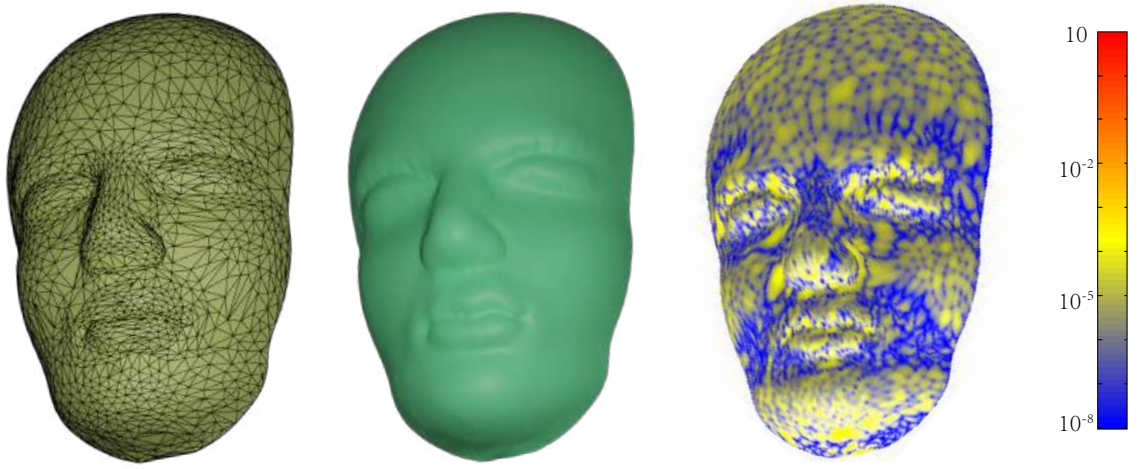

Figure 24: The Mask model which contains 1590 vertices (left), B-spline interpolation surface (middle) and the Hausdorff distance H_d (right) from the B-spline surface to the mesh.

the Open Research Fund Program of State key Laboratory of Hydroscience and Engineering (sklhse-2022-D-04).

A. Proof of Proposition 1

Proof We take a few steps to prove the proposition:

1. For $\forall v = \bar{v}_r$, the curve $C(\bar{v}_r) = \sum_{i=0}^{n} N_{i,p}(u)C_i(\bar{v}_r)$ interpolates the points $\{Q_t|\bar{v}_t = \bar{v}_r\}$ if Condition 1 is true.
2. If Condition 1 and Condition 2 are true, then the matrix $\mathbf{H}$ in Equation 5 is row full rank, thus the data points $\mathbf{Q}$ can be interpolated.

Without loss of generality, we fix $v = \bar{v}_r$. Since for each $\bar{v}_l = \bar{v}_r, l \neq r$, there exists $k_l \neq k_r$, $s.t.$ $N_{k_l,p}(\bar{u}_l) \neq 0, N_{k_r,p}(\bar{u}_r) \neq 0$ (Condition 1), therefore, according to [32, 4, 3], the curve $C(\bar{v}_r)$ interpolates the points $\{Q_t|\bar{v}_t = \bar{v}_r\}$. We assume $t = 0, \ldots, b, b \leq n$. Moreover, the *collocation matrix* $\mathbf{A} = \{a_{i,j}\}$ $(a_{i,j} = N_{j,p}(\bar{u}_i), j = 0, \ldots, n, i = 0, \ldots, b)$ of the curve $C(\bar{v}_r)$ is row full rank. We can construct a $(b+1) \times (b+1)$ matrix $\mathbf{A}'$ by selecting $b+1$ columns of $\mathbf{A}$: $\mathbf{A}' = \{a'_{i,j}\}, a'_{i,j} = N_{k_j,p}(\bar{u}_i), i, j = 0, \ldots, b, k_j < k_{j+1}, N_{k_j,p}(\bar{u}_i) \neq 0$. It can be seen that the selected $\{N_{k_j,p}(\bar{u}_i)\}$ $(j = 0 \ldots b)$ satisfy Condition 1, and iterate over different $\bar{v}_r$, the set α can be constructed. According to [3], the matrix $\mathbf{A}'$ is invertible since $a'_{i,i} \neq 0$ for $\forall i \in \{0, \ldots, b\}$, which means, if fix the other $n - b$ control points for any values, by assigning proper values to the $b+1$ control points $C_{k_i}(\bar{v}_r), i = 0, \ldots, b$, the curve $C(\bar{v}_r)$ always interpolates the $b+1$ points $\{Q_t|\bar{v}_t = \bar{v}_r\}$. So, step 1 is proved.

For convenience, we assume there are $l+1$ different v parameter values $\bar{v}_0, \ldots, \bar{v}_l$. Assemble the collocation matrices of curves $C(\bar{v}_r), r = 0, \ldots, l$ together, the surface interpola-

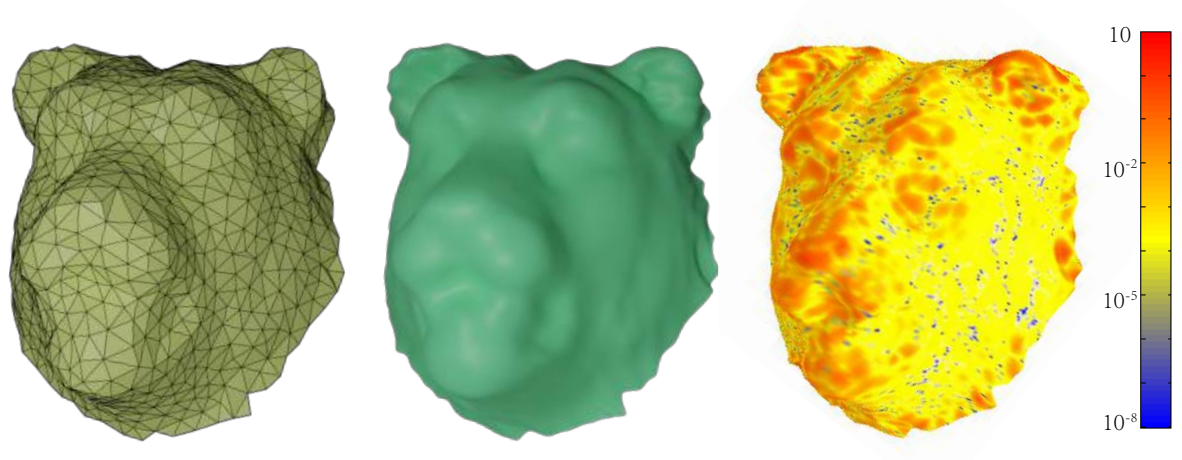

Figure 25: The Tiger model which contains 785 vertices (left), B-spline interpolation surface (middle) and the Hausdorff distance H_d (right) from the B-spline surface to the mesh.

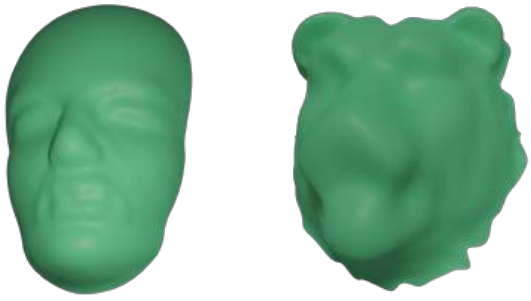

Figure 26: The approximation results of Mask model (left) and Tiger model (right) using IGA [14]. For the Mask model, the number of control points is 40×40, the runtime is $33.14s$, and the maximal approximation error is 0.053; For the Tiger model, the number of control points is 30×30, the runtime is $10.44s$, the maximal approximation error is 1.2.

tion problem can be described as $\mathbf{Q} = \mathbf{EC}$, where

$$\mathbf{E} = \begin{bmatrix} \mathbf{A_0} & & & \\ & \mathbf{A_1} & & \\ & & \ddots & \\ & & & \mathbf{A_l} \end{bmatrix}, \ \mathbf{C} = \begin{bmatrix} C_0(\bar{v}_0) \\ \vdots, \\ C_n(\bar{v}_0) \\ \vdots, \\ C_0(\bar{v}_l) \\ \vdots, \\ C_n(\bar{v}_l) \end{bmatrix},$$

$\mathbf{A_0}, \ldots, \mathbf{A_l}$ are the collocation matrices of the curves $C(\bar{v}_0)$, $\cdots, C(\bar{v}_l)$. If Condition 1 is satisfied, then, for each curve $C(\bar{v}_r)$, $\mathbf{A_r}$ is row full rank. So, $\mathbf{E}$ is row full rank, $rank(\mathbf{E}) = s + 1$. Define α as in Condition 1, we interchange the rows of $\mathbf{C}$ by elementary row transformation $\mathbf{C} = \mathbf{LC_1}$ to make $C_{k_i}(\bar{v}_i)$ as the first $s + 1$ elements of $\mathbf{C}$, $\mathbf{L}$ is an elementary row transformation matrix which contains only interchange operations. Then, $\mathbf{Q} = \mathbf{EC} = \mathbf{ELC_1} = \mathbf{E_1C_1}$, $\mathbf{E_1}$ is the result after applying elementary column transformation to $\mathbf{E}$. From the 1st step of our proof, $\mathbf{E_1}$'s first $s + 1$ columns are linear independent, thus $\mathbf{E_1} = [\mathbf{E_2} \quad \mathbf{E_3}]$, where $\mathbf{E_2}$ is an invertible $(s + 1) \times (s + 1)$ matrix.

To study the structure of $\mathbf{C_1}$, we define another elementary row transformation matrix $\mathbf{L_1}$, $\mathbf{C} = \mathbf{L_1C_2}$,

$$\mathbf{C_2} = [C_0(\bar{v}_0), \cdots, C_0(\bar{v}_l), \cdots, C_n(\bar{v}_0), \cdots, C_n(\bar{v}_l)]^\top = \mathbf{NP},$$

where $\mathbf{P}$ is defined as in Equation 5, and $\mathbf{N}$ is the assembly of a series of collocation matrices $\mathbf{N_1}$:

$$\mathbf{N} = \begin{bmatrix} \mathbf{N_1} & & & \\ & \mathbf{N_1} & & \\ & & \ddots & \\ & & & \mathbf{N_1} \end{bmatrix}, \mathbf{N_1} = \begin{bmatrix} N_{0,q}(\bar{v}_0) & \ldots & N_{m,q}(\bar{v}_0) \\ \vdots & \ddots & \vdots \\ N_{0,q}(\bar{v}_l) & \ldots & N_{m,q}(\bar{v}_r) \end{bmatrix}.$$

The ith $\mathbf{N_1}$ matrix in N corresponds to the curve $C_i(v)$. Define β as in Condition 2, then from Condition 2 and step 1 of our proof, for curve $C_r(v)$, if there are $k + 1 \geq 1$ different values $i_0, \cdots, i_k \in \{0, \cdots, s\}$, $N_r(\bar{u}_{i_0}), \cdots, N_r(\bar{u}_{i_k}) \in \alpha$, then the rank of the collocation matrix of the curve is $k + 1$. Then, we select $s + 1$ rows which corresponds to $C_{k_i}(\bar{v}_i)$ $(N_{k_i,p}(\bar{u}_i) \in \alpha)$ from $\mathbf{N}$, these $s + 1$ rows are linear independent. We apply a row-interchange operation on $\mathbf{N}$ to make these rows as the first $s + 1$ rows of a matrix $\mathbf{N_2}$: $\mathbf{N} = \mathbf{L_2N_2}$, where $\mathbf{L_2}$ is an elementary row transformation matrix which contains only row-interchange operations, $\mathbf{N_2}$ is a matrix whose first $s + 1$ rows are full rank. We denote $\mathbf{N_2} = [\mathbf{N_3} \quad \mathbf{N_4}]^\top$, where $\mathbf{N_3}$ has $s + 1$ rows which are linear independent to each other. Since

$$\begin{aligned} \mathbf{Q} &= \mathbf{EC} = \mathbf{ELC_1} = \mathbf{E_1C_1}, \\ &= \mathbf{EC} = \mathbf{EL_1C_2} = \mathbf{EL_1NP} = \mathbf{EL_1L_2N_2P}, \end{aligned}$$

and $\mathbf{L_1}, \mathbf{L_2}$ are row-interchange matrices, thus $\mathbf{L} = \mathbf{L_1L_2}$, $\mathbf{E_1} = \mathbf{EL_1L_2} = [\mathbf{E_2} \quad \mathbf{E_3}]$, and $\mathbf{C_1} = \mathbf{N_2P}$.

Since $\mathbf{N_3}$ is row full rank, we can apply elementary column transformation to $\mathbf{N_2}$: $\mathbf{N_2} = \mathbf{N_5R}$, where

$$\mathbf{N_5} = \begin{bmatrix} \mathbf{N_6} & 0 \\ \mathbf{N_7} & 0 \end{bmatrix}.$$

$\mathbf{N_6}$ is an invertible $(s+1)\times(s+1)$ matrix, $\mathbf{R}$ is an elementary transformation matrix. We do elementary row transformation to $\mathbf{N_5}$, $\mathbf{N_5} = \mathbf{L_3}\mathbf{N_8}$, where

$$\mathbf{N_8} = \begin{bmatrix} \mathbf{N_6} & 0 \\ 0 & 0 \end{bmatrix}.$$

Here we have $\mathbf{Q} = \mathbf{E_1}\mathbf{L_3}\mathbf{N_8}\mathbf{RP}$. We define $\mathbf{E_4} = \mathbf{E_1}\mathbf{L_3} = [\mathbf{E_5} \quad \mathbf{E_6}]$. It should be noticed that $\mathbf{E_1} = [\mathbf{E_2} \quad \mathbf{E_3}]$, the elementary column transformation $\mathbf{L_3}$ will not change $\mathbf{E_2}$ in $\mathbf{E_1}$, thus, $\mathbf{E_5} = \mathbf{E_2}$, which is a full rank $(s+1)\times(s+1)$ matrix.

Consequently,

$$\mathbf{Q} = \mathbf{HP} = \begin{bmatrix} \mathbf{E_5} & \mathbf{E_6} \end{bmatrix} \begin{bmatrix} \mathbf{N_6} & 0 \\ 0 & 0 \end{bmatrix} \mathbf{RP} = \begin{bmatrix} \mathbf{E_5}\mathbf{N_6} & 0 \end{bmatrix} \mathbf{RP}.$$

Since $\mathbf{E_5}, \mathbf{N_6}, \mathbf{R}$ are invertible matrices, $\mathbf{H} = [\mathbf{E_5}\mathbf{N_6} \quad \mathbf{0}]\mathbf{R}$ is row full rank, the data points $\mathbf{Q}$ can be interpolated by the surface $S(u, v)$.

□

References

[1] Atallah, M.J., 1983. A linear time algorithm for the hausdorff distance between convex polygons. Inf. Process. Lett. 17, 207–209.

[2] Choi, G.P.T., Ho, K.T., Lui, L.M., 2016. Spherical conformal parameterization of genus-0 point clouds for meshing. SIAM Journal on Imaging Sciences 9, 1582–1618.

[3] De Boor, C., 1976. Total positivity of the spline collocation matrix. Indiana University Mathematics Journal 25, 541–551.

[4] De Boor, C., 1978. A practical guide to splines. volume 27. Springer New York.

[5] Deng, C., Lin, H., 2014. Progressive and iterative approximation for least squares b-spline curve and surface fitting. Computer-Aided Design 47, 32–44.

[6] 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, Association for Computing Machinery, New York, NY, USA. p. 173–182.

[7] Greiner, G., 1994. Variational design and fairing of spline surfaces, in: Computer Graphics Forum, Wiley Online Library. pp. 143–154.

[8] Guennebaud, G., Jacob, B., et al., 2010. Eigen v3. http://eigen.tuxfamily.org.

[9] Hadenfeld, J., 1995. Local energy fairing of b-spline surfaces. Mathematical Methods for Curves and Surfaces , 203–212.

[10] Hahmann, S., Konz, S., 1997. Fairing bi-cubic b-spline surfaces using simulated annealing. Curves and Surfaces with Applications in CAGD , 159–168.

[11] Hoschek, J., Lasser, D., 1993. Fundamentals of computer aided geometric design. AK Peters, Ltd.

[12] Jacobson, A., Panozzo, D., 2017. Libigl: Prototyping geometry processing research in c++, in: SIGGRAPH Asia 2017 Courses, Association for Computing Machinery, New York, NY, USA.

[13] Kawasaki, T., Jayaraman, P.K., Shida, K., Zheng, J., Maekawa, T., 2018. An image processing approach to feature-preserving b-spline surface fairing. Computer-Aided Design 99, 1–10.

[14] Kineri, Y., Wang, M., Lin, H., Maekawa, T., 2012. B-spline surface fitting by iterative geometric interpolation/approximation algorithms. Computer-Aided Design 44, 697–708.

[15] Lee, S., Wolberg, G., Shin, S.Y., 1997. Scattered data interpolation with multilevel b-splines. IEEE Transactions on Visualization and Computer Graphics 3, 228–244.

[16] Lévy, B., Petitjean, S., Ray, N., Maillot, J., 2002. Least squares conformal maps for automatic texture atlas generation. ACM Trans. Graph. 21, 362–371.

[17] Lin, H., Maekawa, T., Deng, C., 2018. Survey on geometric iterative methods and their applications. Computer-Aided Design 95, 40–51.

[18] Lin, H., Zhang, Z., 2011. An extended iterative format for the progressive-iteration approximation. Computers & Graphics 35, 967–975.

[19] Liu, L., Zhang, L., Xu, Y., Gotsman, C., Gortler, S.J., 2008. A local/global approach to mesh parameterization, in: Proceedings of the Symposium on Geometry Processing, Eurographics Association, Goslar, DEU. p. 1495–1504.

[20] Ma, W., Kruth, J., 1995. Parameterization of randomly measured points for least squares fitting of b-spline curves and surfaces. Computer-Aided Design 27, 663–675.

[21] Meng, Q., Li, B., Holstein, H., Liu, Y., 2013. Parameterization of point-cloud freeform surfaces using adaptive sequential learning rbfnetworks. Pattern Recognition 46, 2361–2375.

[22] Nishiyama, Y., Nishimura, Y., Sasaki, T., Maekawa, T., 2007. Surface faring using circular highlight lines. Computer-Aided Design and Applications 4, 405–414.

[23] Park, H., Kim, K., 1996. Smooth surface approximation to serial cross-sections. Computer-Aided Design 28, 995–1005.

[24] Park, H., Kim, K., Lee, S.C., 2000. A method for approximate nurbs curve compatibility based on multiple curve refitting. Computer-Aided Design 32, 237–252.

[25] Piegl, L., Tiller, W., 1996. The NURBS book. Springer Science & Business Media.

[26] Piegl, L.A., Tiller, W., 2000a. Reducing control points in surface interpolation. IEEE Computer Graphics and Applications 20, 70–75.

[27] Piegl, L.A., Tiller, W., 2000b. Surface approximation to scanned data. The Visual Computer 16, 386–395.

[28] Piegl, L.A., Tiller, W., 2002. Surface skinning revisited. The Visual Computer 18, 273–283.

[29] Varady, T., Martin, R.R., Cox, J., 1997. Reverse engineering of geometric models—an introduction. Computer-Aided Design 29, 255–268.

[30] Wang, W., Tian, T., Li, S., 2011. B-spline surface skinning to non-parallel sectional curves, in: Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry, pp. 569–572.

[31] Wang, W., Zhang, Y., 2010. Wavelets-based nurbs simplification and fairing. Computer Methods in Applied Mechanics and Engineering 199, 290–300.

[32] Wang, W.K., Zhang, H., Park, H., Yong, J.H., Paul, J.C., Sun, J.G., 2008. Reducing control points in lofted b-spline surface interpolation using common knot vector determination. Computer-Aided Design 40, 999–1008.

[33] Weiss, V., Andor, L., Renner, G., Várady, T., 2002. Advanced surface fitting techniques. Computer Aided Geometric Design 19, 19–42.

[34] Woodward, C.D., 1988. Skinning techniques for interactive b-spline surface interpolation. Computer-Aided Design 20, 441–451.

[35] Yamaura, Y., Nanya, T., Imoto, H., Maekawa, T., 2015. Shape reconstruction from a normal map in terms of uniform bi-quadratic b-spline surfaces. Computer-Aided Design 63, 129–140.

[36] Ye, X., Jackson, T.R., Patrikalakis, N.M., 1996. Geometric design of functional surfaces. Computer-Aided Design 28, 741–752.

[37] Zhang, C., Zhang, P., Cheng, F.F., 2001. Fairing spline curves and surfaces by minimizing energy. Computer-Aided Design 33, 913–923.

[38] Zhang, L., Liu, L., Gotsman, C., Huang, H., 2010. Mesh reconstruction by meshless denoising and parameterization. Computers & Graphics 34, 198–208.

[39] Zhang, Y., Cao, J., Chen, Z., Li, X., Zeng, X.M., 2016. B-spline surface fitting with knot position optimization. Computers & Graphics 58, 73–83.