Spline Smoothing of 3D Geometric Data

Xinyi Li\textsuperscript{a}, Shan Yu\textsuperscript{b}, Yueying Wang\textsuperscript{c}, Guannan Wang\textsuperscript{d} and Li Wang\textsuperscript{c}

\textsuperscript{a}Clemson University, \textsuperscript{b}University of Virginia, \textsuperscript{c}Iowa State University and \textsuperscript{d}College of William \& Mary

Abstract: Over the past two decades, we have seen an increased demand for 3D visualization and simulation software in medicine, architectural design, engineering, and many other areas, which have boosted the investigation of geometric data analysis and raised the demand for further advancement in statistical analytic approaches. In this paper, we propose a class of spline smoothers appropriate for approximating geometric data over 3D complex domains, which can be represented in terms of a linear combination of spline basis functions with some smoothness constraints. We start with introducing the tetrahedral partitions, Barycentric coordinates, Bernstein basis polynomials, and trivariate spline on tetrahedra. Then, we propose a penalized spline smoothing method for identifying the underlying signal in a complex 3D domain from potential noisy observations. Simulation studies are conducted to compare the proposed method with traditional smoothing methods on 3D complex domains.

Key words and phrases: Complex domain; Nonparametric smoothing; penalized splines; Trivariate splines; Tetrahedra partitions.

1. Introduction

Recent advances in computer and information technology have dramatically boosted the availability of three-dimensional (3D) geometric data in many fields, as illustrated in Figure 1.1, including manufacturing ((a)–(c)), medical science ((d)–(e)), arts ((f)–(g)) and animation ((h)–(i)). For example, by utilizing 3D technology within medical imaging, we are able to take and combine those two-dimensional (2D) cross-section slices into a concise 3D visual of the area being scanned. With 3D medical imaging, healthcare professionals can now access new angles, resolutions, and details that offer an all-around better understanding of the body part in question. As a result, statistical learning for 3D geometry is emerging as a critical research area in many fields.

In reality, 3D geometric data usually contains unwanted noise that obscures the features of the volumes of interest. Consequently, denoising is critical in the 3D geometric analysis; for example, in many imaging studies, smoothing methods are widely adopted in image processing tasks \cite{Goldsmith2014,Kang2020}. As shown in Figure 1.1, the objects (or regions) of...
interest are usually irregularly shaped. Many conventional denoising methods, such as Kernel smoothing (Zhu et al., 2014), tensor product smoothing (Reiss and Ogden, 2010), thin plate spline smoothing (Duchon, 1976; Yue and Speckman, 2010), and wavelet smoothing (Morris and Carroll, 2006), suffer from the problem of “leakage” across the complex domains, which refers to the poor estimation over difficult regions caused by smoothing inappropriately across boundary features (Ramsay, 2002; Wang and Ranalli, 2007; Kim and Wang, 2021). Several techniques have been developed in the last decades to tackle the “leakage” problem when recovering the true signals. Many of the work focus on 2D manifold domains, such as the spline smoothing (Wang and Ranalli, 2007; Wilhelm et al., 2016), finite element analysis (Lindgren et al., 2011), and Kernel smoothing (Guo et al., 2010). For 3D cases, Chung et al. (2018) applied the discrete heat kernel smoothing to irregular image domains, and Huo et al. (2020) considered the finite element method for irregular 3D subjects.

To deal with this prevalent challenge in 3D geometric data analysis, we propose to use trivariate penalized spline smoothing methods based on tetrahedral partitions. To handle the complex 3D domains, it is crucial to construct an ideal representation of the 3D geometry. In computer graphics and geometric-aided design, tetrahedral partitions are the standard representation of 3D objects. For example, many 3D scanners use the laser to capture an object’s exact size and shape into the computer as a digital 3D triangle mesh representation. Although traditional medical planning systems often use either voxel-based or surface-based models, tetrahedral meshes are more flexible than both data structures because complex volu-

Figure 1.1: Examples of 3D geometric data and corresponding tetrahedral meshes generated with iso2mesh toolbox and distmeshnd function using MATLAB.
metric data sets can be represented with few primitives. After an appropriate partition of the domain, we can construct the trivariate spline basis function on the tetrahedral partition to approximate data distributed over 3D complex domains.

Moreover, the characteristics of 3D geometry data create substantial challenges in analysis (Wong et al., 2016). First, we need to take into account the large scale and high resolution of the collected data. For example, a standard PET scan from one subject contains \((79 \times 95 \times 68)\) 510,340 voxels. Therefore, even for a single image, the number of observations is very high. One of the significant advantages of the spline smoothing method lies in its computational simplicity. As a global estimation with an explicit model expression, the fitting of splines techniques is achieved by solving a single linear system.

Besides the computation issues raised by the large volume of data, the challenges also arise from the complicated structure of the signal. On one hand, the underlying data may vary mildly or sharply according to the location. We propose the Adaptive Trivariate Penalized Spline on Tetrahedral Partitions (ATPST) by adopting adaptive weights in smoothing spline to overcome this challenge. On the other hand, missing data is common and inevitable in 3D geometry data acquisition. For example, in 3D seismology, the incomplete and sparse data in the collection is often caused by physical and economic constraints (Chai et al., 2020). In our simulation study, we mimic the cases in real life and conduct experiments on missing data with multiple missing schemes. The simulation results reveal the superiority of the proposed estimators, regardless of the setting.

Other than denoising or deblurring, the reconstruction of 3D geometry data in a different (usually higher) resolution is another important task in analysis. In terms of statistics, this is equivalent to providing an accurate prediction based on estimation. The proposed spline smoothing methods illustrate supreme prediction power in simulation studies.

The rest of the paper is structured as follows. In Section 2, we give an overview of the tetrahedral partitions of a 3D domain. Once a tetrahedral partition has been constructed, we can construct the spline space over the tetrahedral partition. In Section 3, we introduce the penalized spline estimators. We then demonstrate the smoothers’ performance on simulated problems in Section 4. Finally, this paper is concluded, and future work is outlined in Section 5.

2. Tetrahedral partitions and trivariate splines on tetrahedr

In this section, we provide a basic framework for tetrahedral partitions and trivariate splines on those partitions. More detailed introductions can be referred to online supplementary materials (Appendices A and B).

Let \( Y_i \) be the response variable observed on the \( i \)th location or design point \( \mathbf{v}_i = (x_i, y_i, z_i) \),
Suppose all \( v_i \)'s range over a bounded domain \( \Omega \subseteq \mathbb{R}^3 \) of arbitrary shape, for example, the 3D horseshoe domain shown in Figure 1.1(a). For any given dataset with design points located on \( \Omega \), one can find a polygonal domain (a domain with a piecewise linear boundary) to include all the design points. With this principle, in the following, we assume that \( \Omega \) is a polygonal domain itself, which may have one or multiple holes not containing any design points.

2.1. Tetrahedral partitions

In the following, we use \( T \) to denote a tetrahedron, which is a convex hull of four non-coplanar points in \( \mathbb{R}^3 \). A collection \( \Delta = \{T_1, \ldots, T_N\} \) of \( N \) tetrahedra is called a \textit{tetrahedral partition} of \( \Omega = \bigcup_{h=1}^{N} T_h \) provided that any pair of tetrahedra in \( \Delta \) intersect at most at a common vertex, along a common edge, or along a common triangular face. In this paper, we restrict our attention to tetrahedralizations of \( \Omega \) because any polygonal domain of arbitrary shape can be partitioned into finitely many tetrahedra. Without loss of generality, we assume that all the location points \( v_i \)'s are inside tetrahedra of \( \Delta \), that is, they are not on edges, vertices, or faces of tetrahedra in \( \Delta \). Otherwise, we can simply count them twice or multiple times if any observation is located on a face, an edge, or at a vertex of \( \Delta \) (Lai and Schumaker, 2007).

2.2. Tetrahedral partitions in practice

We use Figure 2.1 to illustrate the flowchart of obtaining a tetrahedral partition in practice, given a 3D object (taking the cow model in Figure 1.1(h) as an example) with arbitrary complex shape. This mesh generation procedure also works for other typical 3D data objects, like those in Figure 1.1.

For practical purposes, there are plenty of tetrahedral partitions that satisfy our definition in Section 2.1, wherein the Delaunay tetrahedral partition is most preferred. The Delaunay tetrahedral partition has been well-studied with superior properties for 2D objects (Chew, 1989; Lee and Schachter, 1980), for example, a 2D Delaunay triangulation is “angle optimal”, i.e., it maximizes the minimum angle in a triangulation. Extending the 2D Delaunay triangulation to higher dimensionality, a 3D Delaunay tetrahedral partition possesses similar property that it maximizes the minimal solid angle in the tetrahedra.

In the past few decades, Delaunay-based mesh generation algorithms for 3D domains have also been implemented in various software packages and toolboxes, such as the functions delaunayTriangulation, delaunay and distmesh/distmeshnd (Persson and Strang, 2004) in MATLAB, QualMesh (Cheng et al., 2005) and the CGAL library (Jamin et al., 2015) The CGAL Project, 2020) for C++, TetGen (Si, 2015) and iso2mesh (Fang and Boas, 2009) built on CGAL as well.
Figure 2.1: A flowchart demonstrating how a tetrahedral partition can be obtained from a 3D object with arbitrary complex shape.

as TetGen for MATLAB/Octave. Note that even though all the algorithms can be used to generate triangular meshes, the implementation details and usages differ. For example, the functions distmesh and distmeshnd take distance functions as representations of boundary geometry, while delaunayTriangulation generates Delaunay 2D/3D triangulation based on given point sets. In practice, when the mathematical representations of boundary geometry or the vertex sets for Delaunay triangulations are not immediately available, one can employ TetGen instead, whose inputs are triangular surface mesh on domain boundaries, as shown in Figure 2.1.

2.3. Trivariate splines on a tetrahedra partition

Given a tetrahedron $T = \langle v_1, v_2, v_3, v_4 \rangle$, any fixed point $v := (x, y, z) \in \mathbb{R}^3$ has a unique representation in terms of $\langle v_1, v_2, v_3, v_4 \rangle$,

$$v = b_1 v_1 + b_2 v_2 + b_3 v_3 + b_4 v_4, \quad \text{with } b_1 + b_2 + b_3 + b_4 = 1,$$

where $(b_1, b_2, b_3, b_4)$ are called the barycentric coordinates of $v$ relative to the tetrahedron $T$, and they are nonnegative if $v$ is inside or on the faces of $T$. Accordingly, for some nonnegative integers $i, j, k, l$ with $i + j + k + l = d$, define trivariate Bernstein basis polynomial of degree $d$ relative to $T$ as

$$B_{ijkl}^d(v) := \frac{d!}{i!j!k!l!} b_1^i b_2^j b_3^k b_4^l, \quad \text{with } i + j + k + l = d.$$

For any positive integer $d$ and tetrahedron $T$, let $P_d(T)$ be the space of all trivariate polynomials defined on $T$ with degrees less than or equal to $d$. Note that the dimension of $P_d(T)$ is $\binom{d+3}{3}$. 

5
According to Theorem 15.8 in Lai and Schumaker (2007) and Lemma D.1 in Section D.1 in the supplement, the set of Bernstein basis polynomials

\[ B^d_T(v) := \{ B^d_{ijkl}(v) : i,j,k,l \geq 0, i+j+k+l = d \} \]

forms a basis for the space of polynomials \( P_d(T) \). Thus, given Bernstein basis functions \( \{ B^d_{ijkl}(v) \}_{i+j+k+l=d} \), any polynomial \( p(v) \in P_d(T) \) can be written uniquely as \( B \)-form,

\[ p(v)|_T = \sum_{i+j+k+l=d} \gamma_{T;ijkl} B^d_{ijkl}(v) = B^d_T(v)^\top \gamma_T, \]

where the coefficients \( \gamma_T = \{ \gamma_{T;ijkl} \}_{i+j+k+l=d} \) are called \( B \)-coefficients of \( p \).

Consider the whole domain \( \Omega \) and let \( C^r(\Omega) \) be the collection of all \( r \)-th continuously differentiable functions over \( \Omega \) for some nonnegative integer \( r \). For tetrahedral partition \( \Delta = \{ T_1, \ldots, T_N \} \), let \( S^r_d(\Delta) = \{ s \in C^r(\Omega) : s|_T \in P_d(T), T \in \Delta \} \) be a spline space of degree \( d \) and smoothness \( r \) over \( \Delta \), where \( s|_T \) is the polynomial piece of spline \( s \) restricted on tetrahedron \( T \).

**Remark 1.** There are a variety of choices for \( d \) and \( r \). According to Lai (1989), space \( S^r_d(\Delta) \) with tetrahedral partition \( \Delta \) and \( d \geq 6r+3 \), reach the full approximation power.

According to (B.1) in Section B.1 in the supplement, for any \( s \in S^r_d(\Delta) \), there exists \( \gamma_T = \{ \gamma_{T;ijkl} \}_{i+j+k+l=d} \) such that

\[ s|_{T_J}(v) = \sum \gamma_{T;ijkl} B^d_{ijkl}(v) = B^d_{T_J}(v)^\top \gamma_T, \quad J = 1, \ldots, N. \]

Consequently, let \( B_d = ((B^d_{T_1})^\top, \ldots, (B^d_{T_N})^\top)^\top \) and \( \gamma = (\gamma_{T_1}^\top, \ldots, \gamma_{T_N}^\top)^\top \), one has \( s(v) = B_d(v)^\top \gamma \). To satisfy the smoothness conditions in \( s \), certain smoothness constraints are imposed on the \( B \)-coefficients \( \gamma \). One can obtain a smoothness constraint matrix \( H \) such that \( H\gamma = 0 \) by repeatedly applying the equation (B.8) over all shared triangular faces, and more details are available in Section B.2 in the supplement.

### 3. Penalized spline estimators

With all the preparations in the previous introduction, we apply the trivariate spline over tetrahedral partition to estimate functions on a topologically and/or geometrically complex 3D domain \( \Omega \) in this section. For observations \( \{(v_i, Y_i)\}_{i=1}^n \), we consider the following nonparametric regression model

\[ Y_i = m(v_i) + \epsilon_i, \]

where \( m \) is some smooth but unknown function, and \( \epsilon_i \) is random error with mean zero.
In nonparametric smoothing, the roughness penalty approach is widely used when smoothing noisy data (Green and Silverman, 1994; Wood, 2003; Lai and Wang, 2013). Including a roughness penalty and choosing a proper tuning parameter can avoid overfitting problems and balances the bias and variance of the estimator of \( m \). We formulate the roughness penalty approach as the following penalized least squares problem:

\[
\min_m \sum_{i=1}^{n} \{Y_i - m(v_i)\}^2 + \lambda \mathcal{E}(m),
\]

where \( \mathcal{E}(m) \) is a penalty function with details given in Section 3.1, and \( \lambda \) is the roughness penalty parameter. A larger \( \lambda \) leads to a less fluctuating \( m \). If \( \lambda \) goes to infinity, our estimator shrinks to linear functions where the roughness penalty \( \mathcal{E}(m) = 0 \). A proper penalty parameter \( \lambda \) balances the goodness of fit for the data and the volatility of estimated functions. We propose to seek for the minimizer of (3.1) in \( S^d_{\Delta} \) and denote it as \( \hat{m}_\lambda \), which is the Trivariate Penalized Spline over Tetrahedral partition (TPST) estimator of \( m \).

### 3.1. Tetrahedral partitions

In this section, we describe the details of implementing the TPST method. For the penalty function in (3.1), we consider

\[
\mathcal{E}(m) = \sum_{i+j+k=2} \left(2_i \right) \left(2 - i \right) \int_{\Omega} \{D^i_x D^j_y D^k_z m(v)\}^2 dv
\]

as the roughness penalty. The penalty \( \mathcal{E}(m) \) is a commonly used penalty, the second-order derivative to zero for surface smoothing, see Green and Silverman (1994) for the two-dimensional case.

For a spline function \( s \in S^d_{\Delta} \), the roughness penalty can be written as the following:

\[
\mathcal{E}(s) = \sum_{T \in \Delta} \mathcal{E}(s_T) = \sum_{T \in \Delta} \sum_{i+j+k=2} \left(2_i \right) \left(2 - i \right) \int_T \{D^i_x D^j_y D^k_z s_T(v)\}^2 dv,
\]

where \( s_T(v) = \sum_{i+j+k+l=d} \gamma_{T;ijkl} B_{ijkl}^T(v) \) is a function restricted on \( T \). Specifically, we have

\[
\mathcal{E}(s_T) = \sum_{g,g' \in \{x,y,z\}} \int_T \left\{ \sum_{i+j+k+l=d} \gamma_{T;ijkl} D_g D_{g'} B_{ijkl}^T(v) \right\}^2 dv
\]

\[
= \sum_{g,g' \in \{x,y,z\}} \gamma_{g,g'}^T \mathbf{P}_{g,g'}^T \gamma_{g,g'} = \gamma_T^T \mathbf{P}_T \gamma_T,
\]

where \( \gamma_T = \{\gamma_{T;ijkl}\}_{i+j+k+l=d} \) and \( \mathbf{P}_{g,g'}^T \), for \( g, g' = x, y, z \) are the \( \binom{d+3}{3} \times \binom{d+3}{3} \) matrices with entries being \( \int_T \left\{ D_{g,g'}^2 B_{ijkl}^T(v) \right\} \left\{ D_{g,g'}^2 P_{ijkl}^T d\nu \right\} dv \), respectively. See Appendix C for more details in calculations of penalty matrices \( \mathbf{P}_{g,g'}^T \), for \( g, g' \in \{x, y, z\} \) and \( T \in \Delta \).
Remark 2. For a spline function \( s \in \mathcal{S}_d^r(\triangle) \) with B-coefficients \( \gamma \), its roughness penalty \( \mathcal{E}(s) \) can be written as some quadratic function of \( \gamma \), that is, \( \mathcal{E}(s) = \gamma^\top P \gamma \), where \( P = \text{diag}(P_T, T \in \triangle) \) is a block diagonal matrix.

Accordingly, we have \( \mathcal{E}(B_d^\top \gamma) = \gamma^\top P \gamma \), where \( P \) is the block diagonal penalty matrix. Thus, the objective function can be written as

\[
\min_\gamma \sum_{i=1}^n \left\{ Y_i - B_d(v_i)^\top \gamma \right\}^2 + \lambda \gamma^\top P \gamma, \quad \text{subject to } H \gamma = 0. \tag{3.3}
\]

One can use QR decomposition to get rid of the constraint. Specifically, \( H^\top = QR = (Q_1 \ Q_2)(R_1 \ R_2) \), where \( Q \) is an orthogonal matrix, \( R \) is an upper triangular matrix, \( R_1 \) is a full rank matrix with the same rank as \( H \), and \( R_2 \) is a matrix of zeros. Note that for any vector \( \gamma \) satisfying \( H \gamma = 0 \), there exists some \( \theta \) such that \( \gamma = Q_2 \theta \). Also, for any \( \theta \), \( H(Q_2 \theta) = 0 \) holds. Thus, (3.3) is equivalent to a penalized regression problem without constraint:

\[
\sum_{i=1}^n \left\{ Y_i - B_d(v_i)^\top Q_2 \theta \right\}^2 + \lambda \theta^\top Q_2^\top P Q_2 \theta. \tag{3.4}
\]

Let \( Y = (Y_1, \ldots, Y_n)^\top, \ B_d = (B_d(v_1), \ldots, B_d(v_n))^\top \). Then, the solution of (3.4) is given by \( \hat{\theta}_\lambda = (Q_2^\top B_d^\top B_d Q_2 + n^{-1} \lambda Q_2^\top P Q_2)^{-1} Q_2^\top B_d^\top Y \) and \( \hat{\gamma}_\lambda = Q_2^\top \hat{\theta}_\lambda \), which yields the TPST estimator \( \hat{m}_\lambda(v) = B_d(v)^\top \hat{\gamma}_\lambda \). Several methods can be used for choosing the penalty parameter, such as the block cross-validation and generalized cross-validation. Detailed discussion about these methods is given in Section C.4 in the supplement.

3.2. The ATPST estimator

To deal with the complicated variation on the complex 3D domain, we propose a total variation based estimator, ATPST, by adjusting the smoothness penalty part in the TPST estimator.

In (3.2), the roughness penalty function imposes unique penalty weights across \( \Omega \), which is not ideal for functions with various degrees of volatility among different regions. To provide a precise estimation for functions with various volatility, we consider an adaptive penalty function where the penalty weights are assigned according to the local volatility. To be specific, we update the adaptive energy function \( \mathcal{E} \) as

\[
\mathcal{E}^\omega(s) = \sum_{T \in \triangle} \omega_T \sum_{i+j+k=2} \binom{2}{i} \binom{2-i}{j} \int_T \{D_x^i D_y^j D_z^k s_T(v)\}^2 dv,
\]

where \( \omega_T \) are the weights measuring local volatility. Similar to Remark 2 in Section 3.1, for \( s \in \mathcal{S}_d^r(\triangle) \) with B-coefficients \( \gamma \), \( \mathcal{E}^\omega(s) = \sum_{T \in \triangle} \omega_T \gamma_T^\top P_T \gamma_T \). Then, one can obtain the ATPST
by minimizing
\[
\sum_{i=1}^{n} \left\{ Y_i - \mathbf{B}_d(v)^\top \mathbf{\gamma} \right\}^2 + \lambda \sum_{T \in \Delta} \omega_T \mathbf{\gamma}_T^\top \mathbf{P}_T \mathbf{\gamma}_T, \text{ subject to } \mathbf{H} \mathbf{\gamma} = \mathbf{0}.
\]

We define \( \omega_T = (C - C_m^{-1} V_T^{-1} \|m\|_{T;TV})^\tau \), where \( C_m = \max \{ V_T^{-1} \|m\|_{T;TV}, T \in \Delta \} \), \( C \) and \( \tau \) are some positive constants, which can be chosen through cross validation, and \( \|m\|_{T;TV} \) is the total variation of the function \( m \) within tetrahedron \( T \). Note that \( \omega_T \) is a monotonically decreasing function of \( \|m\|_{T;TV} \), which generates high gradient for regions with large variation. Specifically,
\[
\|m\|_{T;TV} = \sup \left\{ \int_T m(v) \text{div} f(v) dv : f \in C^1_c(\Omega, \mathbb{R}^3), \|f\|_{\infty, \Omega} \leq 1 \right\},
\]
where \( C^1_c(\Omega, \mathbb{R}^3) \) is the set of continuously differentiable vector functions of compact support contained in \( \Omega \), \( \text{div} f(v) \) is the divergence of function \( f \), and \( \| \cdot \|_{\infty, \Omega} \) is the essential supremum norm. If \( m \) is a differentiable function, one has \( \|m\|_{T;TV} = \int_T \{D_x m(v)\}^2 + \{D_y m(v)\}^2 + \{D_z m(v)\}^2 \}^{1/2} dv \).

In practice, we suggest using the TPST estimator as an initial estimator for the total variation at tetrahedron \( T \). Figure 3.1 shows the plot of the estimated \( \omega_T \) based on a typical data sample in our simulation study, from which one sees that larger weights are assigned to the less fluctuant region. By the property of the first-order partial derivative of spline functions, \( \|m\|_{T;TV} \) can be approximated by \( \int_T \{\hat{\gamma}_T^\top \mathbf{C}_d^{(1)}(x) \mathbf{B}_{d-1}(v)\}^2 + \{\hat{\gamma}_T^\top \mathbf{C}_d^{(1)}(y) \mathbf{B}_{d-1}(v)\}^2 + \{\hat{\gamma}_T^\top \mathbf{C}_d^{(1)}(z) \mathbf{B}_{d-1}(v)\}^2 \}^{1/2} dv \), where \( \hat{\gamma}_T \) is the spline coefficient of the TPST estimator at tetrahedron \( T \).

### 4. Simulation studies
In this section, we conduct various simulation studies to assess the performance of the proposed methods, TPST and ATPST. For the ATPST, we consider the weight functions defined in Section 3.2 with \( \tau = 2 \) and \( C \) being selected from a set of grid points ranging from 1 to 3 based on the GCV criteria. We compare the proposed methods with the traditional tensor product spline method (Stone, 1994) and thin plate spline smoothing (Wood, 2003). We then implement the tensor product spline and the thin plate spline smoothing using the function \texttt{gam} in R package \texttt{mgcv}.

In all simulation examples in this section, the data are generated from the following model:

\[
Y_i = m(v_i) + \epsilon_i, \; v_i \in \Omega, \; i = 1, \ldots, n.
\]

The random noises, \( \epsilon_i \)'s, are independent and identically distributed and generated from \( N(0, \sigma^2) \), where \( \sigma \) is chosen according to the peak signal-to-noise ratios (PSNR) defined as

\[
\text{PSNR} = 10 \log_{10} \left[ \frac{\max_i m(v_i)^2}{\sigma^2} \right].
\]

In the simulation studies, we set PSNR = 5 and 10, representing scenarios of high and moderate noise levels, respectively. Depending on the nature of the design points, we conduct numerical experiments with both random and fixed designs. In a fixed design, the locations \( v_i \)'s are deterministic, and this is commonly used in imaging analysis; on the other hand, the case of random design is more towards the statistical learning setup.

**4.1. Random design with complete data**

In this example, we consider a random design in which the observations are randomly generated over the entire domain. To be more specific, we set the sample size \( n = 20,000 \) and \( 50,000 \). To mimic some complicated scenarios in practice, we consider two different domains: (i) a cube with a hole inside (\( \Omega_1 \)); (ii) a 3D horseshoe (\( \Omega_2 \)). These domains are illustrated in Figure 4.1. To compare the TPST and the ATPST, we consider two types of functions with different degrees of variation for each domain. The true functions shown in the first column of Figure 4.1 are smooth functions with less volatility, while the functions shown in the second column are more waggling. Furthermore, in this simulation study, we investigate the effect of sample size, degree of spline polynomial, and tetrahedral partition.

To evaluate the estimation and prediction performance of each method, we calculate the out-of-sample mean integrated squared error (MISE). Figure 4.2 presents the average of the MISEs over 200 replications for all different scenarios. Based on Figure 4.2, one can observe that as the sample size or the PSNR increases, the estimation and prediction accuracy improves for all the methods. Regardless of the simulation scenarios, the proposed TPST and ATPST methods outperform the other two traditional methods since the traditional methods cannot handle the “leakage” problem over the complex domain.
Figure 4.1: True functions and tetrahedral partitions over domains $\Omega_1$ and $\Omega_2$.

For the ATPST, as discussed in Section 3.2, we adaptively assign weights for each penalty term based on the total variation within each tetrahedron $T$; see Figure 3.1. Comparing the TPST and the ATPST, one can see that when the actual underlying function is relatively smooth, the performance of the TPST and the ATPST are similar to each other. However, when the actual underlying function is fluctuant, the ATPST improves the estimation and prediction accuracy of the TPST for most of the simulation scenarios.

Figure 4.2: Plots of average MISEs based on different methods under random design.

To evaluate the effect of tetrahedral partitions on the TPST and ATPST, we consider a relatively coarse mesh $\triangle_1$ and a fine mesh $\triangle_2$ for each domain. An illustration of these partitions is given in the third and the fourth columns of Figure 4.1 and the number of vertices and the number of tetrahedrons are summarized in Table 4.1. From Figure 4.2, we can see that for each domain, both TPST and ATPST estimators have similar performance.
based on the two partitions. Moreover, for the proposed penalized splines, when the number of the tetrahedron is sufficiently large to capture the pattern and features of the data, finer tetrahedral partitions will not further benefit the estimation.

Table 4.1: Number of tetrahedrons (vertices) in each partition.

|               | Ω₁ − △₁ | Ω₁ − △₂ | Ω₂ − △₁ | Ω₂ − △₂ |
|---------------|----------|----------|----------|----------|
|               | 240 (120) | 456 (180) | 504 (207) | 1236 (426) |

To study the effect of the degree of spline on the proposed TPST and ATPST. Based on Figure 4.2, one can see that the difference between estimates with \( d = 3 \) and \( d = 4 \) are relatively small. When we use a larger \( d \), the estimators are usually less biased but with larger variance and more computationally intensive. In this example, \( d = 3 \) is preferred as they are more computationally efficient. In general, the choice of \( d \) depends on the smoothness of the underlying function, the strength of signals, and computing resources.

As discussed previously, the spline methods are computationally efficient since they provide a global estimator. In this simulation example, the TPST method takes less than twenty seconds to fit the model for most of the simulation samples. The speed is comparable to tensor product and thin plate spline estimators. The ATPST needs a little more time to select the best weight functions, and it usually takes less than five minutes to obtain the estimates under all scenarios.

4.2. Fixed design with missing data

This example further investigates the proposed method on a fixed design at two different scales/resolutions. Noticing that missing areas or voxels may occur within the domain, we mimic three types of missing schemes in this example, including (i) complete data with no missing, (ii) missing at random, and (iii) missing in a contiguous block as well as at random. Figures 4.3–4.5 illustrate different types of missing data we deal with in this example. Furthermore, we explore various missing rates under different missing mechanisms. For missing at random, we consider the missing rates ranging uniformly from 0 to 0.5, where 0 represents no missing voxels, and 0.5 means half of the voxels are missing. The contiguous block shown in Figures 4.4 (b) and (j) contains 12% of the data, and thus the lowest missing rate in the second missing type is 12%.

Based on simulation example 1, we can see that the performance for various methods is relatively consistent. Thus, in this example, we only consider the first domain, \( Ω₁ \), and try the same tetrahedral partitions as in example 1. We consider two fixed resolutions/scales here: the lower resolution/scale is \( 60 \times 20 \times 20 \) with 22,160 voxels falling within the domain, and the
Similar to example 1, we calculate the average MISEs over 200 replications with different missing types and missing rates and summarize them in Tables 4.2–4.3, while demonstrating them in Figure 4.6. In addition, Figures 4.3 and 4.4 show a typical simulation example for \( n = 50,000 \) and PSNR = 5 based on different methods under different missing schemes. Based on these tables and figures, one can clearly see that the prediction accuracy improves for all of the methods as the missing rate decreases. The proposed methods outperform the two traditional methods regardless of the type of missing scheme and the missing rate.

Furthermore, the type of missing does not affect the two proposed methods very much and the MISEs are very comparable for similar missing rates for the two types of missing. However, the thin plate spline smoothing usually performs better for missing at random, while the tensor product spline is better when missing a contiguous block and at random.

5. Conclusions and Discussion

Challenges in 3D visualization and modern computer-aided design motivated the advanced statistical analysis of 3D geometric data. Conventional smoothing methods sometimes perform poorly over complex domains. The proposed approaches are able to effectively denoise or deblur the collected data while preserving inherent geometric features or spatial structures and provide multi-resolution reconstruction. The experimental results demonstrate the effectiveness of the proposed approaches in comparison to existing smoothing techniques.

Advancements in information technology have dramatically enhanced the resolution of images. Even for a single high-quality image, the number of observations is very high, and thus, the computational burden could also be high. To address the “big” data issue, the divide and conquer algorithm is an appealing method that recursively breaks down a problem into two or more sub-problems of the same or related type until these become simple enough to be solved directly. It then aggregates solutions to the sub-problems to produce a global solution.

Unlike tensor product spline or thin-plate spline, one unique feature of the proposed TPST is its great scalability in computing, because the spline basis function is generated restricted to each triangle without any overlap, while the constraints on the spline coefficients achieve smoothness. Based on this feature, a divide and conquer algorithm could be considered in future works.

Acknowledgements

Li Wang’s research was partially supported by National Science Foundation award DMS-1916204.
Figure 4.3: True and estimated functions when missing at random based on different methods for a typical replication with $n = 50,000$, missing rate = 30%, and PSNR = 5.
Figure 4.4: True and estimated functions when missing a contiguous block based on different methods for a typical replication with $n = 50,000$, and PSNR = 5.
Figure 4.5: True and estimated functions when missing a contiguous block & at random based on different methods for a typical replication with $n = 50,000$, missing rate = 30%, and PSNR = 5.
Figure 4.6: Plots of average MISEs based on different methods under a fixed design.
Table 4.2: MISEs of different methods under fixed design with missing at random.

| \(m\) | PSNR | Grid | Missing | Plate | Product |
|------|------|------|---------|-------|---------|
|      |      |      | Rate    | \(\Delta_1\) | \(\Delta_2\) | \(\Delta_1\) | \(\Delta_2\) |
| 1    | 5    | 0.1370 | 0.1341 | 0.1.20 | 0.1220 | 0.8397 | 0.9077 |
|      | 0.1  | 0.1453 | 0.1424 | 0.1.31 | 0.1300 | 0.8465 | 0.9186 |
| 2    | 0.2  | 0.1555 | 0.1528 | 0.1.41 | 0.1405 | 0.8515 | 0.9556 |
|      | 0.3  | 0.1688 | 0.1662 | 0.1.51 | 0.1537 | 0.8672 | 0.9533 |
|      | 0.4  | 0.1864 | 0.1839 | 0.1.61 | 0.1713 | 0.8831 | 0.9807 |
|      | 0.5  | 0.2048 | 0.2027 | 0.1.71 | 0.1907 | 0.9027 | 1.0141 |
| 5    | 0    | 0.0944 | 0.0911 | 0.0828 | 0.0816 | 0.8091 | 0.8525 |
|      | 0.1  | 0.1002 | 0.0969 | 0.0884 | 0.0870 | 0.8138 | 0.8607 |
| 10   | 0    | 0.0601 | 0.0587 | 0.0614 | 0.0639 | 0.7926 | 0.8300 |
|      | 0.1  | 0.0639 | 0.0625 | 0.0652 | 0.0632 | 0.7952 | 0.8337 |
| 10   | 0    | 0.0612 | 0.0636 | 0.0416 | 0.0397 | 0.7807 | 0.8060 |
|      | 0.1  | 0.0438 | 0.0422 | 0.0443 | 0.0457 | 0.7826 | 0.8124 |
| 10   | 0    | 0.0469 | 0.0453 | 0.0475 | 0.0456 | 0.7847 | 0.8116 |
| 2    | 0    | 0.0510 | 0.0494 | 0.0518 | 0.0498 | 0.7874 | 0.8181 |
|      | 0.1  | 0.0559 | 0.0545 | 0.0569 | 0.0549 | 0.7910 | 0.8237 |
|      | 0.5  | 0.0623 | 0.0610 | 0.0636 | 0.0617 | 0.7958 | 0.8303 |
| 5    | 0    | 1.3008 | 1.2930 | 1.3021 | 1.3021 | 2.3709 | 3.4282 |
|      | 0.1  | 1.3668 | 1.3572 | 1.3097 | 1.3097 | 2.4189 | 3.0703 |
| 10   | 0    | 1.0951 | 1.0606 | 1.4232 | 1.4232 | 2.7460 | 3.2162 |
| 1    | 0.3  | 1.8524 | 1.8124 | 1.5607 | 1.5607 | 2.5584 | 3.0800 |
|      | 0.4  | 2.0584 | 2.0118 | 1.7274 | 1.7274 | 2.6645 | 3.7881 |
|      | 0.5  | 2.3183 | 2.2649 | 1.9632 | 1.9632 | 2.7985 | 3.5030 |
| 5    | 0    | 0.9754 | 0.9715 | 0.7414 | 0.7414 | 2.1619 | 3.2866 |
|      | 0.1  | 1.0415 | 1.0326 | 0.7953 | 0.7953 | 2.1910 | 2.9860 |
| 2    | 0.3  | 1.2325 | 1.1123 | 0.8655 | 0.8655 | 2.2263 | 2.5162 |
|      | 0.3  | 1.2248 | 1.2080 | 0.9575 | 0.9575 | 2.2689 | 2.9415 |
|      | 0.4  | 1.3558 | 1.3329 | 1.0845 | 1.0845 | 2.3296 | 3.1860 |
|      | 0.5  | 1.5255 | 1.4955 | 1.2532 | 1.2532 | 2.4061 | 2.9414 |
| 10   | 0    | 0.6116 | 0.6144 | 0.5636 | 0.5696 | 2.0510 | 2.9641 |
|      | 0.1  | 0.6506 | 0.6519 | 0.6023 | 0.6037 | 2.0888 | 2.6100 |
| 1    | 0    | 0.6978 | 0.6975 | 0.6444 | 0.6433 | 2.0907 | 2.7000 |
|      | 0.3  | 0.7572 | 0.7552 | 0.6948 | 0.6936 | 2.1213 | 2.4602 |
|      | 0.4  | 0.8361 | 0.8319 | 0.7668 | 0.7636 | 2.1615 | 3.0898 |
|      | 0.5  | 0.9410 | 0.9344 | 0.8655 | 0.8596 | 2.2123 | 2.6386 |
| 10   | 0    | 0.9347 | 0.9465 | 0.4946 | 0.4934 | 1.9710 | 2.6668 |
|      | 0.1  | 0.4603 | 0.4683 | 0.4244 | 0.4323 | 1.9879 | 2.7417 |
| 2    | 0    | 0.4891 | 0.4961 | 0.4495 | 0.4569 | 2.0011 | 2.5429 |
|      | 0.3  | 0.5255 | 0.5309 | 0.4823 | 0.4887 | 2.0168 | 2.6908 |
|      | 0.4  | 0.5721 | 0.5761 | 0.5246 | 0.5298 | 2.0386 | 2.8260 |
|      | 0.5  | 0.6490 | 0.6501 | 0.5956 | 0.5974 | 2.0753 | 2.5289 |
Table 4.3: MISEs of different methods under fixed design with missing both in a contiguous block and at random.

| $m$ | PSNR | Grid Missing Rate | TPST $\triangle_1$ | TPST $\triangle_2$ | ATPST $\triangle_1$ | ATPST $\triangle_2$ | Thin Plate Product | Tensor Product |
|-----|------|------------------|---------------------|---------------------|---------------------|---------------------|------------------|----------------|
| 5   |      |                  |                     |                     |                     |                     |                  |                |
| 1   | 0.12 | 1.643            | 0.1628              | 0.1561              | 0.1554              | 0.9323              | 0.8481           |                |
|     | 0.2  | 1.710            | 0.1691              | 0.1625              | 0.1618              | 0.9439              | 0.8536           |                |
|     | 0.3  | 1.812            | 0.1798              | 0.1737              | 0.1731              | 0.9598              | 0.8626           |                |
|     | 0.4  | 1.985            | 0.1972              | 0.1895              | 0.1893              | 0.9852              | 0.8760           |                |
|     | 0.5  | 2.171            | 0.2158              | 0.2085              | 0.2082              | 1.0197              | 0.8915           |                |
|     | 2    | 1.1168           | 0.1142              | 0.1081              | 0.1072              | 0.8728              | 0.8152           |                |
|     | 0.2  | 1.224            | 0.1193              | 0.1138              | 0.1129              | 0.8769              | 0.8193           |                |
|     | 0.3  | 1.303            | 0.1282              | 0.1226              | 0.1215              | 0.8874              | 0.8257           |                |
|     | 0.4  | 1.397            | 0.1373              | 0.1312              | 0.1304              | 0.9004              | 0.8334           |                |
|     | 0.5  | 1.537            | 0.1521              | 0.1453              | 0.1446              | 0.9182              | 0.8438           |                |
| 10  |      |                  |                     |                     |                     |                     |                  |                |
| 1   | 0.12 | 0.0794           | 0.0792              | 0.0847              | 0.0825              | 0.8345              | 0.7969           |                |
|     | 0.2  | 0.0827           | 0.0824              | 0.0870              | 0.0858              | 0.8499              | 0.7992           |                |
|     | 0.3  | 0.0883           | 0.0882              | 0.0930              | 0.0915              | 0.8560              | 0.8028           |                |
|     | 0.4  | 0.0963           | 0.0963              | 0.1010              | 0.0996              | 0.8656              | 0.8081           |                |
|     | 0.5  | 0.1057           | 0.1058              | 0.1102              | 0.1088              | 0.8758              | 0.8143           |                |
| 2   | 0.12 | 0.0559           | 0.0550              | 0.0582              | 0.0574              | 0.8211              | 0.7843           |                |
|     | 0.2  | 0.0584           | 0.0579              | 0.0613              | 0.0600              | 0.8210              | 0.7860           |                |
|     | 0.3  | 0.0628           | 0.0623              | 0.0659              | 0.0646              | 0.8250              | 0.7887           |                |
|     | 0.4  | 0.0676           | 0.0672              | 0.0709              | 0.0696              | 0.8298              | 0.7916           |                |
|     | 0.5  | 0.0746           | 0.0743              | 0.0781              | 0.0768              | 0.8370              | 0.7959           |                |
| 5   |      |                  |                     |                     |                     |                     |                  |                |
| 1   | 0.12 | 1.7368           | 1.7060              | 1.2766              | 1.2741              | 2.6896              | 2.4587           |                |
|     | 0.2  | 1.8429           | 1.8063              | 1.3653              | 1.3625              | 2.8285              | 2.4906           |                |
|     | 0.3  | 1.9842           | 1.9453              | 1.5069              | 1.4934              | 2.9021              | 2.5590           |                |
|     | 0.4  | 2.1575           | 2.1107              | 1.6449              | 1.6452              | 2.9690              | 2.6538           |                |
|     | 0.5  | 2.4060           | 2.3554              | 1.8783              | 1.8608              | 3.2190              | 2.7878           |                |
| 10  |      |                  |                     |                     |                     |                     |                  |                |
| 1   | 0.12 | 1.1926           | 1.1747              | 0.8785              | 0.8785              | 2.3785              | 2.2406           |                |
|     | 0.2  | 1.2536           | 1.2322              | 0.8332              | 0.8321              | 2.3984              | 2.2651           |                |
|     | 0.3  | 1.3544           | 1.3262              | 0.9096              | 0.9064              | 2.4507              | 2.3055           |                |
|     | 0.4  | 1.4779           | 1.4502              | 1.0130              | 1.0103              | 2.5459              | 2.3520           |                |
|     | 0.5  | 1.6444           | 1.6110              | 1.1742              | 1.1695              | 2.6820              | 2.4205           |                |
| 2   | 0.12 | 0.7024           | 0.7038              | 0.6167              | 0.6224              | 2.1975              | 2.1215           |                |
|     | 0.2  | 0.7394           | 0.7398              | 0.6505              | 0.6539              | 2.2987              | 2.1310           |                |
|     | 0.3  | 0.7904           | 0.7901              | 0.6970              | 0.6986              | 2.3045              | 2.1567           |                |
|     | 0.4  | 0.8546           | 0.8514              | 0.7534              | 0.7533              | 2.2841              | 2.1929           |                |
|     | 0.5  | 0.9470           | 0.9423              | 0.8374              | 0.8360              | 2.3999              | 2.2423           |                |
| 10  |      |                  |                     |                     |                     |                     |                  |                |
| 1   | 0.12 | 0.5122           | 0.5192              | 0.4347              | 0.4353              | 2.1131              | 2.0442           |                |
|     | 0.2  | 0.5328           | 0.5388              | 0.4618              | 0.4709              | 2.1144              | 2.0534           |                |
|     | 0.3  | 0.5670           | 0.5709              | 0.4999              | 0.4970              | 2.1341              | 2.0682           |                |
|     | 0.4  | 0.6093           | 0.6135              | 0.5270              | 0.5342              | 2.1744              | 2.0855           |                |
|     | 0.5  | 0.6673           | 0.6696              | 0.5804              | 0.5870              | 2.2680              | 2.1112           |
A. Detailed Introduction of Tetrahedral Partitions

A.1. Tetrahedral partitions

To recall you memory, we use $T$ to denote a tetrahedron, that is, a convex hull of four noncoplanar points in $\mathbb{R}^3$.

**Definition A.1** (Definition 16.5 in Lai and Schumaker (2007)). A collection $\triangle = \{T_1, \ldots, T_N\}$ of $N$ tetrahedra is called a **tetrahedral partition** of $\Omega = \bigcup_{h=1}^N T_h$ provided that any pair of tetrahedra in $\triangle$ intersect at most at a common vertex, along a common edge, or along a common triangular face.

An example of tetrahedral partition is illustrated in Figure A.1(a), where tetrahedron $T_1 = \langle v_5, v_1, v_4, v_3 \rangle$ and tetrahedron $T_2 = \langle v_2, v_1, v_3, v_4 \rangle$ form a tetrahedral partition of $\Omega = \langle v_2, v_5, v_3, v_4 \rangle$. In contrast, the partition in Figure A.1 (b) is not a tetrahedral partition, in which tetrahedra $T_4 = \langle v_2, v_1, v_6, v_4 \rangle$ and $T_3 = \langle v_2, v_1, v_3, v_6 \rangle$ form a tetrahedral partition of $\langle v_2, v_1, v_3, v_4 \rangle$, but tetrahedra $T_1$ and $T_3$ and $T_4$ do not form a tetrahedral partition of $\Omega$. In general, any type of polygon shapes can be used for the partition of $\Omega$.

**Definition A.2** (Definition 15.4 in Lai and Schumaker (2007)). A tetrahedron $T = \langle v_1, v_2, v_3, v_4 \rangle$ with $\{v_i := (x_i, y_i, z_i)\}_{i=1}^4$ is **nondegenerate** provided that it has nonzero volume, and the vertices of $T$ are in **canonical order** provided that if we rotate and translate $T$ so that the triangular face $\langle v_1, v_2, v_3 \rangle$ lies in the $xy$-plane with $v_1, v_2, v_3$ in counterclockwise order, then $z_4 > 0$.

**Definition A.3** (Definition 16.1 in Lai and Schumaker (2007)). Let $|T|$ be the length of the longest edge of $T$, and $\varrho_T$ be the radius of the largest ball which can be inscribed in $T$ (inradius of $T$), then the ratio $\beta_T := |T|/\varrho_T$ is called the **shape parameter** of $T$.

**Remark A.1.** The shape parameter $\beta_T$ describes the shape of $T$. If $T$ is a tetrahedron whose six edges are all of the same length, then $\beta_T = 2\sqrt{6}$. Such a tetrahedron is called a **regular tetrahedron**. For any other tetrahedron, $\varrho_T$ is larger, and the larger $\varrho_T$ becomes, the flatter the tetrahedron $T$ becomes. Another way to describe the shape of a tetrahedron is in terms of certain angles at the vertices of $T$.

**Definition A.4.** A tetrahedral partition $\triangle$ is **$\beta$-quasi-uniform** if there is a positive value $\beta$ such that $\triangle$ satisfies

$$|\triangle|/\varrho_T \leq \beta < \infty, \quad \text{for all } T \in \triangle,$$  \hspace{1cm} (A.1)
Figure A.1: (a) and (b) provides an illustration of an example and a counterexample of tetrahedral partition; (c) shows an illustration of the shape parameters; (d) shows an example of a non-β-quasi-uniform tetrahedral partition; and (e) provides an example of two tetrahedra sharing a common face.
where $|\triangle| := \max\{|T|, T \in \triangle\}$ is referred as the size of $\triangle$, i.e., the length of the longest edge of $\triangle$.

Given a domain $\Omega$, we seek a tetrahedral partition whose minimum angle is as large as possible. When $\beta$ is small, the tetrahedra are relatively uniform in the sense that all angles of tetrahedra in the tetrahedral partition $\triangle$ are relatively the same. Thus, the tetrahedral partition looks more like a uniform tetrahedral partition which has the largest minimum angle. Figure A.1 (d) gives an example of non-$\beta$-quasi-uniform tetrahedral partition

$$\triangle = \{T_1 = \langle v_2, v_1, c_1, v_4 \rangle, T_2 = \langle v_1, c_2, c_1, v_4 \rangle, T_3 = \langle c_1, v_1, c_3, c_2 \rangle, \]
$$

where the edges $\langle v_4, c_1 \rangle$ and $\langle c_2, c_3 \rangle$ are perpendicular to the edge $\langle v_2, v_3 \rangle$, and $\langle c_3, c_4 \rangle$ are perpendicular to $\langle v_3, v_4 \rangle$.

Let $N$ be the number of the tetrahedra in the polygonal domain $\Omega$. From (A.1), we have

$$N \leq (4\pi|\triangle|^3)^{-1}3V_\Omega\beta^3,$$

where $V_\Omega$ denotes the volume of $\Omega$.

A.2. Computer representation of tetrahedra partitions

A widely adopted data structure for efficiently storing a tetrahedral mesh is using two separate matrices for nodes and tetrahedra elements in the mesh. We first use a 3-column matrix to store all the nodes in the partition, where each row corresponds to the Cartesian coordinates $(x, y, z)$ of one node. Then, each tetrahedron within the partition can be represented concisely by specifying the indices of the four nodes that define the tetrahedron. Therefore, the list of tetrahedra elements can be simplified to a matrix with four columns.

**Example A.1.** Two adjacent tetrahedra $T = \langle v_2, v_1, v_3, v_4 \rangle$ and $\tilde{T} = \langle v_5, v_1, v_4, v_3 \rangle$ share a common triangular face $F = \langle v_1, v_3, v_4 \rangle$, as illustrated in Figure A.1 (e), where the Cartesian coordinates of the five vertices are $v_1 = (0, 0, 0)$, $v_2 = (1, 0, 0)$, $v_3 = (0, 1, 0)$, $v_4 = (0, 0, 1)$, $v_5 = (-1, 0, 0)$, respectively.

A.3. Delaunay tetrahedral partitions

Delaunay triangulation is one of the central topics in mesh generation due to its theoretical properties that have been well-studied in 2D space (Chew, 1989; Lee and Schachter, 1980) and can be extended to 3D or even higher. In this section, we mainly focus on Delaunay tetrahedral partition in the 3D Euclidean space. Given a set of vertices $\mathcal{V}$, a tetrahedral partition $\triangle$ is called Delaunay tetrahedral partition if for every
tetrahedron $T$ in $\triangle$, there is no vertex $v \in V$ lying in the interior of the ball passing through the vertices of $T$. The 3D Delaunay tetrahedral partition maximizes the minimal solid angle appearing in the tetrahedra in $\triangle$. Chapter 16 in [Lai and Schumaker (2007)] provides a detailed discussion of various theoretical properties for tetrahedral partitions.

![Figure A.2: Tetrahedral partitions that can/cannot form a Delaunay partition.](image)

Due to the mathematical guarantees of mesh quality, there have been extensive efforts to develop Delaunay-based algorithms that can efficiently handle arbitrarily complicated geometry for 2D domains ([Ruppert, 1995]; [Shewchuk, 1996]). However, extending these algorithms to 3D is challenging. In the past three decades, many studies have been conducted regarding the construction of tetrahedral partitions given complex domains ([Shewchuk, 1998]; [Shewchuk and Si, 2014]; [Cheng et al., 2005]; [Jamin et al., 2015]). Different in the ways of handling domain boundaries and improving mesh quality, the tetrahedralization methodologies can be roughly characterized into three groups: boundary-constrained, Delaunay refinement, and variational methods; see [Si (2015)] for more detailed descriptions.

The Delaunay-based mesh generation algorithms for 3D domains have been embedded in computational softwares, for example, the `delaunayTriangulation` and `delaunay` functions in MATLAB. It has also been implemented in various software packages and toolboxes, including `QualMesh` ([Cheng et al., 2005]), the CGAL library ([Jamin et al., 2015]; [The CGAL Project, 2020]) for C++, `TetGen` program ([Si, 2015]), `iso2mesh` ([Fang and Boas, 2009]) built on CGAL and `TetGen` for MATLAB/Octave, and some MATLAB functions such as `distmesh/distmeshnd` ([Persson and Strang, 2004]). These implementations differ in their methodologies of handling domain boundaries, generating quality meshes, and input formats, and therefore are suitable for various applications. For example, the functions `distmesh` and `distmeshnd` take distance functions as representations
of boundary geometry, while \texttt{delaunayTriangulation} generates Delaunay 2D/3D triangulation based on given point sets. If mathematical representations of boundary geometry or the vertex sets for Delaunay triangulations are not immediately available for complex real 3D objects, one can also use \texttt{TetGen} which requires triangular surface mesh on domain boundaries as input.

We illustrate the overall procedure of mesh generation for a general 3D object with the above tools using the example of a cow in Figure \ref{fig:cow}. Given a 3D object, we first identify the boundary of the domain of interest and generate a triangular surface mesh of the boundary using \texttt{CGALmesh} or \texttt{iso2mesh}. Then, a tetrahedral partition for the complex object can be generated using \texttt{TetGen} or \texttt{iso2mesh}. As one of the most commonly used programs for generating quality tetrahedral meshes, \texttt{TetGen} is based on a mixture of the boundary constrained method (George et al., 1991) and the Delaunay refinement method (Ruppert, 1995; Shewchuk, 1998). It takes triangular surface meshes as inputs and then conducts Delaunay tetrahedralization followed by constrained Delaunay tetrahedralization to preserve some desired constraints, such as domain boundaries and special geometrical features. After these steps, it further applies the Delaunay refinement techniques in boundary constrained methods to improve the quality of tetrahedralizations based on user-specified criteria. The introduced programs and toolboxes have greatly facilitated the construction of tetrahedral meshes used in a wide range of applications, such as medical imaging analysis, engineering, computer science, and others. Figure \ref{fig:mesh} presents a few mesh examples we generated with these tools.

**B. Detailed Introduction of Trivariate Splines on Tetrahedra**

In this section, we give the detailed discussion of the trivariate splines on tetrahedra, along with the introduction for some theoretical properties. In Section B.1, we first introduce the barycentric coordinates associated with a tetrahedron and show that a trivariate polynomial can be written in a convenient form using the barycentric coordinates. In Section B.2, we describe the directional derivatives of a polynomial, and smoothness conditions for polynomials on adjoining triangular face. The introduction of trivariate splines on a tetrahedra partition is given in Section 2.3 in main part.

**B.1. Barycentric coordinates and Bernstein basis polynomials**

Given a tetrahedron $T = \langle \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4 \rangle$, any fixed point $\mathbf{v} := (x, y, z) \in \mathbb{R}^3$ has a unique representation in terms of $\langle \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4 \rangle$,

$$\mathbf{v} = b_1 \mathbf{v}_1 + b_2 \mathbf{v}_2 + b_3 \mathbf{v}_3 + b_4 \mathbf{v}_4, \quad \text{with } b_1 + b_2 + b_3 + b_4 = 1,$$

where $(b_1, b_2, b_3, b_4)$ are called the barycentric coordinates of $\mathbf{v}$ relative to the tetrahedron.
$T$. When the point $v$ is inside or on the faces of $T$, all $b_1, b_2, b_3$ and $b_4$ are nonnegative. By Cramer’s rule, the barycentric coordinate corresponding to vertex $v_i$, satisfies $b_i = \det(M_i)/\det(M), i = 1, \ldots, 4$, where

$$M := \begin{pmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{pmatrix}$$

and $M_i$ replaces $M$’s $i$-th column with $(1 \ x \ y \ z)^\top$.

The barycentric coordinates $(b_1, b_2, b_3, b_4)$ also have an interesting geometric interpretation. As shown in Figure B.1 for any $v \in T$, it divides the tetrahedron $T$ to four sub-tetrahedra, $T_1, T_2, T_3$ and $T_4$. Notice that $\det(M) = 6V_T$, then $b_i = V_{T_i}/V_T$, where $T_i$ replace the vertex $v_i$ in $V_T$ with $v, i = 1, \ldots, 4$.

![Figure B.1: Illustration of barycentric coordinates of a point $v$ in a tetrahedron $T$.](image)

For a nondegenerate tetrahedron $T$ and a point $v \in T$ with barycentric coordinates $(b_1, b_2, b_3, b_4)$, for nonnegative integers $i, j, k, l$ with $i + j + k + l = d$, define trivariate Bernstein basis polynomial of degree $d$ relative to $T$ as

$$B^{d,T}_{ijkl}(v) := \frac{d!}{i!j!k!l!} b_1^i b_2^j b_3^k b_4^l,$$ with $i + j + k + l = d$.

For any positive integer $d$ and tetrahedron $T$, let $\mathcal{P}_d(T)$ be the space of all trivariate polynomials defined on $T$ with degrees less than or equal to $d$. Note that the dimension of $\mathcal{P}_d(T)$ is $\binom{d+3}{3}$.

According to Theorem 15.8 in Lai and Schumaker (2007) and Lemma [D.1] in the Appendix, the set of Bernstein basis polynomials

$$B^d_T(v) := \{B^{d,T}_{ijkl}(v) : i, j, k, l \geq 0, i + j + k + l = d\}$$
forms a basis for the space of polynomials $\mathcal{P}_d(T)$. In addition, Bernstein basis functions $\{B_{ijkl}^{d,T}(\mathbf{v})\}_{i+j+k+l=d}$ have the following properties:

1. $\{B_{ijkl}^{d,T}\}$ form a partition of unity, i.e., for all $\mathbf{v} \in T$, $\sum_{i+j+k+l=d} B_{ijkl}^{d,T}(\mathbf{v}) = 1$;

2. for all $\mathbf{v} \in T$, $0 \leq B_{ijkl}^{d,T}(\mathbf{v}) \leq 1$;

3. $B_{ijkl}^{d,T}$ has a unique maximum at the point $d^{-1}(iv_1 + jv_2 + kv_3 + lv_4)$.

**Remark B.1.** Barycentric coordinates are invariant to linear transformations of Cartesian coordinates, that is, they do not depend on the orientation or location of the tetrahedra. Consequently, the trivariate splines based on Bernstein basis polynomials, which are constructed with barycentric coordinates, are also invariant to linear transformations.

To further illustrate the Bernstein basis functions, we present an example of $\{B_{ijkl}^{d,T}(\mathbf{v})\}_{i+j+k+l=d}$ for $d = 4$ in Figure B.2. In this example, there are $\binom{4}{3} = 35$ basis functions in total. Note that all the function values vary between 0 and 1, and the colors scale to the quantiles of function values.

Thus, given Bernstein basis functions $\{B_{ijkl}^{d,T}(\mathbf{v})\}_{i+j+k+l=d}$, any polynomial $p(\mathbf{v}) \in \mathcal{P}_d(T)$ can be written uniquely as $\mathbf{B}$-form,

$$p(\mathbf{v})|_T = \sum_{i+j+k+l=d} \gamma_{T;ijkl} B_{ijkl}^{d,T}(\mathbf{v}) = \mathbf{B}_T^d(\mathbf{v}) \gamma_T,$$

where the coefficients $\gamma_T = \{\gamma_{T;ijkl}\}_{i+j+k+l=d}$ are called $\mathbf{B}$-coefficients of $p$. For the purpose of computer implementation, in this paper, we employ the lexicographical order for ordering of the coefficients $\gamma_T$. To be specific, $\gamma_{T;ijkl}$ orders ahead of $\gamma_{T;i'j'k'l'}$ either (i) $i > i'$, or (ii) $i = i'$ and $j > j'$, or (iii) $i = i', j = j'$ and $k > k'$, or (iv) $i = i', j = j', k = k'$ and $l > l'$. Consequently, we can express

$$\mathbf{B}_T^d(\mathbf{v}) = \left( B_{d,0,0,0}^{d,T}(\mathbf{v}), B_{d-1,1,0,0}^{d,T}(\mathbf{v}), B_{d-1,0,1,0}^{d,T}(\mathbf{v}), \ldots, B_{0,0,1,d-1}^{d,T}(\mathbf{v}), B_{0,0,0,d}^{d,T}(\mathbf{v}) \right)^\top$$

and

$$\gamma_T = (\gamma_{T;d,0,0,0}, \gamma_{T;d-1,1,0,0}, \gamma_{T;d-1,0,1,0}, \ldots, \gamma_{T;0,0,1,d-1}, \gamma_{T;0,0,0,d})^\top. \tag{B.2}$$

Accordingly, in (B.2), the index of the element $\gamma_{T;ijkl}$ in the vector $\gamma_T$ is:

$$\sum_{m=0}^{d-i} \frac{(m+1)m}{2} + \sum_{n=0}^{d-i-j} (n+1) - k.$$

Note that using a different ordering method will not affect the evaluation results for the trivariate polynomial functions.
Figure B.2: Illustrations of Bernstein basis functions \( \{ B_{ijkl}(v) \} \).
It is convenient to derive conditions of continuous connection for polynomials defined on adjacent tetrahedra in using barycentric coordinates and Bernstein basis polynomials. Consider Example A.1 illustrated in Figure A.1(e), for two tetrahedra \( T = \langle v_2, v_1, v_3, v_4 \rangle \) and \( \tilde{T} = \langle v_5, v_1, v_4, v_3 \rangle \) sharing a common triangular face \( F = \langle v_1, v_3, v_4 \rangle \), assume two sets of Bernstein polynomial basis \( \{B_{ijkl}^d(T)(v)\}_{i+j+k+l=d} \) and \( \{\tilde{B}_{ijkl}^d(T)(v)\}_{i+j+k+l=d} \) are defined on \( T \) and \( \tilde{T} \) using the barycentric coordinates, respectively. Consider two degree-
\( d \) polynomials \( p(v) \) and \( \tilde{p}(\tilde{v}) \) defined on \( T \) and \( \tilde{T} \), respectively, with B-forms

\[
p(v) = \sum_{i+j+k+l=d} \gamma_{ijkl} B_{ijkl}^d(T)(v), \quad \tilde{p}(\tilde{v}) = \sum_{i+j+k+l=d} \tilde{\gamma}_{ijkl} \tilde{B}_{ijkl}^d(\tilde{v}).
\]

For \( w \in F \), the barycentric coordinates with respect to \( T \) and \( \tilde{T} \) are \((0, b_1, b_3, 1-b_1-b_3)\) and \((0, b_1, 1-b_1-b_3, b_3)\), respectively. Accordingly, we have

\[
p(w) = \sum_{j+k+l=d} \gamma_{ijkl} \frac{d!}{j!k!l!} b_1^j b_3^k (1-b_1-b_3)^l,
\]

\[
\tilde{p}(w) = \sum_{j+k+l=d} \tilde{\gamma}_{ijkl} \frac{d!}{j!k!l!} b_1^j (1-b_1-b_3)^k b_3^l.
\]

Therefore, \( p \) and \( \tilde{p} \) are continuous on \( F \) if and only if

\[
\gamma_{ijkl} = \tilde{\gamma}_{ijkl}
\]

for \( j, k, l \geq 0 \) and \( j + k + l = d \).

**B.2. Directional derivatives and smoothness**

To generalize the smoothness restriction over the joint triangular face for two adjacent tetrahedra, we need to introduce the definitions of directional derivative first. For a general multivariate smooth function \( p \), the directional derivative at point \( v \) with respect to direction \( u \) is defined as

\[
D_u p(v) := \frac{\partial}{\partial t} p(v + tu) \bigg|_{t=0} = \lim_{t\to 0} \frac{p(v + tu) - p(v)}{t}.
\]

Accordingly, for vector \( u := (u_x, u_y, u_z) \in \mathbb{R}^3 \) and trivariate function \( p \), the directional derivative at \( v = (x, y, z) \) is

\[
D_u p(x, y, z) := \frac{\partial}{\partial t} p(x + tu_x, y + tu_y, z + tu_z) \bigg|_{t=0}.
\]

**Remark B.2.** If \( p \) is a polynomial of degree \( d \), by calculus, \( D_u p(x, y, z) = u_x D_x p(x, y, z) + u_y D_y p(x, y, z) + u_z D_z p(x, y, z) \), so \( D_u p \) is a polynomial of \( d-1 \).
Consider direction \( \mathbf{u} = \mathbf{w}_1 - \mathbf{w}_2 \), where for \( i = 1, 2 \), \( \mathbf{w}_i \in \mathbb{R}^3 \) have barycentric coordinates \((w_{1i}, w_{2i}, w_{3i}, w_{4i})\) with respect to \( T \). Then \( \mathbf{u} \) is uniquely described by the directional coordinates \( \mathbf{a} = (a_1, a_2, a_3, a_4) = (w_{1i} - w_{21}, w_{12} - w_{22}, w_{13} - w_{23}, w_{14} - w_{24}) \). Obviously, \( a_1 + a_2 + a_3 + a_4 = 0 \). Direct calculation gives the directional derivative of the Bernstein basis polynomial \( B_{ijkl}^d \).

**Lemma B.1** (Lemma 15.12 in Lai and Schumaker (2007)). Consider direction \( \mathbf{u} \) with directional coordinates \( \mathbf{a} = (a_1, a_2, a_3, a_4) \). Then

\[
D_u B_{ijkl}^d (\mathbf{v}) = d \left\{ a_1 B_{i-1,j,k,l}^{d-1} (\mathbf{v}) + a_2 B_{i,j-1,k,l}^{d-1} (\mathbf{v}) \\
+ a_3 B_{i,j,k-1,l}^{d-1} (\mathbf{v}) + a_4 B_{i,j,k,l-1}^{d-1} (\mathbf{v}) \right\},
\]

for any \( \mathbf{v} \in T \) and \( i + j + k + l = d \).

Consequently, one can obtain the directional derivative for any trivariate polynomial \( p \).

**Theorem B.1** (Theorems 15.13 and 15.14 in Lai and Schumaker (2007)). Consider direction \( \mathbf{u} \) with directional coordinates \( \mathbf{a} = (a_1, a_2, a_3, a_4) \). Then for any trivariate polynomial \( p \) with \( B \)-form in \( \overline{B} \), the directional derivative is

\[
D_u p (\mathbf{v}) = d \sum_{i+j+k+l = d-1} \gamma_{ijkl}^{(1)} (\mathbf{a}) B_{ijkl}^{d-1} (\mathbf{v}),
\]

where

\[
\gamma_{ijkl}^{(1)} (\mathbf{a}) = a_1 \gamma_{i+1,j,k,l} + a_2 \gamma_{i,j+1,k,l} + a_3 \gamma_{i,j,k+1,l} + a_4 \gamma_{i,j,k,l+1}.
\]

In general, given \( \mathbf{u}_1, \ldots, \mathbf{u}_m \) with associated directional coordinates \( \mathbf{a}^{(i)} = (a_1^{(i)}, a_2^{(i)}, a_3^{(i)}, a_4^{(i)}) \), \( i = 1, \ldots, m \),

\[
D_{u_m} \cdots D_{u_1} p (\mathbf{v}) = \frac{d!}{(d-m)!} \sum_{i+j+k+l = d-m} \gamma_{ijkl}^{(m)} (\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m)}) B_{ijkl}^{d-m} (\mathbf{v}),
\]

where the coefficients are defined recursively as follows:

\[
\gamma_{ijkl}^{(m)} (\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m)}) = a_1^{(m)} \gamma_{i+1,j,k,l}^{(m-1)} (\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m-1)}) + a_2^{(m)} \gamma_{i,j+1,k,l}^{(m-1)} (\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m-1)}) \\
+ a_3^{(m)} \gamma_{i,j,k+1,l}^{(m-1)} (\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m-1)}) + a_4^{(m)} \gamma_{i,j,k,l+1}^{(m-1)} (\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m-1)}),
\]

for \( i = 1, \ldots, m \), \( m = 1, \ldots, d \), with \( \gamma_{ijkl}^{(0)} (\mathbf{a}) = \gamma_{ijkl} \).

In preparation for the discussion of trivariate spline and spline spaces, we need the following conditions for a smooth join between two polynomial on adjoining tetrahedra, like Example A.1 illustrated in Figure A.1 (e).
**Theorem B.2.** Suppose \( \{ \gamma_{ijkl} \} \) and \( \{ \tilde{\gamma}_{ijkl} \} \) are B-coefficients of \( p \) and \( \tilde{p} \) relative to two tetrahedra \( T = \langle \mathbf{v}_2, \mathbf{v}_1, \mathbf{v}_3, \mathbf{v}_4 \rangle \) and \( \tilde{T} = \langle \mathbf{v}_5, \mathbf{v}_1, \mathbf{v}_4, \mathbf{v}_3 \rangle \), respectively, where \( T \) and \( \tilde{T} \) share a common face \( F = \langle \mathbf{v}_1, \mathbf{v}_3, \mathbf{v}_4 \rangle \). Then the following statements are equivalent:

1. \( p \) and \( \tilde{p} \) join together with \( C^r \) continuity across the face \( F \);
2. For all \( w \in F \), \( m = 0, \ldots, r \) and for all directions \( u \),
   \[
   D^m_u p(w) = D^m_u \tilde{p}(w);
   \]
3. For \( p \) and \( \tilde{p} \) with B-forms in \( \text{[B.3]} \), for \( i + j + k = d - m \), \( m = 0, \ldots, r \),
   \[
   \tilde{\gamma}_{mijk} = \sum_{\nu+\mu+k+\delta=m} \gamma_{\nu,i+\mu,k+j+\delta} B^m_{\nu\mu\nu\delta}(\mathbf{v}_5). \tag{B.8}
   \]

Consider the case of \( d = 2 \) piecewise polynomial. There are in total 10 Bernstein basis polynomials with coefficients

\[
\{ \gamma_{ijkl} \} = (\gamma_{2000}, \gamma_{1100}, \gamma_{1010}, \gamma_{1001}, \gamma_{0200}, \gamma_{0110}, \gamma_{0101}, \gamma_{0020}, \gamma_{0011}, \gamma_{0002})^T,
\]
\[
\{ \tilde{\gamma}_{ijkl} \} = (\tilde{\gamma}_{2000}, \tilde{\gamma}_{1100}, \tilde{\gamma}_{1010}, \tilde{\gamma}_{1001}, \tilde{\gamma}_{0200}, \tilde{\gamma}_{0110}, \tilde{\gamma}_{0101}, \tilde{\gamma}_{0020}, \tilde{\gamma}_{0011}, \tilde{\gamma}_{0002})^T. \tag{B.9}
\]

Note that the barycentric coordinate of \( \mathbf{v}_5 \) with respect to \( T \) is \( (2, -1, 0, 0) \).

If the trivariate polynomial is continuous over the whole region, then applying (B.8) for \( m = r = 0 \) generates

\[
\tilde{\gamma}_{0ijk} = \gamma_{0ijk} B^0_{0000}(\mathbf{v}_5) = \gamma_{0ijk}, \quad i + j + k = d, \tag{B.10}
\]

which matches the conclusion in \( \text{[B.4]} \).

Based on the de Casteljau Algorithm (Theorem D.1 in Appendix), we propose the following computationally efficient algorithm to calculate the derivatives of \( p(\mathbf{v}) \).

**B.3. Example of the constraint matrix**

We illustrate the construction of the constraint matrix \( \mathbf{H} \) using the Example A.1 presented in Figure A.1 (e).

Continue our discussions in Section B.2 and consider the case of \( d = 2 \) piecewise polynomial. Recall that there are in total 10 Bernstein basis polynomials, with coefficients \( \{ \gamma_{ijkl} \} \) and \( \{ \tilde{\gamma}_{ijkl} \} \) in \( \text{[B.9]} \), and the barycentric coordinate of \( \mathbf{v}_5 \) with respect to \( T \) is \( (2, -1, 0, 0) \).
Inputs: Polynomial with B-form
\[ p(v) = \sum_{i+j+k+l = d} \gamma_{ijkl}(v) B_{ijkl}^d(v), \]
directions \( u_1, \ldots, u_m \) with
associated directional coordinates
\[ a^{(ii)}(v^1, v^2, v^3, v^4), \]
ii = 1, ..., m

Initialize: \( ii := 0, \gamma_{ijkl}^{(0)} := \gamma_{ijkl} \)

Outputs: \( D_{u_m} \ldots D_{u_1} p(v) \)

for \( ii = 1, \ldots, m \) do
  for \( i + j + k + l = d - ii \) do
    \[ \gamma_{ijkl}^{(ii)}(a^{(1)}, \ldots, a^{(m)}) \]
    \[ = a_{1}^{(ii)} \gamma_{i+1,j,k,l}^{(ii)}(a^{(1)}, \ldots, a^{(i+1-1)}) + a_{2}^{(ii)} \gamma_{i,j+1,k,l}^{(ii)}(a^{(1)}, \ldots, a^{(i-1)}) \]
    \[ + a_{3}^{(ii)} \gamma_{i,j,k+1,l}^{(ii-1)}(a^{(1)}, \ldots, a^{(i-1)}) + a_{4}^{(ii)} \gamma_{i,j,k,l+1}^{(ii-1)}(a^{(1)}, \ldots, a^{(i-1)}), \]
  end
end

\[ D_{u_m} \ldots D_{u_1} p(v) = \frac{d!}{(d-m)!} \sum_{i+j+k+l = d-m} \gamma_{ijkl}^{(m)}(a^{(1)}, \ldots, a^{(m)}) B_{ijkl}^{d-m}(v) \]

Algorithm 1: Algorithm for Derivatives of \( p(v) \).

As shown in (B.10), if the trivariate polynomial is continuous over the whole region, then
\[ \tilde{\gamma}_{0ijk} = \gamma_{0ijk} \]
for \( i + j + k = d \). In this case, we can write the constraint matrix \( H \) as

\[
H = \begin{pmatrix}
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}.
\]

(B.11)

Furthermore, if the trivariate polynomial has continuous first derivatives over the whole region, then (B.8) holds for both \( m = r = 0 \) and \( m = 1 \). Thus, in addition, we also need for any non-negative integers \( i, j, k \) such that \( i + j + k = 1 \),
\[
\tilde{\gamma}_{1ijk} = \gamma_{1ijk} B_{1000}^1(v_5) + \gamma_{0,i+1,j} B_{1010}^3(v_5) + \gamma_{0,i+1,k} B_{1010}^3(v_5) + \gamma_{0,i,k+1} B_{1001}^3(v_5) + \gamma_{0,i,k+1} B_{0001}^3(v_5)
\]
\[ = 2 \gamma_{1ijk} - \gamma_{0,i+1,k,j}. \]

(B.12)

Therefore, for \( r = 1 \) where the trivariate spline has continuous first derivatives over the
whole region, we will obtain the following $H$

$$
H = \begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 2 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
\end{pmatrix}.
$$

Here, the first six rows match the form in (B.11), and the last three rows correspond to (B.12).

C. Detailed Implementation of TPST

C.1. Directional derivatives for basis functions

Based on the conclusion in (B.7), the $m$th order directional derivatives for all the Bernstein basis functions with degree $d$ can be written as some linear combination of Bernstein basis functions with degree $d - m$. Specifically, for directions $u_1, \ldots, u_m$, there exists a $\binom{d+3}{3} \times \binom{d+3-m}{3}$ matrix $C^{(m)}_{d}(u_1, \ldots, u_m)$ such that

$$
D_{u_m} \cdots D_{u_1} B_d(v) = C^{(m)}_{d}(u_1, \ldots, u_m) B_{d-m}(v).
$$

Remark C.1. Suppose $D_{u_{m-1}} \cdots D_{u_1} B_d(v) = C^{(m-1)}_{d}(u_1, \ldots, u_{m-1}) B_{d-m+1}(v)$, then we have

$$
D_{u_m} D_{u_{m-1}} \cdots D_{u_1} B_d(v) = C^{(m)}_{d}(u_1, \ldots, u_{m-1}) D_{u_m} B_{d-m+1}(v) = C^{(m-1)}_{d}(u_1, \ldots, u_{m-1}) C^{(1)}_{d-m+1}(u_m) B_{d-m}(v),
$$

which implies $C^{(m)}_{d}(u_1, \ldots, u_m) = C^{(m-1)}_{d}(u_1, \ldots, u_{m-1}) C^{(1)}_{d-m+1}(u_m)$. Keep decomposing matrix $C^{(m-1)}_{d}(u_1, \ldots, u_{m-1})$, we can have

$$
C^{(m)}_{d}(u_1, \ldots, u_m) = C^{(1)}_{d}(u_1) \cdots C^{(1)}_{d-m+1}(u_m).
$$

Next, we introduce the explicit form of matrix $C^{(1)}_{d}(u)$. Recall that $D_{u} B_{d}^{(1)}(v)$ can be written as

$$
d \left\{ a_1 B^{d-1}_{i_1,j,k,l}(v) + a_2 B^{d-1}_{i_2,j,k,l}(v) + a_3 B^{d-1}_{i_3,j,k,l}(v) + a_4 B^{d-1}_{i_4,j,k,l}(v) \right\},
$$

where $(a_1, a_2, a_3, a_4)$ is the barycentric coordinate of direction $u$. Then, the $(\sum_{m=0}^{d-i} \binom{m+1}{2} + \sum_{n=0}^{d-i-j} (n+1) - k)$th row of matrix $C^{(1)}_{d}(u)$ is $d(a_1 e_{I_1} + a_2 e_{I_2} + a_3 e_{I_3} + a_4 e_{I_4})$, where $e_{I}$
is unit vector with $I$th element being one, $I_1, I_2, I_3$ and $I_4$ are indexes of basis functions $B_{t-1,j,k,l}^{d-1}(v), B_{t,j-1,k,l}^{d-1}(v), B_{t,j,k,l-1}^{d-1}(v)$, and $B_{t,j,k,l}^{d-1}(v)$.

In the following, we provide a simple example for the first and second order derivatives. Consider a case on the tetrahedron $T$ in the Figure [B.1] with $d = 3$. The barycentric coordinates of $x = v_1 - v_2$. Then, the first and second order derivatives of all the Bernstein basis function are $D_x B_3(v) = C_3^{(1)}(x) B_2(v)$ and $D_{x x} B_3(v) = C_3^{(2)}(x, x) B_1(v) = C_3^{(1)}(x) C_2^{(1)}(x) B_1(v)$, where matrices $C_2^{(1)}(x)$ and $C_3^{(1)}(x)$ are

$$C_2^{(1)}(x) = \begin{pmatrix} 2 & 0 & 0 & 0 \\ -2 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & -2 & 0 & 0 \\ 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad C_3^{(1)}(x) = \begin{pmatrix} 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & -3 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & -3 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & -3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 \end{pmatrix}.$$

C.2. Details of constructing penalty matrix

In this section, we introduce the details of constructing penalty $P$ matrix. Applying the results in Section C.1, the second order derivative of Bernstein basis functions $B_{d}(v)$ at directions $u_1$ and $u_2$ are

$$D_{u_2} D_{u_1} B_{d}(v) = C^{(2)}(u_1, u_2) B_{d-2}(v).$$

Therefore, for directions $g, g' = x, y, z$, we have

$$\int_{T} \left\{ D^2_{gg'} B_{d}(v) \right\} \left\{ D^2_{gg'} B_{d}(v) \right\}^{\top} dv = C^{(2)}(g, g') \int_{T} B_{d-2}(v) B_{d-2}(v)^{\top} dv \left\{ C_{d}^{(2)}(g, g') \right\}^{\top}.$$

Let $L_{d-2}^{T} = \int_{T} B_{d-2}(v) B_{d-2}(v)^{\top} dv$ be the $(d+1) \times (d+1)$ matrix with entries $\int_{T} D_{\nu \mu \delta}^{d-2} D_{\nu' \mu' \delta'}^{d-2} (v) dv$.
By the Lemma 15.29 in \cite{Lai2007}, we have

\[
\int_T B_{\nu,\mu,\kappa,\delta}^{T,d-2}(v) B_{\nu',\mu',\kappa',\delta'}^{T,d-2}(v) \, dv = \frac{(\nu+\nu')(\mu+\mu')(\kappa+\kappa')(\delta+\delta')}{2d-4} \left( \int_T B_{\nu+\nu',\mu+\mu',\kappa+\kappa',\delta+\delta'}^{T,2d-4}(v) \, dv \right) V_T,
\]

recall that \( V_T \) is the volume of tetrahedron \( T \). We finally obtain

\[
P_{gg'}^T = C_d^{(2)}(g,g') L_T^{d-2} \{ C_d^{(2)}(g,g') \}^\top
\]

and

\[
P_T = \sum_{g,g' \in \{x,y,z\}} P_{gg'}^T.
\]

Recall that \( \mathcal{E}(s) = \sum_{T \in \Delta} \mathcal{E}(s_T) \) and \( \mathcal{E}(s_T) = \gamma_T^\top P_T \gamma_T \). Then, one can have \( \mathcal{E}(s) = \gamma^\top P \gamma \), where \( \gamma = (\gamma_1^\top, \ldots, \gamma_N^\top)^\top \), and \( P = \text{diag}(P_T, T \in \Delta) \) is the block diagonal matrix.

### C.3. An example of the penalty matrix

In this section, we provide a simple example for \( P_T \). Consider a case on the tetrahedron \( T \) in the Figure \ref{fig:example} with \( d = 3 \). The barycentric coordinates of \( x = v_1 - v_2 \), \( y = v_3 - v_2 \) and \( z = v_4 - v_2 \) are \((1, -1, 0, 0), (0, -1, 1, 0), \) and \((0, -1, 0, 1)\), respectively. The second directional derivative of these Bernstein basis polynomials are the linear combination of \( B_{1000}^T(v), \ldots, B_{0001}^T(v) \). For example, we have \( D_x B_{2100}^T(v) = 3B_{1100}^T(v) - 3B_{2000}^T(v) \), and \( D_{xx} B_{2100}^T(v) = -12B_{1000}^T(v) + 6B_{0100}^T(v) \). According to the results in Section C.1, the coefficient matrices for second order derivatives are \( C_3^{(2)}(x,x), C_3^{(2)}(y,y), C_3^{(2)}(z,z)\),
$C_3^{(2)}(x, y), C_3^{(2)}(x, z),$ and $C_3^{(2)}(y, z)$

$$
\begin{align*}
C_3^{(2)}(x, x) &= \begin{pmatrix}
6 & 0 & 0 & 0 \\
-12 & 6 & 0 & 0 \\
0 & 0 & 6 & 0 \\
0 & 0 & 0 & 6 \\
6 & -12 & 0 & 0 \\
0 & 0 & -12 & 0 \\
0 & 0 & 0 & -12 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix}, \\
C_3^{(2)}(y, y) &= \begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 6 & 0 \\
0 & 0 & 0 & 6 \\
-12 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
6 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix}, \\
C_3^{(2)}(z, z) &= \begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
-12 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix},
\end{align*}
$$

$$
\begin{align*}
C_3^{(2)}(x, y) &= \begin{pmatrix}
0 & 0 & 0 & 0 \\
-6 & 0 & 0 & 0 \\
6 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
6 & -6 & 0 & 0 \\
-6 & 6 & -6 & 0 \\
-6 & 6 & 0 & -6 \\
0 & 0 & 6 & 0 \\
0 & 0 & 0 & 6 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 6 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix}, \\
C_3^{(2)}(x, z) &= \begin{pmatrix}
0 & 0 & 0 & 0 \\
-6 & 0 & 0 & 0 \\
6 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
6 & -6 & 0 & 0 \\
-6 & 6 & -6 & 0 \\
-6 & 6 & 0 & -6 \\
0 & 0 & 6 & 0 \\
0 & 0 & 0 & 6 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 6 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix}, \\
C_3^{(2)}(y, z) &= \begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
-12 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix},
\end{align*}
$$

35
Applying the results in Section C.2, we have
\[
L_T^1 = \frac{1}{12} \begin{pmatrix}
2 & 1 & 1 & 1 \\
1 & 2 & 1 & 1 \\
1 & 1 & 2 & 1 \\
1 & 1 & 1 & 2
\end{pmatrix},
\]
and
\[
P_T = \sum_{g, g' \in \{x, y, z\}} C_3^{(2)}(g, g') L_T^1 \{C_3^{(2)}(g, g')\}^T.
\]

C.4. Penalty parameter selection

To balance the bias and variance of the proposed estimator and achieve a good estimation and prediction performance, it is crucial to choose a suitable value of the penalty parameter \(\lambda\). A large \(\lambda\) tends to shrink the second-order derivative function of splines functions to zero and gives a smooth function with large fitting errors. On the other hand, a small \(\lambda\) may generate a more wiggly function and result in an overfitting problem.

Note that 3D object data are often generated with spatial dependence. When performing cross-validation (CV), these dependence structures are usually ignored, leading to underestimating the predictive error (Roberts et al., 2017). To tackle this problem, we adopt the block cross-validation strategy in Roberts et al. (2017) and Valavi et al. (2019). All the sample points are first divided into 3D blocks with similar volumes. Then, these blocks are randomly allocated to the CV folds. In this paper, we adopt the tetrahedral partition to divide the 3D object data into small 3D blocks. Each tetrahedron is considered as one single 3D block. Figure C.1 shows an illustration of block CV using a tetrahedral partition. In Figure C.1, we divide the domain into 384 equal-size tetrahedra and randomly assign these tetrahedra into five folds with color indicating different folds.

D. Proof of Theorems

D.1. Theoretical Properties of Bernstein Bases

Lemma D.1. For the Bernstein basis functions \(\{B_{ijkl}^{d}(v)\}_{i+j+k+l=d}\), each \(B_{ijkl}^{d}\) has a unique maximum at the point \(d^{-1}(iv_1 + jv_2 + kv_3 + lv_4)\).

To show Lemma D.1, we first need to show Lemma B.1.

Proof of Lemma B.1. Suppose \(b = (b_1, b_2, b_3, b_4)\) are the barycentric coordinates of \(v\). Then the barycentric coordinates of \(v + tu\) are \((b_1 + ta_1, b_2 + ta_2, b_3 + ta_3, b_4 + ta_4)\). Thus,
Figure C.1: Illustration of block CV using tetrahedral partition. Tetrahedra with same color belong to same folds.

for $i + j + k + l = d,$

$$B^d_{ijkl}(v + tu) = \frac{d!}{i!j!k!l!} (b_1 + ta_1)^i(b_2 + ta_2)^j(b_3 + ta_3)^k(b_4 + ta_4)^l.$$ 

Hence,

$$D_u B^d_{ijkl}(v) = \frac{\partial}{\partial t} p(v + tu) \bigg|_{t=0}$$

$$= \frac{d!}{i!j!k!l!} (ia_1b_1^{i-1}b_2^j b_4 + ja_2b_1^{j-1}b_3^k b_4 + ka_3b_1^k b_2^l b_4 + la_4b_1^l b_3^k b_4^{l-1})$$

$$= d \{a_1 B^d_{i-1,j,k,l}(v) + a_2 B^{d-1}_{i,j-1,k,l}(v) + a_3 B^{d-1}_{i,j,k-1,l}(v) + a_4 B^{d-1}_{i,j,k,l-1}(v)\}.$$ 

\[\square\]

**Proof of Lemma [D.1]** For $T = \langle v_1, v_2, v_3, v_4 \rangle$, the barycentric coordinates for $v_1, v_2, v_3$, and $v_4$ are $(1,0,0,0), (0,1,0,0), (0,0,1,0)$, and $(0,0,0,1)$, respectively. Accordingly, the directional coordinates of $u_1 = v_1 - v_2$, $u_2 = v_1 - v_3$, and $u_3 = v_1 - v_4$ are $(1,-1,0,0), (1,0,-1,0)$, and $(1,0,0,-1)$, respectively. For $v \in T$ with barycentric coordinates $b = (b_1, b_2, b_3, b_4)$, consider derivatives of $B^d_{ijkl}(v)$ with respect to directions $u_1, u_2$ and $u_3$, then

$$D_{u_1} B^d_{ijkl}(v) = B^d_{ijkl}(v)(ib_1^{-1} - jb_2^{-1}), \quad D_{u_2} B^d_{ijkl}(v) = B^d_{ijkl}(v)(ib_1^{-1} - kb_3^{-1}),$$

$$D_{u_3} B^d_{ijkl}(v) = B^d_{ijkl}(v)(ib_1^{-1} - lb_4^{-1}).$$

Set these equations to zero and combine $b_1 + b_2 + b_3 + b_4 = 1$ gives $(b_1, b_2, b_3, b_4) = d^{-1}(i, j, k, l).$ 

\[\square\]
D.2. Proof of Theorems B.1 and B.2

Proof of Theorem B.1. For \( p(v) = \sum_{i+j+k+l=d} \gamma_{ijkl} B_{ijkl}^d(v) \), by Lemma B.1

\[
D_u p(v) = \sum_{i+j+k+l=d} \gamma_{ijkl} D_u B_{ijkl}^d(v)
\]

\[
= \sum_{i+j+k+l=d} \gamma_{ijkl} \left\{ a_1 B_{i-1,j,k,l}^{d-1}(v) + a_2 B_{i,j-1,k,l}^{d-1}(v) + a_3 B_{i,j,k-1,l}^{d-1}(v) + a_4 B_{i,j,k,l-1}^{d-1}(v) \right\}
\]

\[
= d \sum_{i+j+k+l=d} \left( a_1 \gamma_{i+1,j,k,l} + a_2 \gamma_{i,j+1,k,l} + a_3 \gamma_{i,j,k+1,l} + a_4 \gamma_{i,j,k,l+1} \right) B_{ijkl}^{d-1}(v).
\]

Thus, (B.6) follows. Consequently, one can obtain (B.7) by repeatedly applying (B.6) for directions \( u_1, \ldots, u_m \).

To show Theorem B.2, we need the following results.

Theorem D.1 (Theorems 15.10 in Lai and Schumaker (2007)). Suppose \( p(v) \) is a trivariate polynomial with B-form \( p(v) = \sum_{i+j+k+l=d} \gamma_{ijkl} B_{ijkl}^d(v) \). Define \( \gamma_{ijkl}^{(0)} := \gamma_{ijkl} \), \( i + j + k + l = d \). Suppose \( v \) has barycentric coordinates \( b = (b_1, b_2, b_3, b_4) \). Then

\[
p(v) = \sum_{i+j+k+l=d-m} \gamma_{ijkl}^{(m)} B_{ijkl}^{d-m}(v),
\]

where for \( m = 1, \ldots, d \), \( \gamma_{ijkl}^{(m)} \) are computed by the recursion

\[
\gamma_{ijkl}^{(m)} = b_1 \gamma_{i+1,j,k,l}^{(m-1)} + b_2 \gamma_{i,j+1,k,l}^{(m-1)} + b_3 \gamma_{i,j,k+1,l}^{(m-1)} + b_4 \gamma_{i,j,k,l+1}^{(m-1)}, \quad (D.1)
\]

for \( i + j + k + l = d - m \).

Remark D.1. The recursive formula (D.1) is also referred as de Casteljau algorithm. See Section 15.6 in Lai and Schumaker (2007) for more details.

Lemma D.2. The coefficients in the recursive formula of de Casteljau algorithm (D.1) are given by

\[
\gamma_{ijkl}^{(m)} = \sum_{i'+j'+k'+l' = m} \gamma_{i'+j'+k'+l'+v} B_{i'l'}^{m}(v), \quad (D.2)
\]

where \( i + j + k + l = d - m \).

Proof. Define operators \( E_1 \gamma_{ijkl} = \gamma_{i+1,j,k,l} \), \( E_2 \gamma_{ijkl} = \gamma_{i,j+1,k,l} \), \( E_3 \gamma_{ijkl} = \gamma_{i,j,k+1,l} \) and \( E_4 \gamma_{ijkl} = \gamma_{i,j,k,l+1} \). Thus, by (D.1), for \( i + j + k + l = d - m \),

\[
\gamma_{ijkl}^{(m)} = (b_1 E_1 + b_2 E_2 + b_3 E_3 + b_4 E_4) \gamma_{ijkl}^{(m-1)} = (b_1 E_1 + b_2 E_2 + b_3 E_3 + b_4 E_4)^m \gamma_{ijkl}
\]

\[
= \sum_{i'+j'+k'+l' = m} B_{i'l'}^{m}(v) E_1^{i'} E_2^{j'} E_3^{k'} E_4^{l'} \gamma_{ijkl} = \sum_{i'+j'+k'+l' = m} \gamma_{i'+j'+k'+l'+v} B_{i'l'}^{m}(v).
\]

\[
\square
\]
Lemma D.3. Suppose \( p(v) \) is a trivariate polynomial with B-form \( p(v) = \sum_{i+j+k+l=d} \gamma_{ijkl} B_{ijkl}^{d}(v) \). Then for any \( 1 \leq n \leq d \), the \( n \)-th order directional derivative of \( p \) with respect to the direction \( u = v_4 - v_2 \) is given by

\[
D_{u}^{n} p(v) = \frac{d!}{(d-n)!} \sum_{i+j+k+l=d-n} \gamma_{ijkl}^{(n)} B_{ijkl}^{d-n}(v)
\]

with

\[
\gamma_{ijkl}^{(n)} = \sum_{m=0}^{n} \binom{n}{m} (-1)^{m} \gamma_{i+m,j,k,l+n-m}, \quad i + j + k + l = d - n.
\]

Proof. The directional coordinates of \( u = v_4 - v_2 \) is \( a = (-1,0,0,1) \). By \((D.2)\),

\[
\gamma_{ijkl}^{(n)} = \sum_{i'+j'+k'+l'=n} \gamma_{i'+j'+j'+k'+l'+l} B_{ijkl}^{m}(v) = \sum_{i'+j+k+l+n} \gamma_{i'+j+k,l+n-l} \frac{n!}{l!n!} (-1)^{l'}
\]

\[
= \sum_{m=0}^{n} \binom{n}{m} (-1)^{m} \gamma_{i+m,j,k,l+n-m}, \quad i + j + k + l = d - n.
\]

Proof of Theorem B.2. The equivalence between Statements 1 and 2 is obvious by definition. Thus, we just show the equivalence between Statements 2 and 3.

i) We start to consider when \( r = 0 \). It is equivalent to consider directions \( \tilde{u} \) along \( F \), where the directional coordinates are \((0,1-\tilde{b}_3-\tilde{b}_4,\tilde{b}_3,\tilde{b}_4)\) and \((0,1-\tilde{b}_3-\tilde{b}_4,\tilde{b}_4,\tilde{b}_3)\) with respect to the tetrahedron \( T \) and \( \tilde{T} \), respectively. Thus, \( p \) and \( \tilde{p} \) join continuously along \( F \) if and only if

\[
\sum_{j+k+l=d} \gamma_{ijkl} \frac{d!}{j!k!l!} (1-\tilde{b}_3-\tilde{b}_4)j(\tilde{b}_3)k(\tilde{b}_4)l = \sum_{j+k+l=d} \tilde{\gamma}_{ijkl} \frac{d!}{j!k!l!} (1-\tilde{b}_3-\tilde{b}_4)j(\tilde{b}_3)k(\tilde{b}_4)l,
\]

That is, \( \gamma_{ijkl} = \tilde{\gamma}_{ijkl} \), with \( j + k + l = d \), which matches our conclusion in \((B.8)\).

ii) Then we consider for \( r > 0 \). First note that

\[
D_{u}^{n} p(v) = D_{u}^{n} \tilde{p}(\tilde{v})
\]

(holds for any \( v \in F \), \( n = 0, \ldots, r \), if and only if \((D.3)\) holds for the direction \( u = v_5 - v_3 \). It is a fact because by the argument in i), all derivatives of \( p \) and \( \tilde{p} \) corresponding to the directions \( v_3 - v_1 \) and \( v_4 - v_1 \), agree at every point on \( F \). And derivatives in all other directions can be written as linear combinations of \( D_u \), \( D_{v_3-v_1} \) and \( D_{v_4-v_1} \).

Let \( b = (b_1,b_2,b_3,b_4) \) be the barycentric coordinates of \( v_5 \) relative to the tetrahedron \( T \). Correspondingly, the directional coordinates of \( u \) are \( a = (b_1,b_2,b_3-1,b_4) \) and \( \tilde{a} = (1,0,0,-1) \) with respect to \( T \) and \( \tilde{T} \), respectively.
By Theorem [B.1], for each $0 \leq n \leq r$,

$$D^n u p(v)|_F = \frac{d!}{(d-n)!} \sum_{j+k+l=d-n} \gamma_{0jkl}^{(n)}(a) B_{0jkl}^{d-n}(v),$$

$$D^n \tilde{u} p(v)|_F = \frac{d!}{(d-n)!} \sum_{j+k+l=d-n} \gamma_{0jkl}^{(n)}(\tilde{a}) B_{0jkl}^{d-n}(v).$$

Since for points $v \in F$, $\tilde{B}_{0jkl}^{d-n}(v) = B_{0jkl}^{d-n}(v)$, it follows that (D.3) holds if and only if for $j + k + l = d - n$, $n = 0, \ldots, r$,

$$\tilde{\gamma}_{0jkl}^{(n)}(\tilde{a}) = \gamma_{0jkl}^{(n)}(a). \quad \text{(D.4)}$$

By Lemma [D.3] for $j + k + l = d - n$,

$$\tilde{\gamma}_{0jkl}^{(n)}(\tilde{a}) = \sum_{m=0}^{n} (-1)^{n-m} \binom{n}{m} \tilde{\gamma}_{m,j,k,d-m-j-k}^{(m)}(a). \quad \text{(D.5)}$$

In another direction, following the proof of Lemma [D.2] for $j + l + k = d - n$,

$$\gamma_{0jkl}^{(n)}(a) = \{b_1 E_1 + b_2 E_2 + (b_3 - 1) E_3 + b_4 E_4\}^n \gamma_{0jlk}$$

$$= \{b_1 E_1 + b_2 E_2 + b_3 E_3 + b_4 E_4 - E_3\}^n \gamma_{0jlk}$$

$$= \sum_{m=0}^{n} (-1)^{n-m} \binom{n}{m} (b_1 E_1 + b_2 E_2 + b_3 E_3 + b_4 E_4)^n \gamma_{0,j,l+n-m,k}^{(m)}$$

$$= \sum_{m=0}^{n} (-1)^{n-m} \binom{n}{m} \gamma_{0,j,d-j-k-m,k}^{(m)}(b). \quad \text{(D.6)}$$

Combining (D.5) and (D.6), (D.4) holds if and only if for $j + k + l = d - n$ and $n = 0, \ldots, r$,

$$\tilde{\gamma}_{njk} = \gamma_{njk}^{(n)}(b).$$

By Lemma [D.2],

$$\gamma_{0jkl}^{(n)}(b) = \sum_{i' + j' + k' + l' = n} \gamma_{i',j',k',l'+l} B_{i',j',k',l'}^{n}(v_5).$$

Thus, (D.3) holds if and only if

$$\tilde{\gamma}_{njk} = \sum_{i' + j' + k' + l' = n} \gamma_{i',j',k',l'+l} B_{i',j',k',l'}^{n}(v_5),$$

that is, (B.8) follows. \hspace{1cm} \square
References

Chai, X., Gu, H., Li, F., Duan, H., Hu, X., and Lin, K. (2020), “Deep learning for irregularly and regularly missing data reconstruction,” *Scientific Reports*, 10, 1–18.

Cheng, S. W., Dey, T. K., Ramos, E. A., and Ray, T. (2005), “Quality meshing of polyhedra with small angles,” *International Journal of Computational Geometry & Applications*, 15, 421–461.

Chew, L. P. (1989), “Constrained delaunay triangulations,” *Algorithmica*, 4, 97–108.

Chung, M. K., Wang, Y., and Wu, G. (2018), “Discrete Heat Kernel Smoothing in Irregular Image Domains,” in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 5101–5104.

Chung, M. K., Wang, Y., and Wu, G. (2018), “Discrete Heat Kernel Smoothing in Irregular Image Domains,” in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 5101–5104.

Duchon, J. (1976), “Fonctions-spline à énergie invariante par rotation,” Tech. Rep. 27, Université de Grenoble.

Fang, Q. and Boas, D. A. (2009), “Tetrahedral mesh generation from volumetric binary and grayscale images,” in *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pp. 1142–1145.

George, P. L., Hecht, F., and Saltel, E. (1991), “Automatic mesh generator with specified boundary,” *Computer Methods in Applied Mechanics and Engineering*, 92, 269–288.

Goldsmith, J., Huang, L., and Crainiceanu, C. M. (2014), “Smooth scalar-on-image regression via spatial Bayesian variable selection,” *Journal of Computational and Graphical Statistics*, 23, 46–64.

Green, P. J. and Silverman, B. W. (1994), *Nonparametric regression and generalized linear models: a roughness penalty approach*, Chapman and Hall, London.

Guo, J., Duan, X., and Shum, C. (2010), “Non-isotropic Gaussian smoothing and leakage reduction for determining mass changes over land and ocean using GRACE data,” *Geophysical Journal International*, 181, 290–302.

Huo, S., Liu, G., Zhang, J., and Song, C. (2020), “A smoothed finite element method for octree-based polyhedral meshes with large number of hanging nodes and irregular elements,” *Computer Methods in Applied Mechanics and Engineering*, 359, 112646.

Jamin, C., Alliez, P., Yvinec, M., and Boissonnat, J.-D. (2015), “CGALmesh: A Generic Framework for Delaunay Mesh Generation,” *ACM Transactions on Mathematical Software (TOMS)*, 41, 1–24.
Kang, Y. (2020), “Consistent blind image deblurring using jump-preserving extrapolation,” *Journal of Computational and Graphical Statistics*, 29, 372–382.

Kim, M. and Wang, L. (2021), “Generalized spatially varying coefficient models,” *Journal of Computational and Graphical Statistics*, 30, 1–10.

Lai, M. J. (1989), “On construction of bivariate and trivariate vertex splines on arbitrary mixed grid partitions,” PhD dissertation, Texas A&M University.

Lai, M. J. and Schumaker, L. L. (2007), *Spline functions on triangulations*, Cambridge University Press.

Lai, M. J. and Wang, L. (2013), “Bivariate penalized splines for regression,” *Statistica Sinica*, 23, 1399–1417.

Lee, D.-T. and Schachter, B. J. (1980), “Two algorithms for constructing a Delaunay triangulation,” *International Journal of Computer & Information Sciences*, 9, 219–242.

Lindgren, F., Rue, H., and Lindström, J. (2011), “An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach,” *Journal of the Royal Statistical Society, Series B*, 73, 423–498.

Morris, J. S. and Carroll, R. J. (2006), “Wavelet-based functional mixed models,” *Journal of the Royal Statistical Society, Series B*, 68, 179–199.

Persson, P. O. and Strang, G. (2004), “A simple mesh generator in MATLAB,” *SIAM Review*, 46, 329–345.

Ramsay, T. (2002), “Spline smoothing over difficult regions,” *Journal of the Royal Statistical Society, Series B*, 64, 307–319.

Reiss, P. T. and Ogden, R. T. (2010), “Functional generalized linear models with images as predictors,” *Biometrics*, 66, 61–69.

Roberts, D. R., Bahn, V., Ciuti, S., Boyce, M. S., Elith, J., Guillera-Arroita, G., Hauenstein, S., Lahoz-Monfort, J. J., Schröder, B., Thuiller, W., et al. (2017), “Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure,” *Ecography*, 40, 913–929.

Ruppert, J. (1995), “A Delaunay refinement algorithm for quality 2-dimensional mesh generation,” *Journal of Algorithms*, 18, 548–585.
Shewchuk, J. R. (1996), “Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator,” in *Applied Computational Geometry: Towards Geometric Engineering*, eds. Lin, M. C. and Manocha, D., Springer, Berlin, Heidelberg, vol. 1148 of *Lecture Notes in Computer Science*, pp. 203–222.

— (1998), “Tetrahedral mesh generation by Delaunay refinement,” in *Proceedings of the Fourteenth Annual ACM Symposium on Computational Geometry*, pp. 86–95.

Shewchuk, J. R. and Si, H. (2014), “Higher-quality tetrahedral mesh generation for domains with small angles by constrained delaunay refinement,” in *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, pp. 290–299.

Si, H. (2015), “TetGen, a Delaunay-based quality tetrahedral mesh generator,” *ACM Transactions on Mathematical Software (TOMS)*, 41, 1–36.

Stone, C. J. (1994), “The use of polynomial splines and their tensor products in multivariate function estimation,” *The Annals of Statistics*, 22, 118–171.

The CGAL Project (2020), *CGAL User and Reference Manual*, CGAL Editorial Board, 5.1.1 ed.

Valavi, R., Elith, J., Lahoz-Monfort, J. J., and Guillera-Arroita, G. (2019), “blockCV: An R package for generating spatially or environmentally separated folds for k-fold cross-validation of species distribution models,” *Methods in Ecology and Evolution*, 10, 225–232.

Wang, H. and Ranalli, M. G. (2007), “Low-rank smoothing splines on complicated domains,” *Biometrics*, 63, 209–217.

Wilhelm, M., Dedè, L., Sangalli, L. M., and Wilhelm, P. (2016), “IGS: an IsoGeometric approach for Smoothing on surfaces,” *Computer Methods in Applied Mechanics and Engineering*, 302, 70–89.

Wong, R. K., Lee, T. C., Paul, D., and Peng, J. (2016), “Fiber direction estimation, smoothing and tracking in diffusion MRI,” *The Annals of Applied Statistics*, 10, 1137–1156.

Wood, S. N. (2003), “Thin plate regression splines,” *Journal of the Royal Statistical Society, Series B*, 65, 95–114.

Yue, Y. and Speckman, P. L. (2010), “Nonstationary spatial Gaussian Markov random fields,” *Journal of Computational and Graphical Statistics*, 19, 96–116.
Zhu, H., Fan, J., and Kong, L. (2014), “Spatially varying coefficient model for neuroimaging data with jump discontinuities,” *Journal of the American Statistical Association*, 109, 1084–1098.