Restricted Trivariate Polycube Splines for Volumetric Data Modeling

Kexiang Wang, Xin Li, Bo Li, Huanhuan Xu, Hong Qin

Abstract—This paper presents a theoretical volumetric modeling framework to construct a novel spline scheme called restricted trivariate polycube splines (RTP-splines). The RTP-spline aims to generalize both trivariate T-splines and tensor-product B-splines, with a special emphasis on using solid polycube structure as underlying parametric domains and strictly bounded blending functions within such domains. Volumetric RTP-splines are uniquely constructed in a top-down fashion, through four major steps: (1) extending the polycube domain to its bounding volume via space filling; (2) building B-spline volume over the extended domain with restricted boundaries; (3) inserting duplicate knots through adding anchor points and performing local refinement; and (4) removing exterior cells and anchors. Besides local refinement inherited from general T-splines, our RTP-splines have following attractive advantages: (a) naturally modeling solid objects with complicated topologies/bifurcations as a one-piece continuous representation without domain trimming/patching/merging, (b) guaranteed semi-standardness [21] so that the functions and derivatives evaluation is very efficient, (c) restricted support regions of blending functions, preventing control points from influencing other nearby domain regions that stay opposite to the immediate boundaries. These features are strongly desirable for certain applications such as isogeometric analysis. We conduct extensive experiments on converting complicated solid models into RTP-splines, and demonstrate the proposed spline to be a powerful and promising tool for volumetric modeling and other scientific/engineering applications where multi-attribute datasets are prevalent.

1 INTRODUCTION

Volumetric data of massive size are now available in a wide variety of scientific and research fields, because of the rapid advancement of modern data acquisition technologies. A frequently occurring problem is how to convert acquired 3D raw data of discrete samples into a continuous representation upon which simulation and analysis processes can be efficiently developed and accurately computed. The majority of traditional solid modeling techniques during the past four decades have been established upon the following theoretic foundations: constructive solid geometry (CSG), boundary representation (B-reps), and cell/space decomposition. Most of these representations lack the ability of smoothly modeling solid geometry, which is required by modern engineering design in order to directly apply physical simulations on modeled solids, without the necessity of expensive remeshing of finite-element structure and shape data conversion between discrete and continuous representations and between linear finite elements and higher piecewise splines in 3D. In practice, real-world objects (directly acquired via the scanning process) have complex geometry and non-trivial topologies. Therefore, constructing efficient representations for general solid objects in favor of physical simulation and engineering design remains to be a very challenging task. Trivariate simplex splines [7] have been developed to model multi-dimensional, material attributes of volumetric objects. However, computing blending functions and their derivatives on simplex splines is not straightforward and inefficient, compared with NURBS and tensor-product B-splines. Also, how to place boundary knots to avoid numerical degeneracies remains an open problem. Trivariate simplex splines are defined over an unstructured tetrahedral grid, which can be easily obtained from triangular meshes by certain mesh generation softwares such as Tetgen [23]. Although solid object of complex topologies and geometries can be modeled by trivariate simplex splines upon such unstructured grids, the majorities of simulation solvers have preferences on structured grid. This is because, low-quality tetrahedral meshes usually cause large simulation errors or numerical instability. Motivated by current industrial practice in various engineering design and analysis systems, we focus on designing a volumetric spline modeling framework based on structured grid domains.

In the framework of isogeometric analysis proposed by [8], [35], trivariate tensor-product B-splines/NURBS are directly used for modeling smooth geometry, material attributes, and physical simulation of solid objects simultaneously. Martin et al. [17] convert a solid femur mesh to a cylindrical trivariate B-spline by parameterizing the model into a solid cylinder. Due to the topological limitation of the cylinder domain, the constructed trivariate tensor-product splines can not model solid objects with bifurcations and arbitrary topologies, without enormous efforts in patch gluing/trimming, and imposing smoothness constraints along patch boundaries. Furthermore, local refinement required in level-of-detail modeling is not supported by tensor-product splines because basis function refinement will introduce many superfluous control points across the entire domain. As an extension to NURBS, T-splines [21], [22] solve this problem on semi-regular grid domains. To the best of our knowledge, no work has generalized T-splines for three dimensional, multi-attribute data and directly applied them to volumetric geometry and data modeling.

Directly generalizing T-spline surface to volumetric data is not straightforward. A general T-spline function defined...
over a bivariate domain can be formulated as
\[
F(u, v) = \sum_{i=1}^{n} w_i \mathbf{p}_i B_i(u, v) / \sum_{i=1}^{n} w_i B_i(u, v) \quad (u, v) \in \mathbb{R}^2,
\]
where \( \mathbf{p}_i \) are control points associated with weight \( w_i \), and \( B_i(u, v) \) denote basis functions. With this definition, two pieces of T-spline patches can be stitched together by blending boundary basis functions, and we form a new T-spline that can preserve smoothness across the boundary. Trivariate T-splines inherit such nice features, and T-splines defined on polycube volumetric domains can be similarly constructed by gluing a group of T-spline cubes. However, the calculation of this T-spline function and its derivatives requires to divide blending functions by the sum of all the contributed ones. This will make the evaluation computationally inefficient. Recently, Semi-standard T-splines introduced in [21] guarantee \( \sum_{i=1}^{n} w_i B_i(u, v) \equiv 1 \) in Eq (1) across the entire domain. In this setting, computation of \( F(u, v) \) and its derivatives can be much more efficient.

![Fig. 1. Extra support regions. On a concave domain, if the supporting box of a blending function intersects with the domain boundary (e.g., boxes of \( v_1 \) and \( v_2 \)), extra control points (e.g., in red regions) could contribute to the function blending unnecessarily.](image)

However, how to construct a semi-standard T-spline, especially over non-trivial parametric domains, is a challenging problem. Another issue is that, conventional T-splines are defined with open boundaries, i.e., the support regions of blending functions may go across the domain boundaries. Such an open-boundary scheme upon polycube domain will cause control points to unnecessarily contribute to extra domain regions. Two examples are shown as red regions in Figure 1. This might cause geometric inconsistency in modeling underlying solid objects, and in physical simulations. Therefore, it is ideal to have a trivariate spline inherit from T-splines, that (1) is defined within the largest visible region inside the domain, and (2) has the property of semi-standardness. Such novel splines will greatly facilitate direct modeling and physical simulations of arbitrary solid objects with complex geometries and sophisticated topologies. The spline constructed in this paper has these properties, and we call it the Restricted Trivariate Polycube Spline (RTP-spline). We present a framework of RTP-splines construction and the data conversion of volumetric models to this spline representation.

The main contributions of this paper include:

1) A new spline (RTP-spline) scheme is uniquely formulated over polycube domain, with blending functions restricted inside domain boundaries. The RTP-splines also have the following advantages:
   - It is capable of local refinement;
   - Computing RTP-spline functions and their derivatives is much more efficient than that on traditional T-spline surfaces;
   - The polycube domain enables natural modeling of arbitrary solid objects, since low distortions and few singularity points are introduced in volumetric parametrization when the domain mimics the geometries and topologies properly;
   - The restricted boundaries of RTP-spline effectively ensure the physical modeling and simulations adhere to geometry of underlying objects.

2) We develop a novel framework to construct RTP-splines in an effective top-down fashion.

3) We construct RTP-splines on several volumetric models with both geometry and synthesized texture information (to mimic material properties), which demonstrates that our RTP-splines can model not only geometry but also multi-attribute fields within an unified paradigm.

The remainder of this paper is organized as follows. We review the related literature in Section 2, then introduce preliminaries and define necessary notations in Section 3. The methodology of RTP-spline construction is illustrated in Section 4. The entire process of converting discrete volumetric data into the spline representation is then explained in Section 5. We demonstrate experimental results in Section 6 and conclude the paper in Section 7.

2 Related Works
Research on spline-based volumetric modeling has gained much attention recently. 4D uniform rational cubic B-spline volume is used to constructively model FRep solids defined by real-valued functions [20]. The method presented in [18] represents and specifies physical attributes across a trivariate NURBS volume. However, it is more desirable in engineering design to have an integrated modeling framework that represents geometry, material attributes, and conducts simulations simultaneously. Trivariate NURBS are used to model skeletal muscle with anisotropic attributes [35], on which NURBS-FEM analysis is directly conducted. Martin et al. [17] present a method based on volumetric harmonic functions to parameterize a volumetric solid to a solid cylinder in order to fit a single trivariate B-spline to geometric data and model simulation attributes. A modeling technique based on triangular simplex spline [7] is developed to model and render multi-dimensional, material attributes for solid objects with complicated geometries and topologies.

The splines proposed in this paper are founded upon the T-spline technique [22]. T-spline is a generalization of NURBS, but permits T-junctions on its control mesh and enables local insertion of additional knots without introducing superfluous control points. A local refinement method is proposed in [4], [21] to simplify NURBS surfaces to T-spline representations by removing superfluous control points. The merge of B-spline patches defined over different
local domains for getting a single T-spline representation on the manifold domain is thoroughly discussed in [9].

Bazilevs et al. [2] propose an isogeometric analysis framework based on T-splines. Its main focus is on planar T-splines for surfaces, and volumetric T-splines is only briefly mentioned without offering any technical details. Generalized trivariate T-splines (whose control points are associated with weights) are employed by [24] to model free-form deformation fields. For the purpose of shape metamorphosis, 3D level sets represented by T-splines are adopted in [5], [31]–[33] for its efficiency. This is because, the distribution of T-spline control points can be made adaptive to the geometry of the morphing objects.

Our work relies on polycube domain construction and parameterization. Parameterization on polycubes originated for seamless texture mapping with low distortion [25]. Polycubes serve as nice parametric domains because it well approximates the geometry of the model and possesses great regularity. Polycube mapping can be constructed either manually [25]–[27] or automatically [6], [14]. Based upon specially-designed surface parametrization, [26] builds manifold bivariate T-spline over a polycube that can handle models with arbitrary topology. A few recent work [12], [15], [16], [28]–[30] studies the parameterization of a solid object to canonical domains such as spheres, polycubes, star-shaped volumes, etc. Volumetric parameterization typically starts from any given surface mapping, and parameterizing volumetric data onto a solid polycube domain serves as an important pre-processing step for the conversion of any solid model to RTP-splines.

3 Preliminaries and Notations

In this section, we introduce the general algorithm to construct trivariate T-spline with duplicate knots on regular box domain, review the theory of basis function refinement, and define necessary notations for the rest of the paper.

3.1 Trivariate T-splines with Duplicate Knots

![Fig. 2. A vertex \( v_i \) can have at most 27 anchors placed on \( 3 \times 3 \times 3 \) virtual grids. The central one (red) is the master anchor and the rest (black) are sub-anchors.](image)

Defined on a grid structure that allows T-junctions (or T-mesh), the T-spline proposed in [22] is a generalization of non-uniform B-splines (or NURBS). When considering a simple cube domain, the definition of T-spline surfaces can be straightforwardly extended to three dimensions and generate trivariate T-splines on T-lattice grids, where “T-junctions” are referred to the intersections between faces and/or lines.

Let \( T(\mathcal{V}, \mathcal{C}, \mathcal{F}) \) denote a rectilinear grid structure that permits T-junctions, where \( \mathcal{V}, \mathcal{C}, \) and \( \mathcal{F} \) are sets of vertices, cells, and faces, respectively. \( \mathcal{K} \subseteq \mathcal{V} \times \{ -1, 0, +1 \}^3 \) denote a set of anchors attached to the vertices. At most 27 anchors are allowed at each vertex, and they can be imagined to be organized on a \( 3 \times 3 \times 3 \) grid of infinitesimal size, as shown in Figure 2. We require each vertex has a master anchor at the center of the local grid, while the others are optional and called sub anchors. In the reset of the paper, we denote an anchor at \( v_i \) as \( k_{i, (\alpha, \beta, \gamma)} \), in which the triplet \( (\alpha, \beta, \gamma) \) indicates a unique nodal position on the local grid. Given \( v_i = (v_{i0}, v_{i1}, v_{i2}) \), all the corresponding anchors \( k_{i, (\alpha, \beta, \gamma)} \) share the same coordinator \( (v_{i0}, v_{i1}, v_{i2}) / \epsilon \) in parametric space. To distinguish these anchors for T-spline construction, we define \( k_{i, (\alpha, \beta, \gamma)} = v_i + (\alpha, \beta, \gamma) \epsilon \) as the coordinator of \( v_{i, (\alpha, \beta, \gamma)} \) in construction space, where \( \epsilon \) is an infinitesimal with respect to the minimal cell size. In the rest of this paper, we sometimes represent an anchor by a simpler notation \( k_{j, \epsilon} \), where \( j \) indicates the index of \( k_{i, (j, \epsilon)} \) in \( \mathcal{K} \).

Given \( T \) and \( \mathcal{K} \), a trivariate T-spline can be defined as

\[
\mathbf{F}(u, v, w) = \frac{\sum_{i=1}^{\|\mathcal{B}\|} p_i B_i(u, v, w)}{\sum_{i=1}^{\|\mathcal{B}\|} B_i(u, v, w)} \in \mathbb{R}^3, \tag{2}
\]

where \((u, v, w)\) denotes 3D parametric coordinates, \( p_i \) are control points, and \( \mathcal{B} = \{ B_i(u, v, w) \} \) is the collection of blending functions. Each \( B_i(u, v, w) \) is a tensor-product of three B-spline basis functions, written as

\[
B_i(u, v, w) = N_{i0}^3(u)N_{i1}^3(v)N_{i2}^3(w), \tag{3}
\]

where \( N_{i0}^3(u) \), \( N_{i1}^3(v) \) and \( N_{i2}^3(w) \) are defined along \( u, v, \) and \( w \) directions, respectively. In the case of cubic T-spline, the univariate function \( N_{i}^3 \) is constructed upon knot vector \( \Xi_i = [\xi_{i0}, \xi_{i1}, \xi_{i2}, \xi_{i3}, \xi_{i4}] \), which is deduced from \( T \) and a collection of anchors \( \mathcal{K} \).

We refer the knot vector in construction space by notation \( \Xi_i = [\xi_{i0}, \xi_{i1}, \xi_{i2}, \xi_{i3}, \xi_{i4}] \) for the rest of the paper, unless mentioned otherwise. In the case of cubic T-spline, each blending function must be associated with an anchor, which coincides with the center of its three knot vectors.

To infer knot vectors from a T-lattice is parallel to that for T-mesh, except that the searching is conducted in construction space. Starting from an anchor \( k = (\xi_{00}, \xi_{01}, \xi_{02}, \xi_{03}, \xi_{04}) \) and \( \xi_{04} \) are found by shooting a ray \( L(t) = (\xi_{00} + t, \xi_{01}, \xi_{02}) \) into construction. \( \xi_{03} \) and \( \xi_{04} \) are the corresponding coordinate values at the first two intersections where \( L(t) \) comes across either an anchor or a face in \( \mathcal{F} \). If \( L(t) \) does not make two intersections before shooting outside \( T \), the last coordinate value is repeated, e.g. \( \xi_{i0} = \xi_{i4} \) or \( \xi_{i2} = \xi_{i3} = \xi_{i4} \) (see Figure 3). The knots in other direction are determined in a similar fashion.
TABLE 1
Refinement of $N(\xi)$ by inserting $k$ into knot vector $[\xi_0, \xi_1, \xi_2, \xi_3, \xi_4]$, which generates two basis functions $N_1(\xi)$ and $N_2(\xi)$, scaled by $c_1$ and $c_2$ respectively.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>knot vector of $N_1(\xi)$</th>
<th>knot vector of $N_2(\xi)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[0, 1, 2]$</td>
<td>$[0, 1, 2]$</td>
<td>$[0, 1, 2]$</td>
<td>$[0, 1, 2]$</td>
<td>$[0, 1, 2]$</td>
</tr>
<tr>
<td>$[0, 1, 2, 3]$</td>
<td>$[0, 1, 2, 3]$</td>
<td>$[0, 1, 2, 3]$</td>
<td>$[0, 1, 2, 3]$</td>
<td>$[0, 1, 2, 3]$</td>
</tr>
<tr>
<td>$[0, 1, 2, 3, 4]$</td>
<td>$[0, 1, 2, 3, 4]$</td>
<td>$[0, 1, 2, 3, 4]$</td>
<td>$[0, 1, 2, 3, 4]$</td>
<td>$[0, 1, 2, 3, 4]$</td>
</tr>
</tbody>
</table>

Fig. 3. Knot vectors are derived from a T-lattice associated with a set of anchors (dots). The knot vector from $k_0$ is $[0, 0.1, 2, 2]$, with $2$ repeating twice because $L_0$ intersects once with the rightmost boundary. The knot vector from $k_0$ toward $-w$ direction has $0$ repeated three times because $L_1$ intersects nothing from $T$ or $K$ except $k_0$.

3.2 Refinement of B-spline Functions
To refine blending functions on trivariate T-splines, we need to review the knot insertion algorithm for univariate B-spline functions. Let $\Xi = [\xi_0, \xi_1, \xi_2, \xi_3, \xi_4]$ be a knot vector and $N(\xi)$ denote the cubic B-spline basis function defined on it. If there is an additional knot $k \in [\xi_0, \xi_4]$ inserted into $\Xi$, $N(\xi)$ can be written as a linear combination of two scaled B-spline functions as

$$N(\xi) = c_1N_1(\xi) + c_2N_2(\xi),$$

where $c_1$, $c_2$ and knot vectors for $N_1(\xi)$ and $N_2(\xi)$, determined by rules in Table 1.

4 CONSTRUCTING RTP-SPLINES
The construction of RTP-splines includes four major steps (see Figure 4): (1) extending given polycube $P$ domain to a box domain, (2) building trivariate B-splines with restricted boundaries, (3) introducing duplicate knots by inserting additional anchors, and performing local refinement to separate interior and exterior blending functions, and (4) producing RTP-splines by removing structures/anchors outside $P$. These steps are discussed in the following four subsections respectively.

4.1 Extending Polycubes to Bounding-Boxes
Following notations introduced in Section 3.1, on the trivariate T-spline domain, let $P = (V^P, C^P, F^P)$ be a given polycube structure, where $V^P$, $C^P$ and $F^P$ denote vertices, cubes and cell faces respectively. In order to extend $P$ to a box volume with rectilinear grids, $P$ should not have T-junctions or intersections between its cell faces. Our parametric polycube domains (see Section 5.1) do not contain T-junctions. If other polycube mapping methods are used to construct the parametric domain and the generated domain has T-junctions, then we can always eliminate them simply by splitting the cells across the domain, through the extended planes of these intersecting cell faces. Now $P$ can be extended to its bounding-box domain $T(V, C, F)$ by filling in some solid cuboid structures $G = (V^G, C^G, F^G)$, where $V^G = V - V^P$, $C^G = C - C^P$, $F^G = F - F^P$. $G$ represents the exterior structure of $P$ and we call its domain the ghost region. Note that there is a rectilinear grid embedded in the space of $T$, and the grids coordinates in $k$-axis direction are represented by

$$S_k = [s^k_1, s^k_2, \ldots, s^k_n]$$

where $n_k$ is the resolution of rectilinear grid along $k$-axis.

4.2 Building the B-spline Volume with Restricted Boundary
With the bounding box domain $T$ constructed, it is difficult to construct a trivariate tensor-product B-spline from the rectilinear grid structure on $T$ by using $S_1$, $S_2$ and $S_3$. We must augment $S_k$ to have a valid B-spline definition on the $n_1 \times n_2 \times n_3$ control grid. One method is to add extra knots outside domain region, generating an open-boundary scheme. In this paper, we repeat the knots at both ends of $S_k$ in order to restrict the B-spline blending function within domain $T$, i.e., $S_k$ turns into

$$S_k = [s^k_1, s^k_1, s^k_1, s^k_2, \ldots, s^k_{n_k-1}, s^k_{n_k}, s^k_{n_k}, s^k_{n_k}]$$

in which $3$ extra knots are added to each end. Therefore, the trivariate tensor-product B-spline defined on $T$ is formulated as

$$F(u, v, w) = \sum_{i=1}^{n} p_i B_i(u, v, w) \quad (u, v, w) \in \mathbb{R}^3$$

where $n = (n_1 + 2) \times (n_2 + 2) \times (n_3 + 2)$ is the number of control points, and $B_i(u, v, w)$ are blending functions defined in Equation (3).

Alternatively, we can obtain $F$ by constructing blending functions similar to T-spline (Section 3.1, instead of computing them from 3 global knot vectors. We let $S = \{s^{0}_0, s^{0}_1 + \epsilon, \ldots, s^{0}_{n_0} - \epsilon, s^{0}_{n_0}\} \times \{s^{1}_1, s^{1}_1 + \epsilon, \ldots, s^{1}_{n_1} - \epsilon, s^{1}_{n_1}\} \times \{s^{2}_1, s^{2}_1 + \epsilon, \ldots, s^{2}_{n_2} - \epsilon, s^{2}_{n_2}\}$ and choose the anchor
Fig. 4. Overview of RTP-spline construction, which consists of four steps: (1) extending polycube domain to its bounding-box, (2) building B-spline volume with bounded boundaries, (3) inserting anchors and refining blending functions, (4) removing exterior regions.

Fig. 5. (a) Knot configuration at corner, edge and face vertices for restricted boundaries. (b) Examples of extraordinary corners on a polycube.

set \( \mathcal{K} = \{ k_{i(\alpha, \beta, \gamma)} | k_{i(\alpha, \beta, \gamma)} \in S \} \), then build blending functions associated with each anchor. \( \mathcal{K} \) contains sub anchors that only exist at corner, edge, and face vertices (see their configurations in Fig 5(a)). These sub anchors guarantee partition-of-unity of \( F \) and limit the influential regions of blending functions within the domain \( T \).

4.3 Local Refinement and Anchor Insertion

Let internal and ghost blending functions refer to the blending functions associated with anchors in \( P \) and \( G \) respectively. In this section, we seek to modify the shape of existing blending functions with knot insertion and local refinement, so that the resulting internal and ghost blending functions are isolated and restricted boundary forms along the surface of \( P \). More precisely, our goal is to enforce the following rules to blending function set:

(i) No ghost blending functions influences any part of the polycube domain.

(ii) Semi-standardness is preserved on internal blending function set if \( G \) and all the ghost anchors are removed.

(iii) No internal blending functions influences any region outside polycube domain if \( G \) and all the ghost anchors are removed.

To achieve this goal, we need to systematically add new anchors in two major steps. First, sub-anchors are added at the vertices on the polycube boundary (Section 4.3.3). Second, we keep inserting sub-anchors to refine the blending functions in violation of the above rules, until there exist no violations. Adding new sub-anchors ultimately introduces duplicate knots into knot vectors, which serves two purposes: (1) reducing the influential region of a blending function and (2) degenerating the continuity of a blending function to \( C^0 \) at desired place (Section 4.3.2). Moreover, as new anchors may lead to disagreements between existing blending functions and underlying knot vectors implied by \( T \) and new \( \mathcal{K} \), an algorithm (Section 4.3.1) is necessary to resolve these inconsistencies after new anchors being inserted.

4.3.1 Local Refinement of Blending Functions

We need to introduce an algorithm to update blending functions \( B \) accordingly, once there is any changes to anchor set \( \mathcal{K} \) and/or domain structure \( T \). The refinement algorithm proposed in [4], [21] is designed for surface editing, the primary goal of which is to preserve the shape of a T-spline surface whenever new control points are inserted. In this paper, we extend this algorithm to 3D and enhance it to support trivariate T-spline with duplicate knots. By interpreting the B-spline volume previously obtained as a general trivariate T-spline, we can rewrite its representation from Equation 5 to

\[
F(u,v,w) = \frac{\sum_{i=1}^{B} w_i B_i(u,v,w)}{\sum_{i=1}^{B} w_i B_i(u,v,w)} \quad (u,v,w) \in \mathbb{R}^3 \tag{6}
\]

where \( w_i \) is the weight associated with each blending function \( B_i \). It’s notable as the T-spline so far is essentially a B-spline volume, \( \sum_{i=1}^{B} w_i B_i(u,v,w) \equiv 1 \) for any \((u,v,w)\) and \( w_i = 1 \) for any \( i \).

Let \( \mathcal{K}^* \) denote an updated anchor set with new anchors and \( T^*(V^*, C^*, F^*) \) be the new grid structure after vertex insertion and/or cell splitting. Given \( \mathcal{K}^* \), \( T^* \), \( V \) and \( B \), the Algorithm 1 generates a new blending function set \( B^* \) and new weights \( W^* \) accordingly.

In Algorithm 1, the superscript indicates the index of the blending function with which a variable is associated and subscript references the central anchor associated with a blending function. For example, \( B^*_t \) is a blending function centered at anchor \( k_t \) that originates from the \( t \)-th blending function in \( B \). The star superscript indicates that the variables are obtained from modified domain \( T^* \),
Algorithm 1: small Refinement of trivariate T-spline blending functions in support of duplicate knots.

**Input:** $T^* (V^*, C^*, \mathcal{K}^*, \mathcal{B}^*)$ and $\mathcal{W}^*$.  
**Output:** $T^*$, $\mathcal{K}^*$, $\mathcal{B}^*$ and $\mathcal{W}^*$ where $\mathcal{K}^* = \mathcal{K}^* \cup \mathcal{K}_j(\alpha, \beta, \gamma)$ and $\mathcal{W}^* = \mathcal{W}^* \cup \{ \mathcal{W}_j \}$.

1. $Q \leftarrow \{ (w_i^t, B_i^t) : w_i \in \mathcal{W}, B_i \in \mathcal{B} \}$  
2. **while** $\exists (w_i^t, B_i^t) \in Q : \Xi_i^t \neq \Xi_i^t$ in parametric space **do** 
   3. **forall** the $(w_i^t, B_i^t) \in Q$ **do** 
      4. obtain knot vectors $\Xi_i^t$ from $T^*$ 
      5. **if** $\Xi_i^t = \Xi_i^t$ in parametric space **then** 
         6. $\Xi_i^t \leftarrow \Xi_i^t$ 
      7. **else** if $\Xi_i^t$ is more refined than $\Xi_i^t$ **then** 
         8. add an anchor, insert a knot of $\Xi_j^t$ into $\Xi_i^t$ and do the refinement: $B_i^t = c_1 B_j^t + c_2 B_i^t$ (Section 3.2) 
         9. $w_j^t \leftarrow w_i^t \cdot c_1$; $w_j^t \leftarrow w_i^t \cdot c_2$ 
         10. $Q \leftarrow Q \setminus \{ (w_i^t, B_i^t) \} \cup \{ (w_j^t, B_j^t), (w_i^t, B_i^t) \}$ 
      11. **else** if $\Xi_i^t$ indicates an anchor $k_j(\alpha, \beta, \gamma) \notin \mathcal{K}^*$ **then** 
         12. $\mathcal{K}^* \leftarrow \mathcal{K}^* \cup k_j(\alpha, \beta, \gamma)$ 
         13. **if** $k_j(0,0,0) \notin \mathcal{K}^*$ **then** 
         14. $\mathcal{K}^* \leftarrow \mathcal{K}^* \cup \{ k_j(0,0,0) \}$ 
         15. $\mathcal{V}^* \leftarrow \mathcal{V}^* \cup \{ \mathcal{V}_j \}$ // Insert a new vertex 
      16. **end if** 
   17. **end if** 
   18. **end forall** 
   19. **forall** the $c \in \mathcal{C}^*$ **do** 
      20. **if** vertices on the edges of $c$ form an axis-aligned plane that splits $c$ into $c_1$ and $c_2$ **then** 
         21. $\mathcal{C}^* \leftarrow \mathcal{C}^* \setminus \{ c \} \cup \{ c_1, c_2 \}$ // divide $c$ into $c_1$ and $c_2$ 
      22. **end if** 
   23. **end forall** 
   24. **end while** 
25. $\mathcal{B}^* \leftarrow \{ B_i : (w_i^t, B_i^t) \in Q \}$ 
26. $\mathcal{W}^* \leftarrow \{ w_j = \sum_{(w_i^t, B_i^t) \in Q} w_i^t \}$

Note that any blending function introduced by Algorithm 1 must center at a certain anchor, but not vice versa, i.e., there could be anchors not associated with any blending functions. Moreover, the new T-spline after refinement is still semi-standard, because the denominators in Equation 6 remain unchanged in Algorithm 1, due to

$$w_i^t B_i^t = w_i^t \cdot c_1 B_j^t + w_i^t \cdot c_2 B_i^t \equiv w_i^t B_j^t + w_i^t \tilde{B}_i^t$$

### 4.3.2 Modifying Blending Functions with Anchor Insertions

Fig. 6. Examples of eliminating violations against rules (i)(ii) in the case of cubic B-spline basis functions. Suppose $x = 0$ is the boundary and the ghost region covering $(0, \infty)$ in (a), $N_0$ represents an internal basis against (ii). After two extra knots 0 are inserted, $N_0$ is refined to $N_1$ and $N_2$ which comply with rule (ii); In (b) where the ghost region covers $(-\infty, 0]$, ghost basis $N_0$ violates the rule (i). After refinement with insertion of duplicate knots, it is replaced by $N_1$ and $N_2$ in ghost blending functions. Thus, no violation against (i) exists.

Anchor insertion operation is our fundamental tool to modify existing blending functions of trivariate T-spline in order to get rid of all violations against rules (i), (ii) and (iii). As blending functions of trivariate T-spline are tensor-product of three univariate cubic B-spline bases, let’s execute this method in 1D by using two examples given in Figure 6. In Figure 6(a), $N_0 = N[-2, -1, 0, 1, 2]$ represents an internal basis which apparently violates rule (ii). If two extra knots 0s are inserted, $N_0$ is refined into two internal bases $N_1 = N[-2, -1, 0, 0, 1]$ and $N_2 = N[-1, 0, 0, 1, 2]$, such that $N_0 = \frac{2}{3} N_1 + \frac{1}{3} N_2$, $N_1 = \frac{2}{3} N_1 + \frac{1}{3} N_2$ according to the refinement algorithm in Section 3.2. Now once the ghost region is gone, $N_1$ and $N_2$ change to $N_1^* = N[-2, -1, 0, 0, 0]$ and $N_2^* = N[-1, 0, 0, 0, 0]$ respectively and we still have $N_0(u) = \frac{2}{3} N_1^*(u) + \frac{2}{3} N_2^*(u)$ on $u \in [-2, 0]$, as shown in the bottom of Figure 6(a). Therefore, the violation against (ii) is successfully eliminated. Figure 6(b) depicts a scenario where $N_0 = N[-3, -2, -1, 0, 1]$ in violation of rule (i) overlaps with the domain region at $[0, 1]$. By inserting two duplicate knots at 0, we may replace $N_0$ with two resulting ghost bases $N_1$ and $N_2$, both of which abide with the rule (i). For the case of trivariate T-spline, knot insertions are
replaced by anchor insertions conducted on T-lattice, and a much more complex refine algorithm (see Section 4.3.1) is employed instead.

### 4.3.3 Anchor Insertions on Polycube Boundary

It’s easy to see that the blending functions associated with those master anchors either on or adjacent to the interfaces between $P$ and $G$ are in violation of rule (ii). Therefore, we need to insert sub-anchors to boundary vertices. The basic idea is analogous to that in Section 4.2 where sub-anchors are added on the surface of a box domain to ensure its restricted boundary. However, a variety of corner types may be found on polycube surfaces (see Figure 5(b)), thus we have to handle all of them for proper anchor insertions. To exhaust all possible corner types, then choose sub-anchors to insert is tedious and inefficient. Instead, we developed a general algorithm to determine which sub-anchors to be inserted at arbitrary boundary vertex. Given a boundary vertex $v_i$, we first add the master anchor to it, along with all the sub-anchors that lie within the domain of $T$ in construction space. Then the sub-anchors lying within the domain of $P$ in construction space are colored red, and the others are blue. If there exists $k_i(-\alpha,\beta,\gamma) \in K$ for all $k_i(-\alpha,\beta,\gamma) \in K$ and $\text{color}(k_i(-\alpha,\beta,\gamma)) = \text{color}(k_i(\alpha,\beta,\gamma))$ for $\alpha \in \{-1, 1\}, \beta, \gamma \in \{-1, 0, 1\}$, we delete $\{k_i(\pm 1,\beta,\gamma)\}$ from $K$, that is, the sub-anchors on the 1st and the 3rd layers in $0$-axis direction of the $3 \times 3 \times 3$ grid at $v_i$ match in color pattern, they are deleted from $K$. Then this operation is performed similarly in the other directions. The intuition of this method is to generate $C^0$ continuities at the boundaries with as few sub-anchors as possible, in order to keep the smoothness along the other directions. An example is given in Figure 7 in which sub-anchors are inserted at a boundary vertex on a 2D mesh. After all the required sub-anchors added at the interface between $P$ and $G$, Algorithm 1 is then applied to generate a new set of blending functions and a new set of weights.

![Figure 7](image)

Fig. 7. Inserting sub-anchors to a boundary vertex. Red dots denote anchors inside $P$ and blue ones are those in $G$. As the color patterns on the leftmost and rightmost grid layer match, all sub-anchors on both layers are removed.

### 4.3.4 Other Anchor Insertions

Section 4.3.3 has resolved most violations against rules (i) and (ii) arising from the blending functions that are associated with the master anchors close to the polycube boundary. Nevertheless, there are still other violations left. They can be categorized into four types as follows:

1. (see Figure 8(a)) Ghost blending functions associated with sub-anchors violate rule (i). For example, the support region of the blending function associated with $k_{1,0}$ (the other index is omitted for conciseness reason) overlaps with $P$. A pair of anchors $k_{1,0}$ and $k_{0,1}$ can be added to reduce the support region to the boundary while no further violations being introduced. The violation arising from $k_{1,0}$ is treated in the same fashion. In the case of $k_{1,1}$, only 1 sub-anchor $k_{1,0}$ is required to eliminate the violation.

2. (see Figure 8(b)) Internal blending functions associated with sub-anchors violate rule (ii). For example, removal of the ghost region and ghost anchors will cause the changes in the shape of the blending function associated with $k_{1,0}$ because its knot vector goes into the ghost region. Similar to case 1, $k_{0,1}$ and $k_{1,0}$ can be added to cut off the blending function from outside. Only one anchor insertion is necessary to resolve the violation arising from $k_{1,0}$. Even though the new blending functions after refinement still covers nearby ghost region, it doesn’t violate rule (ii) anymore. This has been explained in Section 4.3.2 (Figure 6(a)).

3. (see Figure 8(c)) Ghost blending functions near a convex corner of $P$ violate rule (i). For example, in spite of their knot vectors being apart from $P$ and any internal anchors, the blending functions associated with certain ghost anchors ($e.g., k_{1,1}$, $k_{1,0}$ and etc.) still influence the corner region of $P$. In order to resolve this kind of violation, sub-anchors are added to the surface extensions of the convex corner, separating the blending function from $P$ by shrinking its support region.

4. (see Figure 8(d)) Internal blending functions near a concave corner of $P$ violate rule (iii). This case is exactly the same as case 3 except that the domain region and the ghost regions are interchanged. The purpose of eliminating this kind of violations is to ensure restricted boundary of $P$.

Once new sub-anchors are inserted for all violations, we apply the refinement algorithm given in Section 4.3.1 and obtain new sets of blending functions, weights and anchors along with the updated T-lattice structures. Since extra anchors may be introduced by the refinement, we have to search for new violations and resolve them again. These two steps are repeated until no violation is found. We notice in our experiment that it only takes one or two iterations in practice to eliminate all violation cases. On the other hand, the proposed anchor insertion method is guaranteed to terminate due to the fact that no vertex is added during refinement, and there are only a finite number of sub-anchors can be added to $T$. In the worst case, each cube of $T$ turns into a small Bézier volume.

### 4.4 Generating RTP-spline Function

By removing $G$ and all ghost anchors from $K$, we obtain RTP-spline, a single-piece smooth function, defined over
a polycube domain \( P \). Our anchor insertion method guarantees the resulting RTP-splines have a restricted boundary. Furthermore, the refinement algorithm proposed in Section 4.3.1 ensures semi-standardness of the obtained RTP-splines from the original B-spline volume. Since the denominator remains 1 over the entire domain \( P \), we can rewrite Equation 6 in a simpler formulation:

\[
F(u, v, w) = \sum_{i=1}^{\text{B}} w_i p_i B_i(u, v, w) \quad (u, v, w) \in \mathbb{R}^3 \tag{7}
\]

5 Modeling Solid Objects

It is a challenging task to build single-piece and smooth spline representations for arbitrary solid objects, especially for those with bifurcations and high genus. This section addresses how to construct a RTP-spline for a given solid model. In this work, an input solid model is represented as a dense tetrahedral mesh \( M = \{V, T\} \). Its geometry and other material attributes are discretely represented on vertices \( V \), and are interpolated linearly within each tetrahedron of \( T \). Note that our volumetric mapping algorithm is a meshless method with a closed-form mapping representation, and it works for other volumetric data representations such as point clouds and voxel grids. Therefore, the entire RTP spline construction pipeline can be easily generalized to handle other volumetric data input formats.

We first construct a polycube \( P \) following the geometry and topology of \( M \) and compute a volumetric mapping \( f : P \rightarrow M, P, M \subset \mathbb{R}^3 \). The polycube \( P \) can be constructed either manually \([26],[27],[34]\) or automatically \([6],[14]\). These techniques also provide the boundary mapping \( f' \) from the polycube boundary surface (denoted as \( \partial P \)) to the boundary of \( M (\partial M) \). We use such a surface mapping \( f' : \partial P \rightarrow \partial M \) as the boundary condition of \( f \). The volumetric parameterization is then defined as the seeking of a harmonic energy minimizer:

\[
\begin{align*}
\Delta f(x) &= 0 & x \in P, \\
f(x) &= f'(x) & x \in \partial P.
\end{align*}
\]

where \( \Delta \) is the 3-dimensional Laplace operator, defined for each real function \( f \) in \( \mathbb{R}^3 \) as

\[
\Delta f = \nabla \cdot \nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}.
\]

\( \Delta f = 0 \) for \( f = (f^1, f^2, f^3) \) is equivalent to \( \Delta f^i = 0 \) in all the \( i = 1, 2, 3 \) coordinate directions. We compute the volumetric polycube mapping using the method of fundamental solutions (MFS) \([11],[12]\). We recap the basic algorithm here and refer more details to \([12]\).

Based on the maximum principal of harmonic functions, critical points of harmonic functions exist only on the boundary. Furthermore, function values in the interior region of \( P \) are fully determined by the boundary values \( f(x), x \in \partial P \) and can be computed by Green’s functions. Specifically, the real harmonic function value \( f(x) \) can be computed as the integration of its boundary values and the kernel function (i.e. fundamental solutions associated with the 3D Laplacian operator \( \Delta \)). The kernel function of \( \Delta \) has the following formula:

\[
K(x, x') = \frac{1}{4\pi} \frac{1}{|x - x'|},
\]

which matches the electrostatics. In other words, solving a harmonic function can be converted to designing a specific electric field determined by an electronic particle system, whose electric potential mimics \( f \) and shall satisfy the boundary condition \( f' \) on \( \partial P \).

The computation pipeline is to first place a set of charge points \( \{q_s\} \) outside the domain \( q_s \in \partial P, P \subset \bar{P} \subset \mathbb{R}^3 \).
Then we conduct a boundary fitting which solves the charge distribution \( \{ w_i \} \) on these points \( \{ q_i \} \). The harmonic function \( f(x) \) is represented using the MFS equation:

\[
f(x, W, Q) = \sum_{s=1}^{N_s} w_s \cdot K(x, q_s), x \in P, q_s \in \partial P,
\]

where \( f \) is guaranteed to be harmonic, and we only need to enforce the boundary condition on \( \partial P \). For the boundary fitting, we sample \( N_c \) collocation points on the domain boundary \( \partial P \) to set up the constraint equations. If we have \( N_x \) charge points and \( N_c \) collocation points, for a real harmonic function \( f \) (e.g. on an individual axis direction) we only need to solve an \( Ax = b \) linear system where \( A \) is an \( N_c \times N_x \) matrix. The system can be efficiently solved by truncated Singular Value Decomposition [12], [13].

Parametrization of a general solid model on its adaptive polycube domain can get lower distortion than that on a single box domain, since the polycube can be constructed to have the same topology and similar geometry as the model. Actually, in RTP-spline construction, parameterization without fully conformality and equivalence-property does not bring too much trouble to the volume fitting, as long as the overall parameterization mapping is continuous and smooth. Therefore, the current parameterization is efficient and sufficient, i.e. the shape (angle) distortion and volume distortion of our volumetric mapping are satisfactory.

Along two directions, we will also explore volumetric mapping techniques for parameterization with higher quality: (1) we can use more complicated/general parametric domains such as manifold domains (directly represented by tetrahedral meshes), polytubes [34], and so forth, which may more flexibly approximate the shape and yield lower distortion. However, on such domains it becomes more challenging to construct regular splines providing same favorable features of RTP-splines. (2) the current volumetric mapping is fully determined by the boundary constraint, i.e., the polycube surface mapping [26], can we reduce the distortion by conducting relaxation of boundary surface mapping [10], now driven by the volumetric mapping distortion. However, this makes the mapping computation a nonlinear optimization and inefficient.

### 5.2 RTP-spline Volume Fitting

Given \( f : P \to M \), we evenly select a group of points \( U = \{ u_1, u_2, \ldots, u_m \} \) from the polycube parametric domain \( P \), hence their counterparts in the real world domain are \( X = \{ x_i = f(u_i), i = 1, \ldots, m \} \). The problem of fitting RTP-spline \( F(u, v, w) \) resorts to minimizing the following equation using \( U \) and \( X \), with respect to control points \( p_i \):

\[
\sum_{i=1}^{m} (F(u_i) - x_i)^2
\]

Alternatively, it can be represented in format of

\[
\frac{1}{2} P^T B P - X^T B P
\]

in which \( P_j = p_j^T \), \( X_i = x_i^T \), and \( B_{ij} = I_{3 \times 3} B_i(u_j) \). This is a typical least square problem, and we solve it for \( P \) using the optimization package MOSEK([11]).

If the fitting results don’t meet the requirement, we can improve them by refitting after adaptively subdividing cells where large fitting errors occur. Each cell from \( P \) can be split into two, four or eight smaller ones, depending on its aspect ratio. Once vertices, faces, and cells are added, Algorithm 1 is employed to refine existing RTP-spline and introduce additional degree-of-freedom for better fitting. Note that Algorithm 1 is originally devised to work on a box domain, it can be however straightforwardly applied to RTP-splines defined on polycube domains, with a minor revision. That is, whenever a new boundary vertex is added, we have to insert a few sub-anchors in addition to the master anchor by following the way described in Section 4.3.3, in order to preserve the restricted boundary on the resulting RTP-splines.

Comparing to the number of degree-of-freedoms (DOFs) in the optimization problem (Equation 8), \( U \) normally contains a much greater number of parametric points evenly distributed inside the polycube domain. So the optimization problem is well-posed and the resulting linear equations form a nondegenerate system. If there are too many subdivisions, the increased number of DOFs may undermine the nodegeneracy of the system. In this case, we will enlarge \( U \) by adding more points from the parametric polycube domain near where where subdivisions take place and recalculate \( X \).

### 6 RESULTS AND DISCUSSION

#### Table 2

Statistics of Volume Fitting.

<table>
<thead>
<tr>
<th>Models</th>
<th>Data Points</th>
<th># Control Points</th>
<th>RMS Error ( \times 10^{-3} )</th>
<th>Timing (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bimba</td>
<td>35511</td>
<td>4543</td>
<td>1.20</td>
<td>31.21</td>
</tr>
<tr>
<td>Kitten</td>
<td>60144</td>
<td>3820</td>
<td>1.27</td>
<td>44.53</td>
</tr>
<tr>
<td>2-Torus</td>
<td>26384</td>
<td>2888</td>
<td>3.69</td>
<td>20.65</td>
</tr>
<tr>
<td>Hand</td>
<td>1502700</td>
<td>9035</td>
<td>0.584</td>
<td>1150</td>
</tr>
<tr>
<td>Head</td>
<td>472122</td>
<td>12880</td>
<td>0.291</td>
<td>422.4</td>
</tr>
<tr>
<td>Beethoven</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1st level)</td>
<td>103361</td>
<td>1001</td>
<td>1.80</td>
<td>67.79</td>
</tr>
<tr>
<td>(2nd level)</td>
<td>103361</td>
<td>3283</td>
<td>1.34</td>
<td>80.78</td>
</tr>
<tr>
<td>(3rd level)</td>
<td>103361</td>
<td>14699</td>
<td>0.718</td>
<td>123.28</td>
</tr>
</tbody>
</table>

A system consisting of volumetric parametrization, RTP-spline construction and data fitting is implemented in C++ and the experiments are carried out on a 3GHz Pentium-IV PC with 4G RAM. Our experimental data include solid models of Bimba, Beethoven, eight (genus 2), kitten (genus 1), hand (5 bifurcations) and head (with brain excavated), which are represented as tetrahedral meshes. We successfully convert them into representations of single-pieceed smooth RTP-splines by using the method proposed in this paper. The experimental results are given in Fig. 10.

The step of RTP-spline construction is efficient and usually takes only a few seconds, which consists of deducing knot vectors, building blending functions, calculating...
weights and initializing necessary data structures. In all our experiments, this step takes at most 6 seconds (for the Beethoven model at level 3). In contrast, fitting RTP-splines to volumetric dataset is more computationally expensive. The statistics of volumetric fitting are documented in Table 2, where the data points are parameterized on polycube domains, the fitting qualities are measure by RMS errors, and the fitting errors are normalized to the overall sizes of solid models. From Table 2, we find that the volumetric fitting of RTP-splines can be finished efficiently and yield reasonable results. In addition, RTP-splines enable local subdivision of cells over desired regions to improve fitting qualities. As shown in the Beethoven model: the initial error is $1.80 \times 10^{-3}$ without subdivisions and is reduced to $7.18 \times 10^{-4}$ after two levels of subdivisions. The geometric details of the Beethoven model are also gradually revealed with the increasing level of subdivision (see Figure 11).

RTP-spline is semi-standard and hence computing blending functions and their derivatives on it is much more efficient than on traditional T-splines. To prove this, we compared the computational cost on the models Bimba, Kitten and two-hole torus in both kinds of spline representations. To ensure the fairness in the comparisons, we use the same source codes of RTP-splines to compute blending functions and derivatives for traditional T-splines, by including calculation of denominators. The comparison results given in Table 3. As a result, the costs of the calculations of $B$, $B'$, and $B''$ using traditional T-splines are roughly reduced by 47%, 46%, and 58% respectively if RTP-splines are used instead.

We can model other attributes in addition to geometry in RTP-splines by increasing the vector sizes of control points. In one of our experiments, we synthesize a scalar field on the head model, and then successfully recover a single RTP-spline representation of both the geometry and scalar values as shown in Figure 12. Two kinds of scalar fields are involved in the experiment. One is the distance field to both the head surface and the brain surface inside (see Figure 12(d)). The fitting result for the distance field and the corresponding fitting error map are demonstrated in Figure 12(e)(f) respectively. Note that the fitting errors shown here are also normalized RMS errors as the distances are related to the model geometry. The other type of scalar field is a synthesized procedure 3D texture, generated using the fractal sum of Perlin noise [19] as $T(p) = \sum_{i=1}^{4} \frac{1}{i} noise(i p), p \in R^{3}$ (see Figure 12(g)).

As RTP-spline function is continuously and smoothly defined over a polycube, we can evaluate any properties that depends on function values and derivatives anywhere over the domain. If we interpret the RTP-spline function $F$ obtained from data fitting as a deformation from a polycube to the shape of the fitted solid model, the deformation gradient tensor is $G = F \otimes \nabla$ and its Jacobian $det(G)$ measures the volume changes produced by the deformation. In Figure 9, the Jacobian values for the hand and Bimba model are directly evaluated from function value and derivatives of $F$ and there are no abrupt jumps in color due to the smoothness and continuity of RTP-splines.

**Linear Independence of Blending Functions.** In this paper, we use the term RTP-spline blending functions instead of basis functions because whether they are linearly independent is not clear. It has been proven in [3] that linear independence is not guaranteed for general T-splines. It’s also pointed out that [3] linear independence can be inherited from a coarse T-mesh after a sequence of refinement only if in each step we insert anchors following certain constraints. However, a similar conclusion has not been revealed on T-lattices for trivariate T-splines, so the linear independence property of RTP-splines remains open.

Note that, presence of linear dependence does not deteriorate the results of RTP-spline volumetric fitting which is posed as a linear least square problem. Intuitively, if the refinement in Section 4.3.1 introduces no sub-anchors

<table>
<thead>
<tr>
<th>Model</th>
<th>Sample Points</th>
<th>Polycube Spline</th>
<th>General T-spline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$B(u, v, w)$</td>
<td>$B'(u, v, w)$</td>
</tr>
<tr>
<td>Bimba</td>
<td>2512</td>
<td>0.18s</td>
<td>0.6s</td>
</tr>
<tr>
<td>Kitten</td>
<td>23076</td>
<td>1.61s</td>
<td>5.21s</td>
</tr>
<tr>
<td>2-Torus</td>
<td>9768</td>
<td>0.71s</td>
<td>2.42s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$B''(u, v, w)$</td>
<td>$B'(u, v, w)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$B''(u, v, w)$</td>
<td>$B''(u, v, w)$</td>
</tr>
</tbody>
</table>

The statistics of volumetric fitting are documented in Table 2, where the data points are parameterized on polycube domains, the fitting qualities are measure by RMS errors, and the fitting errors are normalized to the overall sizes of solid models. From Table 2, we find that the volumetric fitting of RTP-splines can be finished efficiently and yield reasonable results. In addition, RTP-splines enable local subdivision of cells over desired regions to improve fitting qualities. As shown in the Beethoven model: the initial error is $1.80 \times 10^{-3}$ without subdivisions and is reduced to $7.18 \times 10^{-4}$ after two levels of subdivisions. The geometric details of the Beethoven model are also gradually revealed with the increasing level of subdivision (see Figure 11).

RTP-spline is semi-standard and hence computing blending functions and their derivatives on it is much more efficient than on traditional T-splines. To prove this, we compared the computational cost on the models Bimba, Kitten and two-hole torus in both kinds of spline representations. To ensure the fairness in the comparisons, we use the same source codes of RTP-splines to compute blending functions and derivatives for traditional T-splines, by including calculation of denominators. The comparison results given in Table 3. As a result, the costs of the calculations of $B$, $B'$, and $B''$ using traditional T-splines are roughly reduced by 47%, 46%, and 58% respectively if RTP-splines are used instead.

We can model other attributes in addition to geometry in RTP-splines by increasing the vector sizes of control points. In one of our experiments, we synthesize a scalar field on the head model, and then successfully recover a single RTP-spline representation of both the geometry and scalar values as shown in Figure 12. Two kinds of scalar fields are involved in the experiment. One is the distance field to both the head surface and the brain surface inside (see Figure 12(d)). The fitting result for the distance field and the corresponding fitting error map are demonstrated in Figure 12(e)(f) respectively. Note that the fitting errors shown here are also normalized RMS errors as the distances are related to the model geometry. The other type of scalar field is a synthesized procedure 3D texture, generated using the fractal sum of Perlin noise [19] as $T(p) = \sum_{i=1}^{4} \frac{1}{i} noise(i p), p \in R^{3}$ (see Figure 12(g)).

As RTP-spline function is continuously and smoothly defined over a polycube, we can evaluate any properties that depends on function values and derivatives anywhere over the domain. If we interpret the RTP-spline function $F$ obtained from data fitting as a deformation from a polycube to the shape of the fitted solid model, the deformation gradient tensor is $G = F \otimes \nabla$ and its Jacobian $det(G)$ measures the volume changes produced by the deformation. In Figure 9, the Jacobian values for the hand and Bimba model are directly evaluated from function value and derivatives of $F$ and there are no abrupt jumps in color due to the smoothness and continuity of RTP-splines.

**Linear Independence of Blending Functions.** In this paper, we use the term RTP-spline blending functions instead of basis functions because whether they are linearly independent is not clear. It has been proven in [3] that linear independence is not guaranteed for general T-splines. It’s also pointed out that [3] linear independence can be inherited from a coarse T-mesh after a sequence of refinement only if in each step we insert anchors following certain constraints. However, a similar conclusion has not been revealed on T-lattices for trivariate T-splines, so the linear independence property of RTP-splines remains open.

Note that, presence of linear dependence does not deteriorate the results of RTP-spline volumetric fitting which is posed as a linear least square problem. Intuitively, if the refinement in Section 4.3.1 introduces no sub-anchors
in addition to those selectively inserted in Section 4.3.3 and Section 4.3.4, degeneracy will not happen. It is still desirable to have a thorough understanding on linear independence property of RTP-splines, and/or to have an adaptive modification scheme on RTP-splines construction to ensure it; because linear independence is required for certain applications such as isogeometric analysis.

7 Conclusion

In this paper we have proposed the concept and construction algorithm of RTP-splines and presented an effective framework to transform volumetric data (both geometries and associated attributes of solid objects) into representation of RTP-splines. Because of the topological flexibility of the polycube domain, RTP-splines can naturally model solid objects with bifurcations and high genus, while ensuring lower parametrization distortion in comparison with traditional splines defined over standard box domains. Our algorithm guarantees that the initially-constructed RTP-splines are semi-standard, so that it enables the efficient computation of spline functions and their derivatives, without any division overhead. The proposed RTP-spline supports local refinement, and a refinement algorithm has been developed to preserve the semi-standardness on the RTP-splines undergoing anchor insertion and local subdivision. The particular restricted boundary requirement of RTP-splines prevents control points from affecting domain regions spanning across nearby boundaries.

We demonstrate the efficacy of our RTP-splines as a powerful solid modeling tool in various experiments. This unified paradigm enables the transformation from discrete solid models (represented by tetrahedral meshes) into continuous RTP-spline representations, accurately modeling both geometry and possibly multi-dimensional attributes. One unclear property of the RTP-spline is its linear independence, we will explore the constraints during the RTP-spline construction that ensures it. When the linear independence problem is solved, we would also like to explore the isogeometric analysis founded upon RTP-splines. Moreover, the particular polycube domains of RTP-splines can be naturally decomposed into a set of regular structures, which will enable GPU-friendly computing and image-based geometric shape processing.

References

Fig. 10. From left to right, original solid models represented by tetrahedral meshes, polycube domains of RTP-splines, and hexahedral meshes rendered from RTP-spline functions. (The edges of the hand tetrahedral model are omitted on purpose due to their extraordinary numbers.)
Fig. 11. From top to bottom: fitting results of the Beethoven model with 0, 1 and 2 levels of subdivisions.
Fig. 12. Fitting results for the head model associated with synthesized scalar field (red denotes high value while blue denotes low value). (a) polycube in parametric domain, (b,c) are the volumetric meshes reproduced from fitted RTP-splines, (d) synthesized distance field texture, (e) texture generated from the fitted RTP-spline, (f) fitting error map, where the maximum error is $0.92 \times 10^{-2}$ and the average is $6.0 \times 10^{-4}$; (g) texture synthesized as a Perlin noise function. (h,i) show the fitting errors and the error map respectively. The maximum fitting error for noise texture is $0.066$ and the average is $7.3 \times 10^{-4}$. 