Restricted Trivariate Polycube Splines for Volumetric Data Modeling

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

Abstract—This paper presents a 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; it uses solid polycube structure as underlying parametric domains and strictly bounds blending functions within such domains. We construct volumetric RTP-splines in a top-down fashion in four 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 by adding anchor points and performing local refinement; and (4) Removing exterior cells and anchors. Besides local refinement inherited from general \( T \)-splines, the RTP-splines have a few attractive properties as follows. (a) They naturally model solid objects with complicated topologies/bifurcations using a one-piece continuous representation without domain trimming/patching/merging. (b) They have guaranteed semi-standardness \([1]\) so that the functions and derivatives evaluation is very efficient. (c) Their restricted support regions of blending functions prevent control points from influencing other nearby domain regions that stay opposite to the immediate boundaries. These features are highly 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 datasets with multi-attributes 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. Smooth modeling of solids is critical for modern engineering design that needs to directly and efficiently apply physical simulation on volumetric regions, without the expensive procedures of remeshing finite-element structure, and converting from discrete representations to continuous ones and from linear finite elements to higher-degree piecewise splines.

In practice, many real-world objects have complex geometries and non-trivial topologies. Constructing efficient representations for general solid objects in favor of physical simulation and engineering design is therefore highly challenging. Trivariate simplex splines \([2]\) 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 much less efficient compared with NURBS and tensor-product \( B \)-splines. Also, how to place boundary knots to avoid numerical degeneracies remains open. Trivariate simplex splines are defined over an unstructured tetrahedral mesh, which can be generated from boundary triangular meshes (e.g. using \([3]\)). Although solid object of complex topologies and geometries can be modeled by trivariate simplex splines upon such unstructured structure, the majorities of simulation solvers have preferences on regular grids. Because low-quality tetrahedral meshes usually cause large simulation errors and numerical instability. Motivated by current industrial practice in engineering design and analysis systems, we focus on designing a volumetric spline modeling framework based on structured grid domains. Tensor-product splines have potentials to become an ideal representation scheme for this purpose.

In the framework of isogeometric analysis \([4], [5]\), trivariate tensor-product \( B \)-splines/NURBS are directly used to model smooth geometry and material attributes of solid objects for physical simulation. Martin et al. \([6]\) convert solid models to cylindrical trivariate \( B \)-splines by parameterizing models on solid cylinders. Due to the topological limitation of the cylindrical domain, the constructed trivariate tensor-product splines can not model solid objects with bifurcations and arbitrary topologies, without enormous patch gluing/trimming efforts and imposing smoothness constraints along patch boundaries. Furthermore, tensor-product splines do not support local refinement and level-of-detail modeling because refining their basis functions will introduce many superfluous control points across the entire domain. As an extension of NURBS, T-splines \([1], [7]\) solve this problem on semi-regular grid domains. To the best of our knowledge, T-splines have not been generalized for three dimensional, multi-attribute volumetric geometry and data modeling.

Generalization of T-splines from surface to volumetric data is nontrivial. A general T-spline function defined over
a bivariate domain can be formulated as

$$F(u, v) = \sum_{i=1}^{n} w_i p_i B_i(u, v)$$

where \( p_i \) are control points associated with weights \( w_i \), and \( B_i(u, v) \) denotes the basis function. Two pieces of T-spline patches can be stitched together by blending boundary basis functions, and form a new T-spline that preserves 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 dividing blending functions by the sum of all the contributed ones. This will make the evaluation computationally inefficient. On the other hand, Semi-standard T-splines introduced recently in [1] guarantee \( \sum_{i=1}^{n} w_i B_i(u, v) = 1 \) in Eq. (1) across the entire domain. In this setting, the computation of \( F(u, v) \) and its derivatives can be much more efficient.

$$\sum_{i=1}^{n} \frac{w_i p_i B_i(u, v)}{\sum_{i=1}^{n} w_i B_i(u, v)} \quad (u, v) \in \mathbb{R}^2,$$

where \( p_i \) are control points associated with weights \( w_i \), and \( B_i(u, v) \) denotes the basis function. Two pieces of T-spline patches can be stitched together by blending boundary basis functions, and form a new T-spline that preserves 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 dividing blending functions by the sum of all the contributed ones. This will make the evaluation computationally inefficient.

![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, constructing a semi-standard T-spline is challenging over non-trivial parametric domains. Also, conventional T-splines are defined with floating boundaries, i.e., the support regions of blending functions may go beyond the domain boundaries. Such a floating-boundary scheme upon a polycube domain will cause control points to unnecessarily contribute to extra domain regions. Two examples are illustrated as red-regions in Fig. 1. This might cause geometric inconsistencies in modeling underlying solid objects and in physical simulations. Therefore, it would be 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 splines will greatly facilitate direct modeling and physical simulation 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).

This paper presents a framework of RTP-spline construction and the data conversion of volumetric models to this spline representation. It has the following major contributions:

- We formulate a new spline (RTP-spline) scheme over a polycube domain, restricting blending functions inside the domain boundaries. The RTP-splines also have many advantages: (1) It is capable of local refinement; (2) Its function and derivative evaluations are much more efficient than that of traditional T-spline surfaces; (3) Its polycube domain enables natural modeling of arbitrary solid objects, and the domain shape mimics the geometry and topology of the model and introduces low parameterization distortion and few singularity points; (4) Its restricted boundaries ensure the physical modeling and simulation adhere to geometry of underlying objects.

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

- We construct RTP-splines on several volumetric models with both geometry and synthesized textures (to mimic material properties), which demonstrates that our RTP-splines can model not only geometry but also multi-attribute fields within a 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 [8]. Martin and Cohen [9] model physical attributes across a trivariate NURBS volume. It is more desirable in engineering design to have an integrated modeling framework that can represent geometry and material and conduct simulations simultaneously. Trivariate NURBS are used to model skeletal muscle with anisotropic attributes [4], on which NURBS-FEM analysis is directly conducted. Martin et al. [6] parameterize a volumetric solid into a solid cylinder upon which a trivariate B-spline is constructed. Hua et al. develop a framework based on triangular simplex splines [2] to model and render multi-dimensional material attributes of solid objects with complicated geometries and topologies.

The splines proposed in this paper are founded upon the T-spline technique [7]. The 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 [1], [10] to simplify NURBS surfaces to T-spline representations by removing superfluous control points. [11] thoroughly discusses the merge of B-spline patches defined over different local domains to get a single T-spline representation on the manifold domain.

Bazilevs et al. [12] propose an isogeometric analysis framework based on T-splines. Its main focus is on planar...
T-splines for surfaces, and volumetric T-splines are only briefly mentioned without offering any technical details. Generalized trivariate T-splines (whose control points are associated with weights) are employed by [13] to model free-form deformation fields. For the purpose of shape metamorphosis, 3D level sets represented by T-splines are adopted in [14]–[17] 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 the construction and parameterization of a polycube domain. The parameterization on polycubes originated for seamless texture mapping with low distortion [18]. Polycubes serve as nice parametric domains because they approximate well the geometry of the model and possess great regularity. A polycube mapping can be constructed either manually [18]–[20] or automatically [21], [22].

Let \( \mathbf{v}_i \) be a point in the parametric space. Given \( \mathbf{v}_i = (v_i^0, v_i^1, v_i^2) \), all the corresponding anchors \( \mathbf{k}_{i(\alpha, \beta, \gamma)} \) share the same coordinate \( (v_i^0, v_i^1, v_i^2) \) in parametric space. To distinguish these anchors for T-spline construction, we define \( \mathbf{k}_{i(\alpha, \beta, \gamma)} = \mathbf{v}_i + (\alpha, \beta, \gamma)\epsilon \) as the coordinate of \( \mathbf{v}_{i(\alpha, \beta, \gamma)} \) 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 \( \mathbf{k}_j \), where \( j \) indicates the index of \( \mathbf{k}_{j(\alpha, \beta, \gamma)} \) in \( \mathcal{K} \).

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

\[
\mathbf{F}(u, v, w) = \sum_{i=1}^{\lvert \mathcal{B} \rvert} p_i B_i(u, v, w) \quad (u, v, w) \in \mathbb{R}^3,
\]

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

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

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 \( N_{ij}^3 \) is constructed upon the knot vector \( \Xi_i^j \), which is deduced from \( \mathbf{T} \) and a collection of anchors \( \mathcal{K} \).

Let \( \mathcal{T}(\mathcal{V}, \mathcal{C}, \mathcal{F}) \) 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. \( \mathcal{K} \subset \mathcal{V} \times \{-1, 0, +1\}^3 \) denote a set of anchors attached to the vertices. At most 27 anchors are allowed at each vertex, and they can be visually imagined to be organized on a \( 3 \times 3 \times 3 \) grid of infinitesimal size, as shown in Fig. 2. We require that each vertex has a master anchor at the center of the local grid, while the others are optional and called sub anchors. In the rest of the paper, we denote an anchor at \( \mathbf{v}_i \) as \( \mathbf{k}_{i(\alpha, \beta, \gamma)} \), in which the triplet \( (\alpha, \beta, \gamma) \) indicates a unique nodal position on the local grid. Given \( \mathbf{v}_i = (v_i^0, v_i^1, v_i^2) \), all the corresponding anchors \( \mathbf{k}_{i(\alpha, \beta, \gamma)} \) share the same coordinate \( (v_i^0, v_i^1, v_i^2) \) in parametric space.

Fig. 2. A vertex \( \mathbf{v}_i \) can have at most 27 anchors organized on a \( 3 \times 3 \times 3 \) virtual grid. The central one (red) is the master anchor and the rest (black) are sub-anchors.

Defined on a grid structure that allows T-junctions (or T-mesh), the T-spline proposed in [7] 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.

Fig. 3. Knot vectors are derived from a T-lattice associated with a set of anchors (dots). The knot vector from \( k_a \) is \([0, 0, 1, 2, 2] \) in the -\( w \) direction, and \( k_b \) is \([0, 0, 0, 0, 0, 0] \) in the +\( u \) direction, where 2 replaces twice because \( L_0 \) intersects once with the rightmost boundary. The other knot vector from \( k_a \) toward -\( w \) direction has 0 repeated three times because \( L_1 \) intersects nothing from \( T \) and \( \mathcal{K} \) except \( k_b \).

We refer the knot vector in construction space by notation \( \Xi_i^j = [\xi_{i0}^j, \xi_{i1}^j, \xi_{i2}^j, \xi_{i3}^j, \xi_{i4}^j, \xi_{i5}^j] \) for the rest of the paper, unless mentioned otherwise. In the case of cubic T-splines, each
blending function must be associated with an anchor, which coincides with the middle knot of its three knot vectors in the construction space.

To infer knot vectors from a T-lattice is essentially the same as that for T-mesh, except that the searching is conducted in construction space. Starting from an anchor side determined by the rules in Table 1.

To extend polycubes to bounding-boxes, 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

$$N(\xi) = c_1 N_1(\xi) + c_2 N_2(\xi), \quad (4)$$

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

4 CONSTRUCTING RTP-SPLINES

The construction of RTP-splines includes four major steps (see Fig. 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 the 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 grid coordinates in $k$-axis direction are represented by

$$S_k = [s_1^k, s_2^k, \ldots, s_n^k] \quad k = 0, 1, 2$$

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 not difficult to construct a trivariate tensor-product B-spline on it. We may use $K_s$ as the knot vectors to define the trivariate B-spline. However, $K_s$ must be augmented to ensure that the definition is valid and covers the entire domain $T$. One method is to add extra knots outside the domain region, generating a floating-boundary scheme. In this paper, we duplicate the knots at both ends of $S_k$ in order to restrict the B-spline blending function within the domain $T$, i.e., $S_k$ turns into

$$S_k = [s_1^k, s_1^k, s_1^k, s_1^k, \ldots, s_{n_k}^k - 1, s_{n_k}^k, s_{n_k}^k, s_{n_k}^k, s_{n_k}^k]$$

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

$$F(u, v, w) = \sum_{i=1}^{n} p_i B_i(u, v, w) \quad (u, v, w) \in \mathbb{R}^3, \quad (5)$$

where $n = (n_0 + 2) \times (n_3 + 2) \times (n_2 + 2)$ is the number of control points, and $B_i(u, v, w)$ are blending functions defined in Eq. (3).

Alternatively, we can obtain $F$ by constructing blending functions similar to T-splines (Section 3.1), instead of computing them from 3 global knot vectors. We let $S = [s_{01}^1 + \epsilon, \ldots, s_{0n_0}^1 - \epsilon, s_{00}^0] \times [s_1^1, s_1^1, s_1^1, \ldots, s_{n_1}^1] \times [s_{1}^2, s_{1}^2, \ldots, s_{n_2}^2 - \epsilon, s_{n_2}^2] \times [s_1^3, s_1^3, \ldots, s_{n_3}^3 - \epsilon, s_{n_3}^3]$ and choose the anchor set $K = \{K_i(\alpha, \beta, \gamma) \mid K_i(\alpha, \beta, \gamma) \in S \}$, then build blending functions associated with each anchor. $K$ 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 refine 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 the blending function set:

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

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

(iii) No internal blending function influences any region outside the polycube domain if $G$ and all the ghost anchors are removed.
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>$\xi_0 \leq k &lt; \xi_1$</td>
<td>$\xi_0 - \xi_1$</td>
<td>$\xi_0 - \xi_1$</td>
<td>$[\xi_0, k, \xi_1, \xi_2, \xi_3]$</td>
<td>$[\xi_0, \xi_1, \xi_2, \xi_3, \xi_4]$</td>
</tr>
<tr>
<td>$\xi_1 \leq k &lt; \xi_2$</td>
<td>$\xi_1 - \xi_0$</td>
<td>$\xi_1 - \xi_0$</td>
<td>$[\xi_0, \xi_1, k, \xi_2, \xi_3]$</td>
<td>$[\xi_0, \xi_1, \xi_2, \xi_3, \xi_4]$</td>
</tr>
<tr>
<td>$\xi_2 \leq k &lt; \xi_3$</td>
<td>$\xi_2 - \xi_1$</td>
<td>$\xi_2 - \xi_1$</td>
<td>$[\xi_0, \xi_1, \xi_2, k, \xi_3]$</td>
<td>$[\xi_2, \xi_3, \xi_4]$</td>
</tr>
<tr>
<td>$\xi_3 \leq k \leq \xi_4$</td>
<td>$1$</td>
<td>$1$</td>
<td>$[\xi_0, \xi_1, \xi_2, \xi_3, k]$</td>
<td>$[\xi_1, \xi_2, \xi_3, \xi_4]$</td>
</tr>
</tbody>
</table>

**Fig. 4.** Overview of 4-step RTP-spline construction pipeline. (1) Extend polycube domain to its bounding-box; (2) Build B-spline volume with bounded boundaries; (3) Insert anchors and refine blending functions; (4) Remove exterior regions.

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

To achieve this goal, we systematically add new anchors in two steps. First, add sub-anchors at the polycube boundary vertices (Section 4.3.3). Second, keep inserting sub-anchors to refine those blending functions that violate the above rules, until there exist no violations. Adding new sub-anchors ultimately introduces duplicate knots into knot vectors, which serves for 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 places (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 $K$, an algorithm (Section 4.3.1) is necessary to resolve these inconsistencies after new anchors have been inserted.

**4.3.1 Local Refinement of Blending Functions**

We need to introduce an algorithm to update blending functions $B$ accordingly, once there occurs any change in the anchor set $K$ and/or the domain structure $T$. The refinement algorithm proposed in [1], [10] 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 Eq. 5 to

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

where $w_i$ is the weight associated with each blending function $B_i$. Note that the T-spline so far is essentially a B-spline volume: $\sum_{i=1}^{\vert B \vert} w_i B_i(u, v, w) \equiv 1$ for any $(u, v, w)$ and $w_i = 1$ for any $i$.

Let $K^*$ denote the updated anchor set and $T^*(V^*, C^*, F^*)$ be the new grid structure after vertex insertion or cell splitting. Given $K^*$, $T^*$, $W$ and $B$, Algorithm 1 generates new blending function set $B^*$ with new corresponding weights $W^*$, as well as the updated versions of $K^*$ and $T^*$.

In Algorithm 1, the superscript indicates the index of the original blending function from $B$, with which a variable is associated, and the subscript indicates the index of the associated central anchor. For example, $B^*_k$ is a blending function associated with anchor $k$, and originates from the $t$-th blending function from $B$. The star superscript indicates that the variables are obtained from the modified domain $T^*$, e.g., $\Xi^*_t$ denotes the knot vectors deduced from $T^*$ and centered at $k^*_t$ (i.e., the three middle knots of $\Xi^*_t$ coincide with $k^*_t$).

The basic idea of Algorithm 1 is as follows. First, we decouple blending functions from $T$ and $K$. Then, by either inserting new anchors or refining basis functions (Section 3.2), we keep resolving the inconsistencies between $B$ and the local knot vectors implied by $K^*$ and...
Algorithm 1: small Refinement of trivariate T-spline blending functions in support of duplicate knots.


Output: $T^*$, $K^*$, $B^*$ and $W^*$ such that $\sum_{i=1}^{|B|} w_i^* B_i^* \equiv \sum_{i=1}^{|B|} w_i B_i$

1. $Q \leftarrow \{(w_i^*, B_i^*) : w_i^* \in W, B_i^* \in B\}$
2. while $\exists (w_i^*, B_i^*) \in Q : \Xi_i^* \neq \Xi_i^*$ in parametric space do
3. for all $(w_i^*, B_i^*) \in Q$ do
4. infer the knot vectors $\Xi_i^*$ that is centered at $k_i^*$ from $T^*$
5. if $\Xi_i^* = \Xi_i^*$ in parametric space then
6. $\Xi_i^* \leftarrow \Xi_i^*$
7. else if $\Xi_i^*$ is more refined than $\Xi_i^*$ then
8. insert a knot of $\Xi_i^*$ into $\Xi_i$ which is not there, and do the refinement:
9. $B_i^* = c_1 B_i^* + c_2 B_i^*$ (Section 3.2)
10. $w_i^* \leftarrow w_i^* \cdot c_1$; $w_i^* \leftarrow w_i^* \cdot c_2$
11. $Q \leftarrow Q - \{(w_i^*, B_i^*)\} \cup \{(w_i^*, B_i^*), (w_i^*, B_i^*)\}$
12. else if $\Xi_i^*$ indicates an anchor $k_{j(\alpha, \beta, \gamma)} \notin K^*$ then
13. $K^* \leftarrow K^* \cup k_{j(\alpha, \beta, \gamma)}$
14. if $k_{j(0,0,0)} \notin K^*$ then
15. $K^* \leftarrow K^* \cup \{k_{j(0,0,0)}\}$
16. $V^* \leftarrow V^* \cup \{v_j\}$ // Insert a new vertex
17. end if
18. end if
19. end for all
20. for all $c \in C^*$ do
21. if vertices on the edges of $c$ form an axis-aligned plane that splits $c$ into $c_1$ and $c_2$ then
22. $C^* \leftarrow C^* \setminus \{c\} \cup \{c_1, c_2\}$ // divide $c$ into $c_1$ and $c_2$
23. end if
24. end for all
25. $B^* \leftarrow \{B_i : \{w_i^*, B_i^*\} \in Q\}$
26. $W^* \leftarrow \{w_j : \sum_{(w_i^*, B_i^*) \in Q} w_i^*\}$

$T^*$. A cell splits into half if there are vertices on its edges forming an axis-aligned plane. Finally, any blending functions arising from different refinements but having equivalent knot vectors in parametric space are merged into a single one with their weights being added together.

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 Eq. 6 remain unchanged in Algorithm 1, because of the fact

$$w_i^* B_i^* \equiv w_i^* \cdot c_1 B_i^* + w_i^* \cdot c_2 B_i^* \equiv w_i^* B_j^* + w_i^* B_i^*.$$  

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 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.

The anchor operation is our fundamental tool to modify existing blending functions of trivariate T-splines in order to get rid of all violations against rules (i), (ii) and (iii). As blending functions of trivariate T-splines are tensor-products of three univariate cubic B-spline bases, let us illustrate this method in 1D by using two examples given in Fig. 6. In (a), $N_0 = \mathcal{N}[-2, -1, 0, 1, 2]$ represents an internal basis which apparently violates rule (ii). If two extra knots 0s are inserted, $N_0$ is refined into two basis $N_1 = \mathcal{N}[-2, -1, 0, 0, 0]$ and $N_2 = \mathcal{N}[-1, 0, 0, 1, 1]$, one ghost $N_3 = \mathcal{N}[0, 0, 0, 1, 2]$, such that $N_0 = \frac{2}{3} N_1 + \frac{1}{3} N_2 + \frac{1}{3} N_3$ according to the refinement algorithm in Section 3.3. Since the ghost region is gone, $N_1$ and $N_2$ change to $N_1^* = \mathcal{N}[-2, -1, 0, 0, 0]$ and $N_2^* = \mathcal{N}[-1, 0, 0, 0, 0]$ respectively and we still have $N_0(0) = \frac{2}{3} N_1(0) + \frac{1}{3} N_2(0) + \frac{1}{3} N_3(0)$ as shown in the bottom of Fig. 6(a). Therefore, the violation against (ii) is successfully eliminated. Fig. 6(b) depicts a scenario where $N_0 = \mathcal{N}[-3, -2, -1, 0, 1]$ violates 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 rule (i). For trivariate T-splines, 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 is 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 Fig. 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 develop 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 \( k_i(\alpha,\beta,\gamma) \in \mathcal{K} \) for all \( k_i(\alpha,\beta,\gamma) \in \mathcal{K} \) and \( \text{color}(k_i(\alpha,\beta,\gamma)) = \text{color}(k_i(\alpha,\beta,\gamma)) \) for \( \alpha \in \{-1,1\} \) and \( \beta, \gamma \in \{-1,0,1\} \), we delete \( \{k_i(\pm1,\beta,\gamma)\} \) from \( \mathcal{K} \), that is, the sub-anchors on the 1st and 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 \( \mathcal{K} \). 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 Fig. 7 in which sub-anchors are inserted at a boundary vertex on 2D mesh. After all the required sub-anchors are 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.

**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 Fig. 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 \( k_{a(1,1)} \) and \( k_{a(1,-1)} \) can be added to reduce the support region to the boundary while no further violations being introduced. The violation arising from \( k_{f(1,-1)} \) is treated in the same fashion. In the case of \( k_{b(1,1)} \), only 1 sub-anchor \( k_{b(1,-1)} \) is required to eliminate the violation.

2) (see Fig. 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 \( k_{a(1,0)} \) because its knot vector goes into the ghost region. Similar to case 1, \( k_{a(1,1)} \) and \( k_{a(1,-1)} \) can be added to cut off the blending function from outside. Only one anchor insertion is necessary to resolve the violation arising from \( k_{f(1,-1)} \). Even though the new blending functions after refinement still covers nearby ghost region, it does not violate rule (ii) anymore. This has been explained in Section 4.3.2 (Fig. 6(a)).

3) (see Fig. 8(c)). We illustrate four different cases together in (c). Thus we consider the existence of only one blue anchor each time.) Ghost blending functions near a convex corner of \( P \) violate rule (i). For example, in spite of its knot vectors being apart from \( P \) and any internal anchors, the blending function associated with \( k_{a(0,0)} \) still influence the internal corner region. To remedy this violation, two sub-anchors \( k_{c(1,0)} \) and \( k_{k(0,1)} \) are added to the extended surfaces of the convex corner, shrinking the support region of the blending function centered at \( k_{a(0,0)} \) via refinement to separate it from \( P \). Similarly, \( k_{c(1,0)} \) and \( k_{b(-1,1)} \) are added for \( k_{a(-1,0)} \), \( k_{c(1,-1)} \) and \( k_{b(0,1)} \) for \( k_{a(0,-1)} \) and \( k_{c(1,-1)} \) and \( k_{b(-1,1)} \) for \( k_{a(-1,-1)} \).

4) (see Fig. 8(d)). There are also four independent cases presented in (d). Internal blending functions near a concave corner of \( P \) violate rule (iii). This type of violation is similar to case 3 except that the domain region and the ghost regions are interchanged and 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 eliminated during refinement, and there are only a finite number of sub-anchors that 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 \( \mathcal{K} \), we obtain a RTP-spline, which is a single-piece smooth function,
defined over a polycube domain \( P \). Our anchor insertion method guarantees that 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 Eq. 6 in a simpler formulation:

\[
F(u, v, w) = \sum_{i=1}^{\vert B \vert} w_ip_iB_i(u, v, w) \quad (u, v, w) \in \mathbb{R}^3. \tag{7}
\]

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 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 \([19],[20],[29]\) or automatically \([21],[22]\). These techniques also provide the boundary mapping \( g \) from the polycube boundary surface (denoted as \( \partial P \)) to the boundary of \( M (\partial M) \). We use such surface mapping \( g : \partial P \rightarrow \partial M \) as the boundary condition of \( f \). The volumetric parameterization is then defined as the seeking of a harmonic energy minimizer:

\[
\begin{aligned}
\Delta f(x) &= 0 \quad x \in P, \\
f(x) &= g(x) \quad x \in \partial P.
\end{aligned}
\]

A volumetric parameterization of a solid model \( M \) embedded in \( \mathbb{R}^3 \) on a polycube \( P \) is a bijection mapping \( f : P \rightarrow M, P, M \subset \mathbb{R}^3 \). The polycube \( P \) can be constructed either manually \([19],[20],[29]\) or automatically \([21],[22]\). These techniques also provide the boundary mapping \( g \) from the polycube boundary surface (denoted as \( \partial P \)) to the boundary of \( M (\partial M) \). We use such surface mapping \( g : \partial P \rightarrow \partial M \) as the boundary condition of \( f \). The volumetric parameterization is then defined as the seeking of a harmonic energy minimizer:

\[
\begin{aligned}
\Delta f(x) &= 0 \quad x \in P, \\
f(x) &= g(x) \quad x \in \partial P.
\end{aligned}
\]

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) \([24],[30]\). We recap the basic algorithm here and refer more details to \([24]\).

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 \( g \) on \( \partial P \).

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

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

where \( f \) is guaranteed harmonic, and we need to enforce the boundary condition on \( \partial P \). For boundary fitting, we sample \( N_c \) collocation points on the domain boundary \( \partial P \) to set up the constraint equations. If we have \( N_s \) charge points and \( N_c \) collocation points, for a real harmonic function \( f \) (e.g., along an individual axis direction) we only need to solve an \( Ax = b \) linear system where \( A \) is an \( N_c \times N_s \) matrix. The system can be efficiently solved by a truncated singular value decomposition [24], [31].

The parameterization 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. In fact, in RTP-spline construction, the parameterization without fully satisfying the conformality and equilibrium property does not have side effects 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 the following two directions, we would like to 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), polycubes [29], 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 the same favorable features of RTP-splines; (2) the current volumetric mapping is fully determined by the boundary constraint, i.e., the polycube surface mapping [19]. We can reduce the distortion by conducting relaxation of boundary surface mapping [32], now driven by the volumetric mapping distortion. However, this makes the mapping computation a nonlinear optimization which is inefficient.

### 5.2 RTP-spline Volume Fitting

Given \( f : P \rightarrow M \), 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 the RTP-splines \( F(u, v, w) \) 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

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

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 ([33]).

If the fitting results do not 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 degrees 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 slight modification. 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.

Compared with the number of degrees of freedom (DOFs) in the optimization problem (Eq. 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 lead to degenerate systems. In this case, we will first enlarge \( U \) by adding more points on the parametric domain near the regions where subdivisions take place and then recalculate \( X \).

### 6 Results and Discussion

<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>0.554</td>
<td>1150</td>
</tr>
<tr>
<td>Head</td>
<td>472122</td>
<td>12880</td>
<td>0.291</td>
<td>422.43</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>

We implement the entire volumetric parameterization, RTP-spline construction, and data fitting framework in C++ and perform experiments 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 being excavated), which are represented as tetrahedral meshes. We successfully convert them into representations of single-pieced smooth RTP-splines using our proposed algorithms. The experimental results are given in Fig. 11.

This step of the RTP-splines 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 datasets 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 the RTP-splines can be achieved 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 Beethoven are also gradually revealed with the increasing level of subdivision (see Fig. 10). RTP-spline is semi-standard and hence computing blending functions and their derivatives on the domain is much more efficient than on traditional T-splines. To empirically prove this, we compared computational cost on the models Bimba, Kitten, and two-hole torus with both kinds of spline representations. To ensure the fairness in comparison, 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 are 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 Fig. 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 Fig. 12(d)). The fitting results for the distance field and the corresponding fitting error map are demonstrated in Fig. 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 procedural 3D texture, generated using the fractal sum of Perlin noise [34] as $T(p) = \sum_{i=1}^{4} \frac{1}{4^{i}}noise(i \cdot p), p \in \mathbb{R}^{3}$ (see Fig. 12(g)). In the experiment, the value of $T(p)$ varies from 0.8 to −1.33. And the absolute RMS fitting error to $T(p)$ is $7.3 \times 10^{-4}$ (see Fig. 12(h)(i)).

As a RTP-spline function is continuously and smoothly defined over a polycube, we can evaluate any properties that depend 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 Fig. 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.** It has been proven in [35] that linear independence is not a guaranteed property on general T-splines, and the property of linear independence can be inherited from T-splines with coarser T-meshes through refinement operations, as long as new anchors are added in certain ways.

The RTP-spline is an extension of the trivariate $T$-spline, but its construction does not follow the rules proposed in [35] when new anchors are inserted. So the question of whether the blending functions of a RTP-spline are linearly independent remains open. Therefore in this paper, we call $B_i$ (in Eq. 7) a blending function instead of a basis function.

The volume fitting problem is solved in a least square system, which does not require the linear equations involved to be linearly independent. Therefore, the fitting

<table>
<thead>
<tr>
<th>Model</th>
<th>Sample Points</th>
<th>Polycube Spline</th>
<th>General T-spline</th>
</tr>
</thead>
<tbody>
<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>2.42s</td>
</tr>
<tr>
<td>2-Torus</td>
<td>9768</td>
<td>0.71s</td>
<td>0.35s</td>
</tr>
</tbody>
</table>

**TABLE 3**

Computational costs in calculating blending functions and their derivatives: RTP-splines v.s. general T-splines. The time includes total computation spent on blending functions values and their derivatives at all sample points. The blending functions used in comparison are defined as $B_i(u, v, w) = w_i B_i(u, v, w)$ for RTP-splines, and $\sum_{j=1}^{n} B_j(u, v, w)$ for T-splines, respectively.
results will not be severely affected no matter the RTP-spline blending functions are linear independent or not. In our experiment, we observed that if there is no sub-anchor introduced in the refinement step as discussed in Section 4.3.1, the blending functions of the RTP-splines thus constructed would remain linear independent. So it is possible to have a modified RTP-spline construction scheme using adaptive sub-anchor insertion that guarantees the linear independence property. We will explore this direction in the near future.

Fig. 10. Adaptive fitting of Beethoven (from top to bottom: fitting with 0, 1 and 2 levels of subdivision).

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 as a single piece smooth function with a restricted boundary, while ensuring lower parameterization 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.

At present, one unclear property of the RTP-spline is the linear independence of its basis functions, and we shall explore possible constraints during the RTP-spline construction and propose necessary or sufficient conditions that could guarantee such property. 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. We are planning to investigate the aforementioned topics.

Acknowledgements

This research is supported in part by US National Science Foundation grants IIS-1047715, IIS-1049448, IIS-0949467, and IIS-0710819, Louisiana Board of Regents Research Competitiveness Subprogram (RCS) LEQSF(2009-12)-RD-A-06, PFund: NSF(2009)-PFund-133, LSU Faculty Research Grant 2010.

References


Kexiang Wang received the BS and MS degrees in computer science from the University of Science and Technology of China in 1998 and 2001, respectively, and the PhD degree in computer science from Stony Brook University (SUNY) in 2010. He currently works as a data analyst at Renaissance Technologies, LLC.

Hongqin Qin is a Professor of Computer Science in Department of Computer Science at Stony Brook University. He received his BS degree in Computer Science from Zhejiang University in 1984. He received his MS degree in Computer Science from Stony Brook University (SUNY) in 1986. He received his PhD degree in Computer Science from the University of Toronto in 1997. Profesor Qin was awarded NSF CAREER Award from the National Science Foundation (NSF). He was also a recipient of Honda Initiation Award, and Alfred P. Sloan Research Fellow by the Sloan Foundation. Professor Qin served as the general Co-Chair for Computer Graphics International 2005. He was the Conference Co-Chair for ACM Solid and Physical Modeling Symposium in 2007. In 2008, he served as the Conference Chair for ACM Solid and Physical Modeling Symposium and IEEE International Conference on Shape Modeling and Applications. Currently, he is an associate editor for Graphical Models, The Visual Computer, and Journal of Computer Science and Technology. His research interests include geometric and solid modeling, graphics, physics-based modeling and simulation, computer aided geometric design, human-computer interaction, visualization, and scientific computing. For more details, please visit http://www.cs.sunysb.edu/~qin

Xin Li is an assistant professor of Department of Electrical and Computer Engineering and Center for Computational and Technology at Louisiana State University. He obtained his BS in Computer Science from University of Science and Technology of China, MS and PhD in Computer Science from Stony Brook University (SUNY). His research interests include geometric modeling, computer graphics, vision, and visualization. His recent works include curve, surface and volumetric shape mapping/matching, and their applications in digital forensics, computational medicine, and robotics. He is a member of IEEE Computer Society. For more information, please visit http://www.ece.lsu.edu/xinli.

Bo Li is a PhD candidate in Department of Computer Science at Stony Brook University (SUNY). He received his BS degree in Computer Science from ZhongShan University (Sun Yet-sen University). His research interests include computer graphics, geometric modeling, computer-aided geometric design (CAGD), physical animation/simulation, and visualization. For more details, please visit http://www.cs.stonybrook.edu/~bli
Fig. 11. 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 without being displayed due to the fact that their significantly large number will deteriorate the rendering quality.)
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. In addition, (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 $7.3 \times 10^{-4}$. 