Computers & Graphics 36 (2012) 329–340

Contents lists available at SciVerse ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

SMI 2012: Full Paper

Component-aware tensor-product trivariate splines of arbitrary topology ☆

Bo Li *, Hong Qin

Stony Brook University, United States

ARTICLE INFO

Article history:
Received 4 December 2011
Received in revised form
15 March 2012
Accepted 17 March 2012
Available online 29 March 2012

Keywords:
Tensor-product splines
Trivariate splines
Solid models of arbitrary topology
Semi-standardness
Boundary restriction

ABSTRACT

The fundamental goal of this paper aims to bridge the large gap between the shape versatility of arbitrary topology and the geometric modeling limitation of conventional tensor-product splines for solid representations. Its contribution lies at a novel shape modeling methodology based on tensor-product trivariate splines for solids with arbitrary topology. Our framework advocates a divide-and-conquer strategy. The model is first decomposed into a set of components as basic building blocks. Each component is naturally modeled as tensor-product trivariate splines with cubic basis functions while supporting local refinement. The key novelty is our powerful merging strategy that can glue tensor-product spline solids together subject to C^2 continuity. As a result, this new spline representation has many attractive advantages. At the theoretical level, the integration of the top-down topological decomposition and the bottom-up spline construction enables an elegant modeling approach for arbitrary high-genus solids. Each building block is a regular tensor-product spline, which is CAD-ready and facilitates GPU computing. In addition, our new spline merging method enforces the features of semi-standardness (i.e., $\sum_i w_i B_i(u,v,w) \equiv 1$ everywhere) and boundary restriction (i.e., all blending functions are confined exactly within parametric domains) in favor of downstream CAE applications. At the computational level, our component-aware spline scheme supports meshless fitting which completely avoids tedious volumetric mapping and remeshing. This divide-and-conquer strategy reduces the time and space complexity drastically. We conduct extensive experiments to demonstrate its shape flexibility and versatility towards solid modeling with complicated geometries and non-trivial genus.

1. Introduction and motivation

The rapid advancement of 3D data-acquisition techniques gives rise to a wide variety of tremendous 3D volumetric data. Although many 3D representations (e.g., structured grids or tetrahedral meshes) exhibit various virtues for solid modeling and processing, most of these representations are lack of the compactness of smoothly modeling solid geometry, which is required for CAE, including both CAGD modeling process and downstream physical analysis without data conversion. Therefore, a frequently occurring challenge is how to effectively convert a discrete complex volumetric data into a compact and continuous spline formulation towards CAE applications.

Our primary goal in this paper is to develop efficient methods for arbitrary solids undergoing spline transformation. Nevertheless, we must address the following key challenges. (1) *High genus*: an attractive spline representation must accommodate high-genus solid models with complicated shapes. (2) *Local refinement and adaptive fitting*: for trivariate splines, both structurally complicated shape models and feature-enriched models need local refinement. For example, a genus-0 solid bounded by six simple four-sided B-spline surfaces has originally 6×1024^2 control points (DOFs). The size of DOFs increases drastically to 1024^3 or even larger when we naively convert it into a volumetric spline representation. This exponential increase poses a great challenge in terms of both storage and fitting costs. It is advantageous to use high resolution to approximate boundary surface and low resolution for interior space. (3) *Singularity free*: a *singular point in a volumetric domain* is a node with valence larger than four along one iso-parametric plane (Fig. 1(a)). Handling singularity with tenor-product splines is highly challenging in FEM, thus a singularity-free domain is highly desirable. Unfortunately, singularities commonly exist in many volumetric domains such as hexahedral meshes and cylinder (tube) domains. (4) *Boundary restriction*: It is a basic requirement for a spline that all blending functions are completely confined within the parametric domain. (5) *Semi-standardness*: In a hierarchical spline, the sum of weighted basis function $\sum_{i=1}^{\mathcal{B}} w_i B_i(u,v,w) \equiv 1$ holds for all (u,v,w). It has a broader appeal to both theoreticians and practitioners.

Recently, much work has been attempted towards the aforementioned requirements while following a top-down fashion. For example, Wang et al. [1] have proposed a spline scheme

☆ If applicable, supplementary material from the author(s) will be available online after the conference. Please see http://dx.doi.org/10.1016/j.cag.2012.03.007.

* Corresponding author.
E-mail address: bli@cs.stonybrook.edu (B. Li).

0097-8493/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cag.2012.03.007

being built upon a one-piece volumetric poly-cube domains. Poly-cube is a shape composed of cuboids that abut with each other. Although this method successfully inherits many attractive properties, it also exposes all typical difficulties in a top-down scheme. A one-piece poly-cube domain, together with its 3D embedding, is not versatile enough to handle highly twisted and high-genus solid datasets. Creating a poly-cube to mimic the input shape requires tedious user intervention. The boundary restriction procedure in the vicinity of gluing regions (Fig. 2, dots/lines) is extremely complicated. Computationally speaking, the global fitting is very time consuming which is completely unsuitable for trivariate splines.

To ameliorate, our framework takes advantage of the bottom-up scheme. The global domain is first divided into several components. Then we build tensor-product trivariate splines separately for each component, and finally glue them together. Figs. 3 and 4 show the detailed, step-by-step procedure using the model (called "g3") as an example. Specifically, it includes the following major phases:

(1) Construct a surface poly-cube mapping: To better support our divide-and-conquer scheme, we use the technique [2] to decompose the entire surface model into several components. Each component is a part-aware surface patch and we map it to the boundary surface of a cuboid (Section 3). The separate cuboid mappings are also globally aligned.

(2) Construct a local trivariate tensor-product T-spline on each cuboid (Section 4). Adaptive fitting is allowed for a better fitting result.

(3) Merge local cuboids into a single global spline (Section 5). Note that, the novelty of our merging strategy lies at its comprehensive and complete solution to guarantee the desirable properties: semi-standardness and boundary restriction.

Our new shape modeling framework has the following significant contributions on both spline theory and practice: compared with prior top-down strategies, our new divide-and-conquer approach is more flexible and powerful to handle complex solids with arbitrary topology. Each component can be easily converted to a trivariate semi-standard regular spline, which is embraced by industry-standard CAD kernels and facilitates GPU computing like [3].

At the theoretical level, we develop the theory and algorithm to merge adjacent trivariate splines together. Through adding knots and modifying weights, our merging method can enforce semi-standardness and boundary restriction for all possible merging types, even after local adaptive refinement.

For solids with homogeneous material, we are capable of generating trivariate splines from poly-cube surface parameterization directly, thus we avoid complicated interior volumetric remeshing. Moreover, our divide-and-conquer strategy makes the modeling and analysis tasks scalable to large-scale volumetric data, in terms of computation time and space consumption during the fitting.

2. Related work and background review

Spline-based volumetric modeling and analysis have gained much attention recently with many applications. For geometric processing, Song et al. [4] have employed trivariate splines with non-uniform weights to model free-form deformation. For physical analysis, Hughes et al. [5] have proposed isogeometric analysis on surface using bivariate NURBS, and conducting physical analysis simultaneously. In visualization, Rössl et al. have utilized

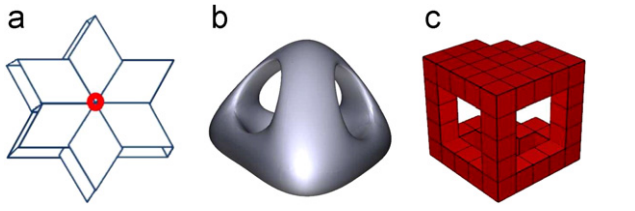

Fig. 1. (a) The singular point in the volumetric domain. (b and c) A poly-cube domain can mimic the geometry of input and avoid such type of singular point.

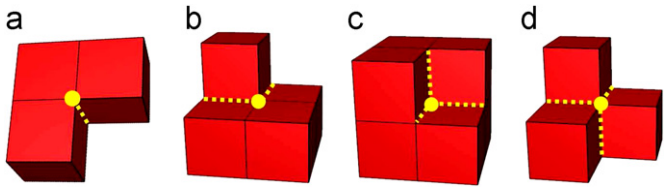

Fig. 2. All possible merging types in a poly-cube (from "Type-1" to "Type-4"). To preserve both boundary restriction and semi-standardness, we add extra knots around the control points on the merging boundary (yellow lines and dots). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

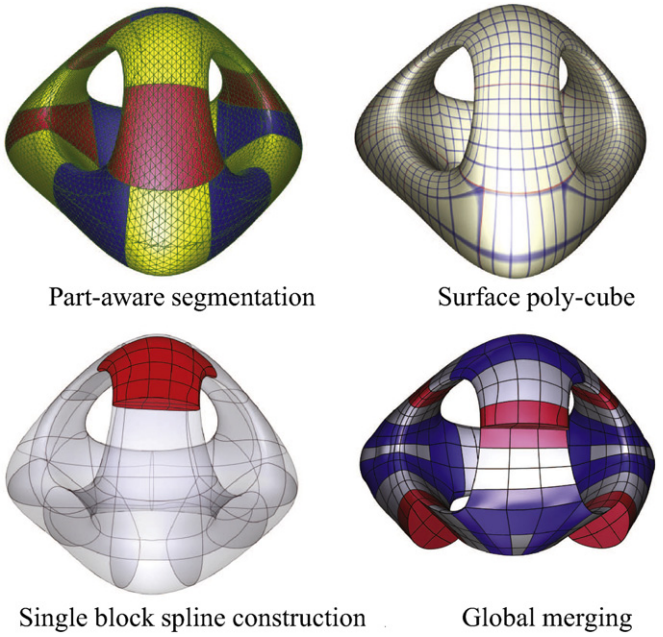

Fig. 4. Steps to convert the g3 model into a trivariate spline solid.

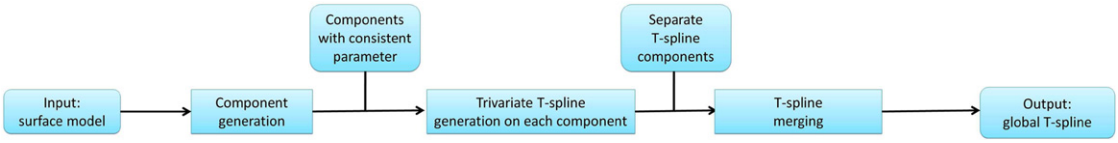

Fig. 3. The divide-and-conquer scheme. The system adopts the general triangular mesh as the input. The output trivariate splines interpolate both the input surface and interior volume.

trivariate super splines [6] to model and render multi-dimensional material attributes for solids. We, in this paper, focus on converting a closed surface bounded region into a trivariate spline (for readers who are interested in surface-to-bivariate spline, we refer them to [7,8]). A modeling technique introduced in [9] has been developed to model skeletal muscle with anisotropic attributes and conduct FEM analysis directly on NURBS solid. Martin et al. [10] have presented a method to fit a solid model using a cylindrical trivariate NURBS and support continuum force analysis. However, these existing spline schemes tend to handle only simple inputs like genus-0 surfaces. For more complicated shapes, Zhang et al. [11] have proposed the method to convert the long-branch/bifurcation dominant shapes. Martin et al. [12] have studied shapes with a symmetry (called "mid-face") structure. These methods always attempt to transform the model through a top-down scheme, which inspires us to explore a new method in a divide-and-conquer fashion.

To better support physical analysis directly on continuous representations, we concentrate on splines with two desirable properties: semi-standardness and boundary restriction. The concept of semi-standardness is founded upon the standard T-spline technique, which is invented in [13]. It permits T-junctions on its control grid and enables local knot insertion. The semi-standard T-spline in [14] simplifies the local refinement method and guarantees that the summation of all weighted blending functions equals to one everywhere across the entire parametric domain. The boundary restriction is usually implemented by non-uniform splines. On surface splines, Sederberg et al. [15] have discussed relations between knot intervals and subdivision surfaces on arbitrary topology. This idea is further extended to the T-spline setting [13]. Our new construction method requires the merging of splines defined over different local domains. Surface patch merging has been thoroughly discussed first in [13,16] and later is used in [17], in order to glue the trimmed region to form a single spline. However, it is far more complicated to design semi-standard trivariate splines which demand much more in-depth studies.

Compared with surface splines designed to extract features (e.g., [7,18]), our trivariate splines mainly focus on finding part-aware component structures. Besides poly-cube domains, another commonly used part-aware domain is cylinder (tube) [10]. Martin et al. in [12] have extended this domain to mimic more complex shapes. However, in terms of spline construction, the cylinder (tube) domain inevitably produces singular points along the tube axis. Handling singularity while enforcing high-order continuity is extremely difficult in spline research. For surface modeling, Loop and Scheafer in [19] have given an example of a G^2 polynomial construction with general connectivity to handle singularities. On the other hand, Peters and Fan [20] have introduced rational linear maps to replace affine linear atlas and handle singularities between charts. We observe that poly-cube can use part-aware cuboids as building block and avoids any singularities. Thus it serves very well as the trivariate spline domain. It is pioneered in [21] for seamless texture mapping and can be used as a parametric domain for spline construction like in [8]. However, although poly-cube domain can be constructed in an automatic fashion like [22,23], in practice, users may have to rely on manual construction for fine quality control and model refinement. The main challenges include how to detect and decompose the input into part-aware components, and connect them in a singularity-free way.

Conventionally, when converting a surface input to a spline-ready format, the first step is meshing or remeshing the interior volume into a tetrahedral mesh [24] or any other format [25]. Then we compute the volumetric parameterization on the remeshing result. A few recent works [12,26,27] have studied the volumetric parameterization calculated on the tetrahedral mesh. They typically start from a surface mapping as the boundary constraint. These interior mapping methods always involve time-consuming numerical procedures. It would be intriguing to ask whether we could better embed volumetric mapping step into our framework.

3. Component generation and T-splines

This section briefly reviews the required surface poly-cube generation algorithm. We also define the necessary notations for the rest of the paper. In the interest of understanding, most illustrative figures about knots are simply shown in 2D layout, as their 3D generalizations are straightforward.

3.1. Component generation

The starting point of our whole procedure is to generate several components from the input. We take the general triangular mesh as the input. Each component surface is part-aware and maps to a cuboid face. Meanwhile, an appropriate decomposition and mapping must obey the following rules. *Rule 1*: Parameters between neighboring components are consistent (i.e., we can directly glue their parameters together as a seamless globally aligned poly-cube mapping); *Rule 2*: The decomposition result avoids cube gluing that could produce singularities like Fig. 1(a).

We remain agnostic as to which method should be used for such decomposition. However, in order to better enforce these requirements, we utilize the algorithm [2] for this step. The algorithm is briefly summarized here as the resulting cuboid-connecting structure is essential for introducing our spline merging algorithm. The core idea of the algorithm resides in conducting the topology surgery to get the uniform "T-shape" (Fig. 5(a)), then maps each one to four cuboids. The resulting poly-cube has very simple cuboid connections and obeys both *Rule 1* and *Rule 2*, with a fully automatic generation procedure. Fig. 6 visualizes our algorithm.

Step 1: We first construct an abstract graph to encode the model topology. We take shape diameter functions [28] to detect prominent component-aware branches. Upon a completed detection, we construct a topology abstraction graph G (Fig. 7(a)): each node represents a detected branch and we add an edge between two nodes if two corresponding branches are immediate neighbors. If a branch has a handle, we add into the graph an edge with both ends on the same node to form a loop.

This topology abstract graph now becomes an ideal tool to decompose the model and obey *Rule 2*. To achieve this, we split and remove the nodes in the graph (i.e., re-segment and merge the branches) to get all nodes' degree $n \leq 3$:

- For each branch with a torus, we generate the shortest handle path [29] and cut along the path.
- For each branch with n boundaries (node degree $n > 3$, like Fig. 7(b)), we choose two boundaries (a pair with the closest

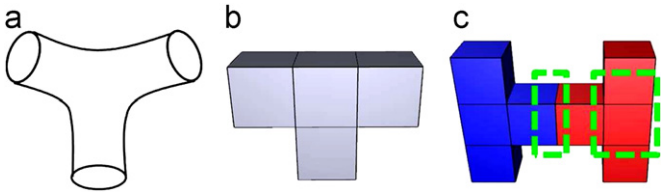

Fig. 5. The idea of component generation. (a) Decompose the model into T-shapes. (b) Map each T-shape into four cuboids. (c) Only two merging types exist in the resulting poly-cube.

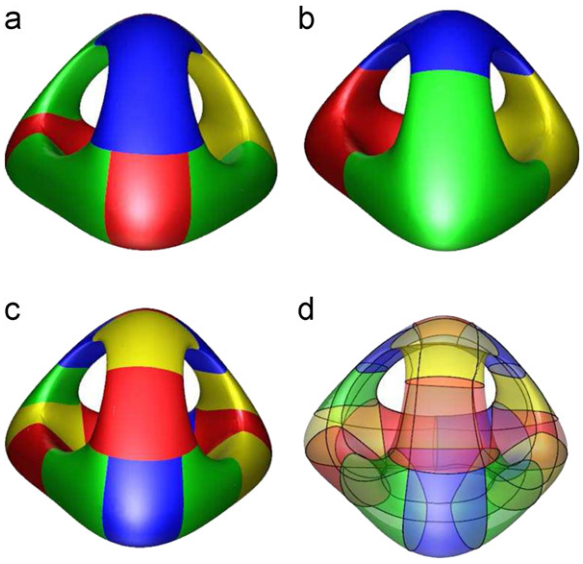

Fig. 6. Illustration of component generation. (a) The g3 model. Color encodes the topology structure. (b) T-shapes. (c) All T-shapes are converted to four cuboids. (4) The "poly-edges" (preimage of cuboid edges, dark lines) guarantee globally aligned poly-cube mapping. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

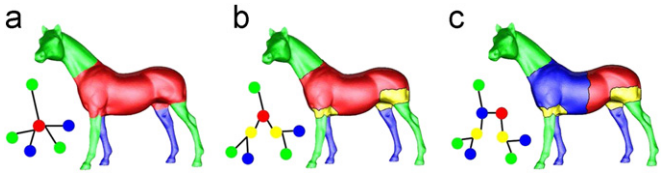

Fig. 7. Illustration of T-shape decomposition. (a) Encode the topology by an abstract graph. (b) Reduce the node degrees. (c) The resulting T-shapes.

distance) and then generate a bounding loop as in [30]. We cut along the loop, generating a 3-boundary and a $(n-1)$-boundary branch. We iteratively execute this step until all resulting branches have three boundaries.

- For each branch with 1 or 2 boundaries (like a "tube" shape), we merge the short branch into a connected 3 boundary neighbor. We also split a "tube" into two if its axis is too long.

Each resulting branch with three boundaries is geometrically similar to a volumetric "T" (*"T-shape"*), thus we can easily map it to four cuboids and avoid violating *Rule 2*.

Step 2: In order to transform a T-shape to four cuboids, we trace curves on every T-shape. These curves cut each T-shape into four patches (Fig. 6(c)). The resulting patch may include several cutting boundaries and we fill them by [31], converting the patch into a closed genus-0 surface. Each patch is bounded by 12 curves and they will be mapped onto the cuboid domain edges. (Thus we call these curves *"poly-edges"*, like grey lines in Fig. 6(d).) Note that the poly-edges between two connected patches are aligned. Their parameters on the common boundary are identical, thus we avoid violating *Rule 1*.

Step 3: We map each patch to a cuboid surface (Fig. 4(b)). We first map the 12 poly-edges onto cuboid edges. Then we use this mapping as the constraint and compute three harmonic equations $\Delta u=0$, $\Delta v=0$, $\Delta w=0$. In practice, we solve every equation on the discrete triangular mesh by mean value coordinates [32]. We also locally modify the coordinates along boundaries between two connected patches to keep the parameters aligned and consistent.

Advantages: Compared with the conventional poly-cube mapping method like [21], our construction is specifically suitable for the divide-and-conquer strategy and spline construction. (1) The conventional method always generates an integral poly-cube domain to mimic the whole shape at first. Then we have to decompose this integral domain into small pieces for applying the divide-and-conquer strategy. In contrast, our method directly uses a small set of connected local cuboids, each of which represents a geometrically meaningful patch (e.g., part-aware). This property is particularly suitable for highly twisted/non-axis-aligned/high-genus models (e.g., the g3 model). More importantly, we can use the divide-and-conquer technique directly on our resulting poly-cube without further decomposing the domain. (2) Our method can also reduce the number of cuboids, and control the merging types efficiently: as shown in Fig. 5(c), it only generates "Two-cube" and *"Type-1"* (Fig. 2(a)) merging, thus it simplifies the merging requirement.

3.2. Trivariate T-splines

To better prepare readers for the better understanding of the following algorithm, we briefly define the volumetric T-spline representation. (The surface T-spline formulation is detailed in [14].) Also we give the detailed explanation of *"Semi-standardness"* and *"Boundary Restriction"* as follows.

We use $T(\mathcal{V},\mathcal{F},\mathcal{C})$ (or simply T) to denote a control grid domain, where $\mathcal{V}$,$\mathcal{F}$, and $\mathcal{C}$ are sets of vertices, faces, and cells, respectively. Given T, a trivariate T-spline can be formulated as

$$\mathbf{F}(u,v,w)=\frac{\sum_{i=1}^{\mathcal{B}} w_i\mathbf{p}_i B_i(u,v,w)}{\sum_{i=1}^{\mathcal{B}} w_i B_i(u,v,w)}, \tag{1}$$

where (u,v,w) denotes parametric coordinates, $\mathbf{p}_i$ is a control point, $\mathcal{W}$ and $\mathcal{B}$ are the weight w_i and blending function B_i sets. Each pair of $\langle w_i B_i\rangle$ is associated with a control point $\mathbf{p}_i$. Each $B_i(u,v,w)\in\mathcal{B}$ is a blending function:

$$B_i(u,v,w)=N^3_{i0}(u)N^3_{i1}(v)N^3_{i2}(w), \tag{2}$$

where $N^3_{i0}(u)$, $N^3_{i1}(v)$ and $N^3_{i2}(w)$ are cubic B-spline basis functions along u,v,w, respectively.

In the case of cubic T-spline blending functions in Eq. (1), the univariate function N^3_j for each blending function B_i is constructed upon knot vector $R^j=[r^j_{-2},r^j_{-1},r^j_0,r^j_1,r^j_2]$, where R^j is a tracing ray parallel to the control grid (see Fig. 8(b)): Starting from a knot $k=r^0_0,r^1_0,r^2_0$, we can trace to r^0_1 and r^0_{-1}, which are the very first intersections when the ray $R(t)=(r^0_0\pm t,r^1_0,r^2_0)$ comes across one cell face. Naturally, we define the parameter of a control point as the central knot of the knot sequence for the control point.

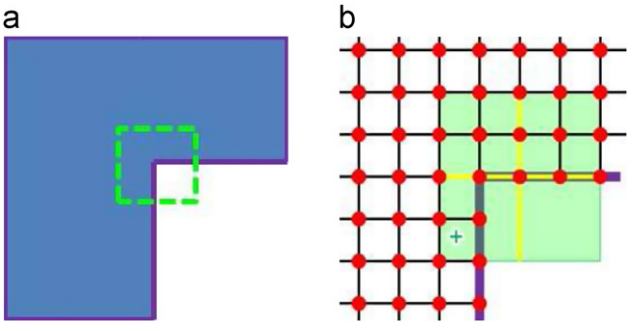

Fig. 8. Counter-example of boundary restriction. (a) A "Type-1" merging in a 2D layout. (b) The blending function's supporting region (green box) crosses the boundary. The supporting region is determined by tracing rays (yellow lines). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

To support downstream CAE applications, our spline framework has the following requirements:

Semi-standardness: $\sum_{i=1}^{\mathcal{B}} w_i B_i(u,v,w) \equiv 1$ holds for all (u,v,w) in Eq. (1), so that the evaluation of spline functions and their derivatives is both efficient and stable. Eq. (1) can be rewritten as

$$\mathbf{F}(u,v,w) = \sum_{i=1}^{\mathcal{B}} w_i \mathbf{p}_i B_i(u,v,w). \tag{3}$$

Boundary restriction: We require that blending functions of all control points are strictly confined within parametric domain boundaries. Unfortunately, achieving this requirement is not trivial, especially around the cuboid merging regions. Fig. 8 shows a counter-example. A standard control point's blending function (green box), without confinement procedure, tends to intersect with the boundary. In CAE-based force analysis, it means the strain energy "escapes the border", which might lead to an abrupt bend, twist, and flip-over phenomena in experiments. In the follow sections, we usually use *"central points"* for the control point/knot with an unconfined blending function, since the confinement procedure is mainly through adding extra knots/control points around the central point. However, even we design the additional knots carefully and successfully confine the blending function, we still have to recompute all control points' weights around the knots-adding region, otherwise we will break the semi-standardness around this local region.

4. T-spline construction for each component

The construction of trivariate splines on each component is very critical in our divide-and-conquer method. Two major goals are involved in this step. Besides constructing T-splines preserving desirable features, we have to satisfy the necessary requirement in each component in anticipation for merging. We propose the following procedure to satisfy both goals:

Step 1. Construct a boundary restricted control grid.

Step 2. Perform the meshless fitting to determine locations of all control points.

Step 3. Subdivide the control grid via local refinement iteratively. Perform fitting again in each iteration for a better fitting result.

Step 4. Modify the control grid around merging boundary after each subdivision iteration in anticipation for merging.

4.1. Boundary-restricted control grid

In order to construct a control grid, we first divide the cuboid block into cells by grid coordinates. The grid coordinates along k-axis are denoted as

$$\mathbf{S}_k = [s_1^k, s_2^k, \ldots, s_{n_k}^k], \quad k = 1,2,3,$$

where n_k is the resolution of rectilinear grid along k-axis and each value in $\mathbf{S}_k$ is the normal subdivision of cuboid parameter along k-axis. The tensor-product of $\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3$ divides the block into $(n_1-1)\times(n_2-1)\times(n_3-1)$ cells and gives rise to a point-based spline on $n_1 \times n_2 \times n_3$ control points.

However, this naive spline construction leads to open boundary and violates the requirement of boundary restriction. To improve, we replicate the non-uniform knots at both ends of $\mathbf{S}_k$ to restrict the blending functions within the domain (see Fig. 9(a)top): We add 3 extra knots, called *boundary knots (bd-knots)*, at the end of domain to restrict the boundary. The knot set is expanded:

$$\mathbf{S}_k = [s_1^k, s_1^k, s_1^k, s_1^k, s_2^k, \ldots, s_{n_k}^k, s_{n_k}^k, s_{n_k}^k, s_{n_k}^k].$$

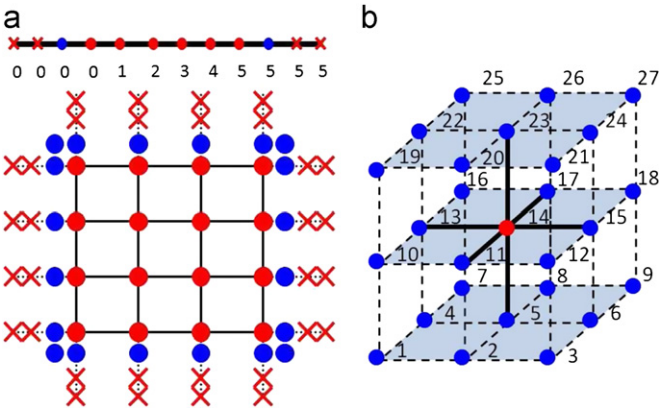

Fig. 9. (a) Top: Boundary restriction is illustrated on a 1D domain, with six "boundary knots" (or called "bd-knots", [0,0,0] and [5,5,5]) and two "boundary control points" (or called "bd-control-points", blue dots) inserted. (a) Bottom: Boundary-restricted control grid in a 2D layout. (b) All possible bd-control-points around one central point. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

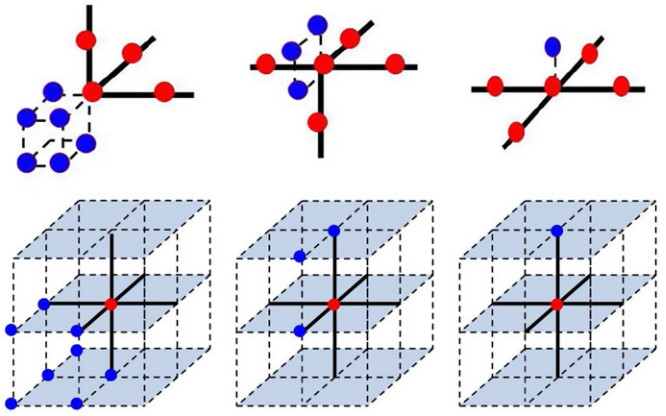

Fig. 10. The bd-control-point distributions around a central point on the corner/edge/face vertex, respectively.

We also add one extra *boundary control point (bd-control-point)* (blue dots), on the bd-knot outside the last control point on the boundary. Fig. 9(a)bottom extends it to a 2D domain, and its extension to the 3D domain is in the same pattern. Our spline definition achieves: (1) now every blending function in each domain is confined within the domain boundary; (2) only bd-control-points' blending functions influence the cuboid boundary, so our following fitting method can rely on this usable property.

In order to represent the bd-control-points conveniently, we can arrange them into a $3\times3\times3$ grid around the central point as Fig. 9(b) (recall that the central point is the control point with an unconfined blending function). These 27 possible knots share the same parameters as the central point. It is only designed to explicitly record topological relations of these control points in preparation for efficient spline merging. After adding bd-control-points to the 3D control grid, each central point on the corner/edge/face has 8/4/2 control points, respectively (Fig. 10). This special bd-control-point representation is uniquely suitable for handling the spline merging as shown in Section 5.

4.2. Meshless fitting

Our input only includes a control grid and a group of surface sample points extracted from the surface patch (already mapped to a cuboid domain surface). The challenge consists in designing

a fitting method for solids without interior volumetric parameterization or remeshing.

Step 1: Boundary fitting. We first determine the positions of bd-control-points only. Recall that only bd-control-points $\mathbf{p}_i^b$ influence the cuboid surface sample points. Therefore, we can determine their positions by minimizing the following equation w.r.t. surface sample point $\mathbf{v}_j^b$:

$$\operatorname{argmin}\left(\sum_{j=1}^{m}\|\mathbf{F}(f^{-1}(\mathbf{v}_j^b))-\mathbf{v}_j^b\|\right) \tag{4}$$

$$\Rightarrow \frac{\partial}{\partial \mathbf{p}_i^b}\sum_{j=1}^{m}(\mathbf{F}(f^{-1}(\mathbf{v}_j^b))-\mathbf{v}_j^b)^2,$$

where $\mathbf{F}$ denotes the spline function as Eq. (1) and $f^{-1}(\mathbf{v}_j^b)$ the parameters of $\mathbf{v}_j^b$ in the cuboid. The above equation can be rewritten in matrix format as in the least squares method:

$$\tfrac{1}{2}\mathbf{P}^T\mathbf{B}^T\mathbf{B}\mathbf{P}-\mathbf{V}^T\mathbf{B}\mathbf{P}=0, \tag{5}$$

where $\mathbf{B}$ is the matrix of blending functions $\mathbf{B}_{ij}=I_{3\times 3}\mathbf{B}_i(f^{-1}(v_j^b))$, $\mathbf{V}$ and $\mathbf{P}$ denote the vectors of surface sample points $\mathbf{v}_j^b$ and bd-control-points $\mathbf{p}_i^b$, respectively. This equation determines bd-control-points and they serve as the constraint in the next interior fitting step.

Step 2: Interior fitting. Let $\mathbf{u}$ in the set $\mathbf{U}$ be the interior parametric value. Each $\mathbf{u}_i=(u,v,w)$ is the interior parameter triplet in the tensor-product parametric grid $(u_0,u_1,\ldots,u_{n_0})\times(v_0,v_1,\ldots,v_{n_1})\times(w_0,w_1,\ldots,w_{n_1})$. Theoretically, we have the following harmonic equation w.r.t. interior control points $\mathbf{p}_j^{in}$:

$$\operatorname{argmin}\left(\sum_{i=1}^{m}\int_{\Omega_i}\|\nabla\cdot\nabla\mathbf{F}(\mathbf{u}_i)\|\,d\mathbf{u}\right) \tag{6}$$

$$\Rightarrow \frac{\partial}{\partial \mathbf{p}_j^{in}}\sum_{i=1}^{m}\int_{\Omega_i}(\Delta\mathbf{F}(\mathbf{u}_i))^2\,d\mathbf{u}=0,$$

where Ω_i is an infinitesimal parametric volume around $\mathbf{u}_i$. Similar as [33], the above minimized energy $\int_{\Omega_i}\|\Delta\mathbf{F}(\mathbf{u}_i)\|$ can be approximated by the following formulation:

$$\sum_{j=0}^{m}w_{ij}\mathbf{F}(\mathbf{u}_j)=0,\quad w_{ij}=\begin{cases}1, & i=j,\\ -\frac{1}{6}, & \mathbf{u}_j\in\mathbf{Nbr}(\mathbf{u}_i),\\ 0 & \text{others},\end{cases} \tag{7}$$

where **Nbr** includes six immediate neighbors of $\mathbf{u}_i$ in the tensor-product parametric grid. We substitute Eq. (7) into Eq. (6), which can be solved by the least squares method similar to Eq. (4). During computing we set already-known $\mathbf{p}_i^b$ as constraints and get all other control point positions.

Global alignment: Although we execute volumetric fitting separately on every cuboid, our fitting technique still guarantees global alignment of interior fitting results. Recall that we already obtain the identical surface parameters between cuboids before fitting, since we generate globally aligned poly-edges (i.e., cuboid edges). Therefore, two cuboids minimize precisely the same energy in Eqs. (4) and (6) on the boundary, leading to the equivalent fitting results.

4.3. Cell subdivision and local refinement

If the fitting results do not meet a user-decided satisfactory fitting error threshold on each cuboid, we can always perform subdivision over such a cell: It is split along 3-axis and divided into eight sub-cells naturally. Our iterative refinement stops when the updated fitting error is lower than the threshold.

The challenge is how to preserve the semi-standardness during subdivision. Sederberg et al. [14] have proposed a feasible approach to refine blending functions on surface patch. We generalize this technique in our 3*D* control grid. Let $R=[r_0,r_1,r_2,r_3,r_4]$ be a ray-tracing knot vector and $N_R(u)$ denotes the corresponding cubic B-spline basis function. If there is an additional knot $k\in[r_0,r_4]$ inserted into R, N can be written as a linear combination of two B-spline functions:

$$N_R(u)=c_1N_{R_1}(u)+c_2N_{R_2}(u). \tag{8}$$

Table 1
Refining N_R by inserting k into knot vector $[r_0,r_1,r_2,r_3,r_4]$ generates two basis functions N_{R_1} and N_{R_2}.

k	R_1	R_2
$r_0\le k<r_1$	$[r_0,k,r_1,r_2,r_3]$	$[k,r_1,r_2,r_3,r_4]$
$r_1\le k<r_2$	$[r_0,r_1,k,r_2,r_3]$	$[r_1,k,r_2,r_3,r_4]$
$r_2\le k<r_3$	$[r_0,r_1,r_2,k,r_3]$	$[r_1,r_2,k,r_3,r_4]$
$r_3\le k\le r_4$	$[r_0,r_1,r_2,r_3,k]$	$[r_1,r_2,r_3,k,r_4]$

Two knot vectors R_1, R_2 are shown in Table 1, c_1 and c_2 are two weights that can not exceed 1:

$$c_1=\min\left(\frac{k-r_0}{r_3-r_0},1\right),\quad c_2=\min\left(\frac{r_4-k}{r_4-r_1},1\right).$$

Since the blending function of B is the tensor product of N along 3-axis, we can also formulate the refined blending functions along one axis:

$$B_i\equiv c_1B_{i1}+c_2B_{i2}. \tag{9}$$

The procedure of our 3D subdivision and local refinement consists of following steps. The input is a queue of cell Q_c.

Step 1: Subdivide cells in Q_c and insert the new vertices into the domain T, and update T to T^*.

Step 2: For all pairs of blending functions $\langle w_iB_i\rangle$, $w_i\in\mathcal{W}$, $B_i\in\mathcal{B}$, compute its new knot vector R^* (see Section 3). Then,

- If the R^* includes the knot which does not exist in T^*, insert a new vertex on that knot into the domain T^*.
- If the R^* is more refined than R, compute the refinement $B_i=c_1\times B_{i1}+c_2\times B_{i2}$. Insert the new blending functions $\langle w_i\times c_1B_{i1}\rangle$ and $\langle w_i\times c_2B_{i2}\rangle$ into the control grid. Delete the old pair $\langle w_iB_i\rangle$.

Step 3. Repeat the last step until no new knot vector in R^*. Collect all blending functions on the same control point and use the total weight as its new weight.

The above procedure can handle refinement and knot extraction on a complicated 3D control grid automatically. Note that unlike [14], we perform spline fitting again after each refinement iteration to update control point positions. This is mainly because our goal of refinement is to seek a more accurate fitting result. In contrast, the refinement in [14] aims to keep the shape unchanged.

4.4. Boundary modification

Boundary modification is necessary for our semi-standard T-spline component, because of the fundamental difference between standard B-spline and our merging strategies. Fig. 11 intends to visually show the difference. It illustrates the 1D merging method introduced in [13] on our boundary-restricted grid. For a C^2 merging, three control points on one component will be merged with three control points on the other component to form a joint new spline. However, the procedure does not take the associated weights into consideration. In standard B-spline, all the weights are uniform. However, in semi-standard T-spline, it is possible that two corresponding soon-to-be-joined control points have different weights. As a result, the semi-standardness around the merged regions will break down. Therefore, we have to add

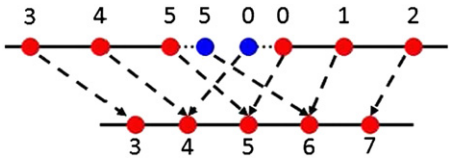

Fig. 11. Two cube merging in 1D layout. Two control points are combined to form one new control point (4, 5 and 6).

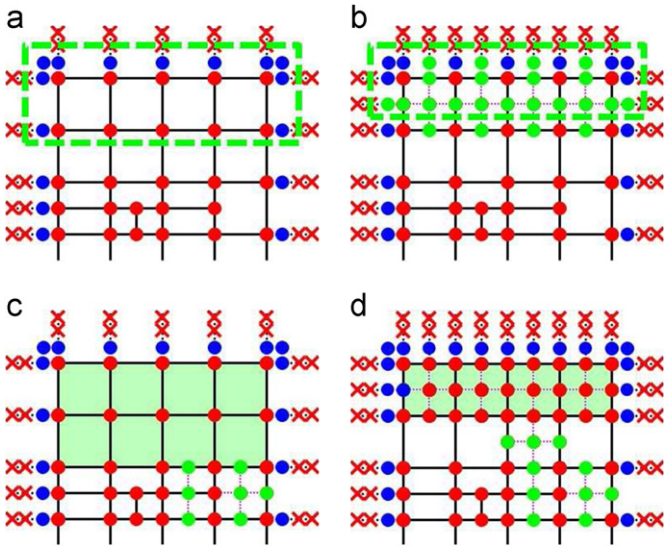

Fig. 12. Boundary modification. (a) Original "To-be-merged" control points (in the green box). (b) Subdivision all cells along the boundary, according to Proposition 3. The green box covers updated "To-be-merged" control points. (c) and (d) "Modification zone" (green box) of (a) and (b). According to Proposition 2, cell subdivision (by green dots) outside "Modification zone" does not violate "Boundary requirement" (Proposition 1). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

extra requirement about weights to make these control points ready for merging.

Definition 1 (*To-be-merged control point*). For a control point, if its blending function includes bd-knots around merging boundary, we say this control point is a "To-be-merged" control point (for example, Fig. 12(a and b) in a 2D layout).

Definition 2 (*Modification zone*). For any cell in the control grid, if one of its eight vertices is "To-be-merged" control point for one boundary, we say this cell is in the "Modification zone".

A merging-ready spline must have the following properties:

Proposition 1 (*Boundary requirement*). *The weights of all "To-be-merged" control points on this boundary must equal to one, such that we can merge two splines and the resulting spline still preserves semi-standardness* (*see proof in Appendix* [34]).

To guarantee that all "To-be-merged" control points' weights equal to one, we need to be able to recognize if a local subdivision breaks the above rule or not:

Proposition 2. *Any cell subdivision outside "Modification zone" never violates Proposition*1 (*see proof in Appendix* [34]).

After detecting the potential violation, we can properly handle it using the following proposition:

Proposition 3. *If we subdivide all boundary cells around merging region at the same time, the new "To-be-merged" control points still guarantee "Boundary requirement" and their weights all equal to one* (*see proof in Appendix* [34]).

Based on the above propositions, we propose our modification procedure as follows. The input is the newly refined control grid with new subdivided cell set $\mathbf{C}_{new}$.

Step 1: For each boundary, assign the cell set $\mathbf{C}_T$ as "Modification zone". For any cell with one vertex as a "To-be-merged" control point, we add this cell into $\mathbf{C}_T$.

Step 2: For each boundary, detect if there is any new subdivided cell in the "Modification zone":

- $\mathbf{C}_{new} \bigcap \mathbf{C}_T = \emptyset$. According to Proposition 2, the refined grid preserves the standardness on the boundary, so no further processing.
- $\mathbf{C}_{new} \bigcap \mathbf{C}_T \neq \emptyset$. Modify the boundary according to Proposition 3: Subdivide all cells on the boundary to satisfy "Boundary requirement".

Step 3: Update control point positions. Instead of fitting again like in Section 4.3, we use the same method as in [14] because we seek to keep spline shape unchanged in this step.

5. Global merging strategies

In our framework, the decomposed components can be merged in various different merging types. Here we develop algorithms to handle different types of merging. As we discussed in Section 3.1, our domain only includes "Two-cube" merging (Section 5.1) and "Type-1" merging (Section 5.2). Also, we seek to handle more complicated conventional poly-cube domains, including all other types of merging in Fig. 2 (Section 5.3).

5.1. "Two-cube" merging

Merging of 3D components can be simply illustrated by 1D merging. In 1D merging, each boundary parameter corresponds to a new position after merging. For example, in Fig. 11, the bd-control-point with parameter 5 corresponds to a new parameter 6. The control point corresponding to $n(n \geq 2)$ original control points simply takes the average position as its new position. Similarly, the merging of two cuboids includes the following steps.

Step 1: Boundary modification. If bd-knot intervals of two components are different, subdivide the cube boundary using the procedure in Section 4.4 iteratively until they share the same knot interval (Fig. 13(a)). *Step* 2: Merging control points. Correspond the original control point to the new control grid. As shown in Fig. 13(a)Right, we merge each column along the merging direction as 1D case. *Step* 3. Computing control point positions. Each new control point $\mathbf{p}'$ corresponds to $n(n \leq 2)$ original control points $\mathbf{p}_i$. The new control point position is computed by $\mathbf{p}' = \sum_i^n \mathbf{p}_i/n$.

5.2. "Type-1" merging

The goal of is to merge three cuboids into one control grid, like Fig. 2(a). We can still use the "Two-cube" merging technique to treat most merging regions. But we have to design special confinement method to handle the central points on the yellow dot/lines. Fig. 14(b) shows the extra bd-control-points we add around the central point on the yellow lines. For the yellow dot, we add additional eight bd-control-points around it to confine it into the surface boundary, as shown in Fig. 14(c).

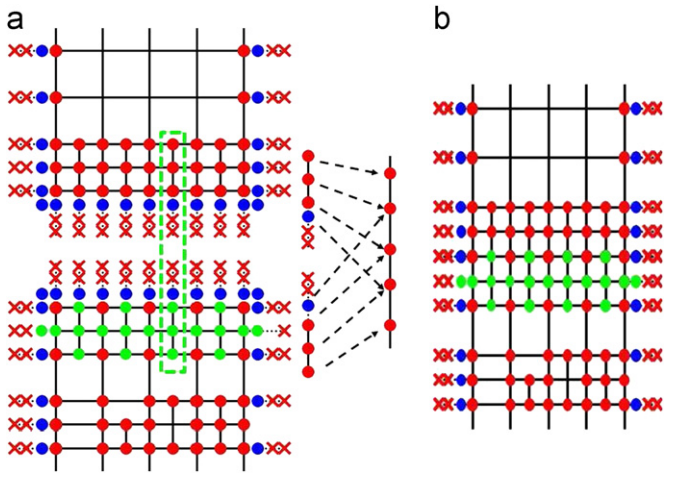

Fig. 13. "Two-cube" merging. (a) Left: Subdivide the bottom cuboid and insert new control points (green dots) to keep the same knot intervals. Right: Merge along the merging direction. (b) The merged control grid. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

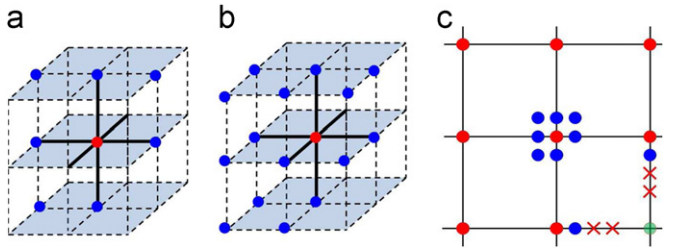

Fig. 14. "Type-1" merging: (a-b) The 3-D distribution of bd-control-points around the central point on the yellow lines/dot in Fig. 2(a). (c) To preserve semi-standardness, bd-knots (blue dots and red crosses) and an auxiliary knot (green) are added. Then we can use local refinement algorithm to compute new control points' weights. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 15 illustrates the confinement effect. Fig. 15(a) shows a confined 2D control grid in 2D layout. The extra bd-control-points (blue dots) are inserted around the central point. Fig. 15(b–d) showcase its advantage: unlike Fig. 8, for any chosen parametric position, none of its control points penetrates the boundary to influence the chosen position.

Preserving semi-standardness: Now we still have another challenge. Simply adding these extra control points would violate the semi-standardness property. To preserve semi-standardness, we also modify weights in this newly merged control grid structure. The weight can be computed as follows (see Fig. 14(a)): (1) before adding bd-knots around the central point, we add an auxiliary control point (green dot) at the corner. Now we locally have a standard rectangular control grid with weights all equal to one initially; (2) Insert the designed bd-knots (blue knots and red crosses in Fig. 14(a)) to the grid; (3) Inserting knots triggers the local refinement procedure to recompute the weight of each control point. Note that after refinement, the auxiliary point does not affect regions inside boundary anymore. Therefore, it is "transparent" and free to be deleted from the spline representation.

Besides preserving semi-standardness, our weight modification technique also has advantage for pre-computation. The weight computation only depends on the initial knot interval of merged control grid. Thus, we can pre-compute this step and build a look-up table for speedup. Table 2("Indices") shows the indices of control points around the central point (the same as

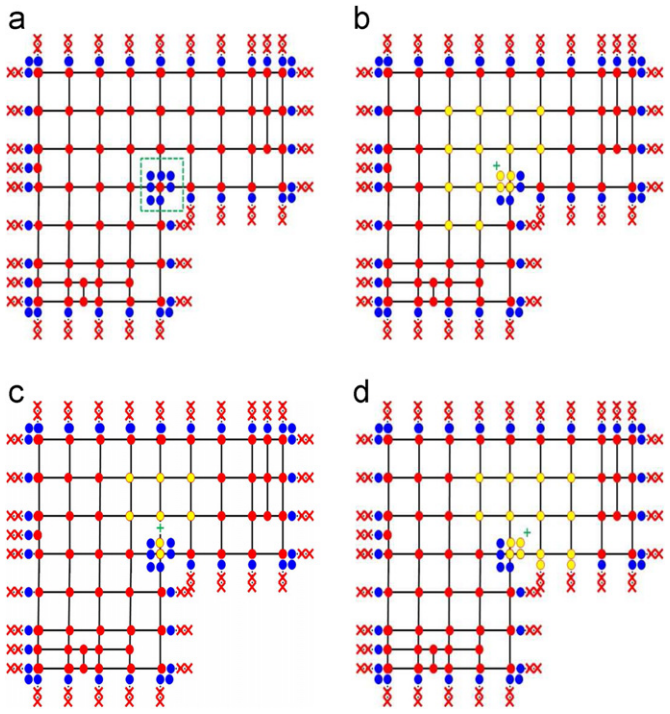

Fig. 15. Confinement effect of "Type-1" merging. (a) The 2D layout of a refined control grid, with added bd-control-points (blue dots) around the central point (green box). (b-d) For each parameters (green cross), we highlight all control points (yellow points) that influence this parameter. The violation like Fig. 8 is completely eliminated. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 2
Look-up tables. Row 1: an index table for 27 possible br-control-points in Fig. 9. Row 2: weights for "Type-1" merging in Fig. 15(b) (weights in parentheses correspond to additional eight control points in Fig. 15(c)). Row (3–5): weights for "Type-2,3,4" merging.

Indices								
7	8	9	16	17	18	25	26	27
4	5	6	13	14	15	22	23	24
1	2	3	10	11	12	19	20	21
Type-1								
–	–	–	–	–	–	–	–	–
1	1	–	$\frac{17}{18}$	$\frac{35}{36}$	1	$\frac{8}{9}$	$\frac{17}{18}$	1
(1)	(1)	–	$\left(\frac{17}{18}\right)$	$\left(\frac{35}{36}\right)$	(1)	$\left(\frac{8}{9}\right)$	$\left(\frac{17}{18}\right)$	(1)
Type-2								
$\frac{26}{27}$	$\frac{53}{54}$	1	$\frac{53}{54}$	$\frac{107}{108}$	1	1	1	–
$\frac{53}{54}$	$\frac{107}{108}$	1	$\frac{107}{108}$	$\frac{209}{216}$	$\frac{17}{18}$	1	1	–
1	1	1	1	$\frac{17}{18}$	$\frac{8}{9}$	–	–	–
Type-3								
$\frac{20}{27}$	$\frac{22}{27}$	$\frac{8}{9}$	$\frac{22}{27}$	$\frac{95}{108}$	$\frac{17}{18}$	$\frac{8}{9}$	$\frac{17}{18}$	1
$\frac{22}{27}$	$\frac{95}{108}$	$\frac{17}{18}$	$\frac{95}{108}$	$\frac{25}{27}$	$\frac{35}{36}$	$\frac{17}{18}$	$\frac{35}{36}$	1
$\frac{8}{9}$	$\frac{17}{18}$	1	$\frac{17}{18}$	$\frac{35}{36}$	1	1	1	–
Type-4								
$\frac{26}{27}$	$\frac{53}{54}$	1	$\frac{53}{54}$	$\frac{107}{108}$	1	1	1	–
$\frac{53}{54}$	$\frac{107}{108}$	1	$\frac{107}{108}$	$\frac{215}{216}$	1	1	1	–
1	1	–	1	1	1	–	–	–

indices in Fig. 9(b)). Table 2("Type-1") shows the corresponding weights for all control points in Fig. 14(a) (the numbers in parentheses correspond to additional control points in Fig. 14(b)).

To summarize, "Type-1" merging includes the following steps. The first three steps are the same as "Two-cube" merging. *Step* 1, modify boundary; *Step* 2, merge control points; *Step* 3, compute control point positions; *Step* 4, insert extra bd-control-points as shown in Fig. 14(a and b) (We assign the position of the control point on the central point to these new inserted control points); *Step* 5, modify weight (change the weight of these bd-control-points by checking the look-up table (Table 2("Indices"))).

5.3. "Type-2,3,4" merging

The above two merging algorithms (in Sections 5.1 and 5.2) are already functionally sound when handling the merging of all components in our divide-and-conquer framework, because these are the only two merging types in our T-shape based poly-cube. Without being limited to just that, Our ambitious goal is to handle any shape of poly-cube domains. We offer several more powerful merging operations, which are designed to merge the components like "Type-2,3,4" in Fig. 2(b–d). Once again, in order to enforce the boundary restriction, we need to insert extra bd-control-points. For the central points on all yellow lines in Fig. 2(b–d), they are just "Type-1" merging, so we use the same merging method as shown Fig. 14(a). For the central points on 3 yellow dots, we design the extra bd-control-points, as shown in Fig. 16, to preserve boundary restriction.

To guarantee semi-standardness, we recompute the weight using the same method in Section 5.2 as follows. First, we add auxiliary control points, expanding given control grid around the central point to a complete cube-like grid. Second, we insert the designed bd-control-points and perform local refinement to compute the new weight for each control point. Their look-up tables are documented in Table 2.

6. Implementation issues and experimental results

Our experimental results are implemented on a 3 GHz Pentium-IV PC with 4 Giga RAM. Our first experimental results (Figs. 17 and 18) show the application of "Two-cube" merging by considering the kitten and Beethoven model as the datasets. These are the only merging types that exist in our component generation framework. For "Type-2,3,4" merging types that do not exist in our framework, we design a special screw driver model and domain to demonstrate the power of "Type-2" merging (Fig. 19). In terms of poly-cube construction, we recognize that "Type-2" merging is very popular to handle the input with long branches. Yet, "Type-3,4" merging cases rarely exist even in the most conventional poly-cube domains. Geometrically speaking, they are more suitable to mimic highly concave shapes. We use the dark T-junction lines to show control grid knots and use

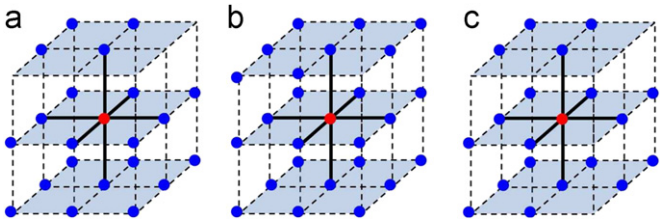

Fig. 16. The 3D distribution of bd-control-points in "Type-2,3,4" merging. The central points are on the yellow dots in Fig. 2(b–d). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

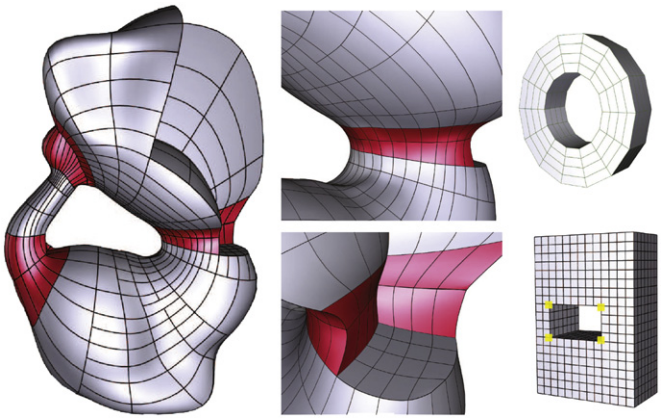

Fig. 17. "Two-cube" merging for the kitten model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

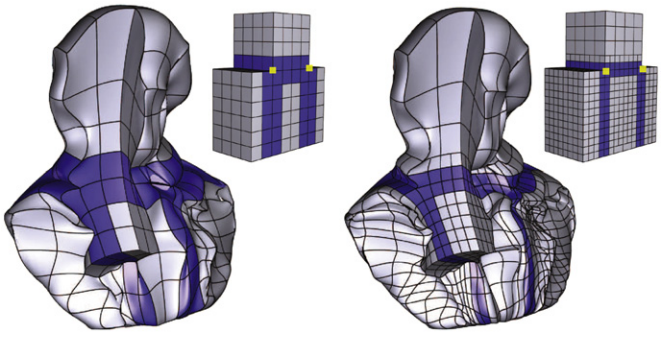

Fig. 18. "Type-1" merging for the Beethoven model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

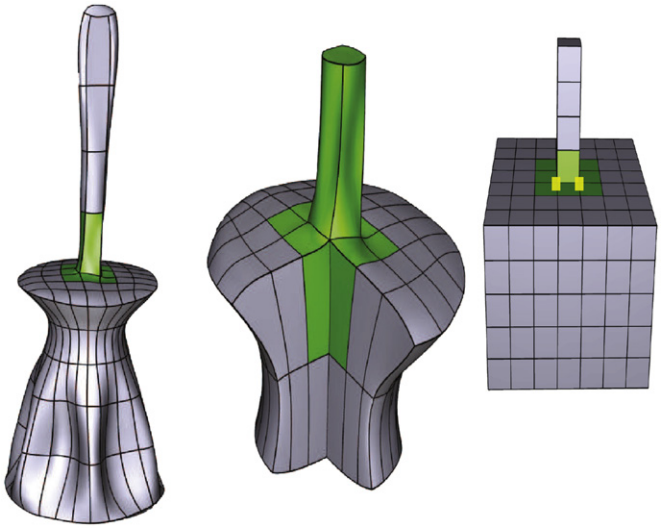

Fig. 19. "Type-2" merging for the screw driver model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

different colors to represent different merging types. Red/Blue/Yellow marks all "to-be-merged" control point knots in three merging cases, respectively. We also have a close-up view to show the interior fitting result, demonstrating smoothness around the

merging region. The yellow marks on the control grid highlight the ill-points.

In the second group of experimental results (Figs. 4, 21–23), we integrate all merging types together to handle the models with high-genus and complex bifurcations, including the eight (genus 2), g3 (genus 3), rockarm, and wrench (genus 1 with bifurcations) models. We first display their component generation results. Then we show a spline model for one local component and the final spline results with a close-up view to highlight the interior fitting and merging regions. Fig. 20 also visualizes components' T-shape/poly-cube structures in a more efficient way. We use the same color cuboid to represent one component and the edges to show the cuboid connections. Each green box covers cuboids from the same T-shape. This structure clearly demonstrates that only "Two-cube" and "Type-1" merging are functionally sound in our framework.

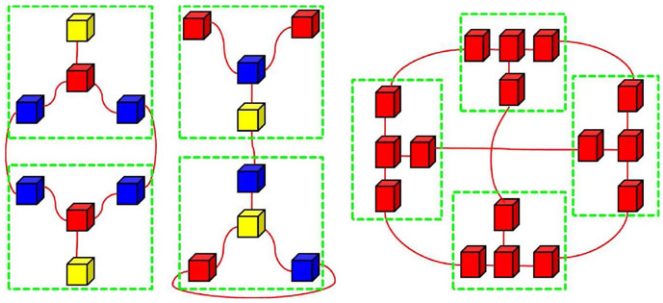

Fig. 20. The divide-and-conquer structures of the rockarm/wrench/g3 model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

In Table 3, we document numbers of control points and fitting error. T-spline can significantly reduce the number of control points. The fitting results are measured by RMS ("root-mean-square") errors which are normalized to the dimension of corresponding solid models. Meanwhile, we demonstrate the interior fitting quality in a close-up view of each model. Also, the table illustrates that adaptive refinement is necessary for trivariate splines, even on a simple surface input model. It is desirable to use high resolution with more DOFs to approximate boundary surface and low resolution with fewer DOFs for volume interior. For example, in the kitten/Beethoven model, if we naively use the B-spline scheme with hierarchical refinement, their control points will increase to 3718/4850, respectively. In the last experiment (Fig. 24), we apply our technique to convert the fertility model, with the noisy surface, into a trivariate spline and remesh it into a smooth result. The poly-edges (gray-lines) decompose the fertility model into components. Note that poly-edges are aligned everywhere so our local parameters are consistent globally.

Away from extraordinary points, our spline achieves C^2 everywhere, except that duplicate knots on the boundary will lower the continuity order. On the boundary, we have duplicate knots along one iso-parametric direction. These knots lower the continuity order along its own iso-parametric direction only but we still have C^2 along the other two iso-parametric directions for tensor-

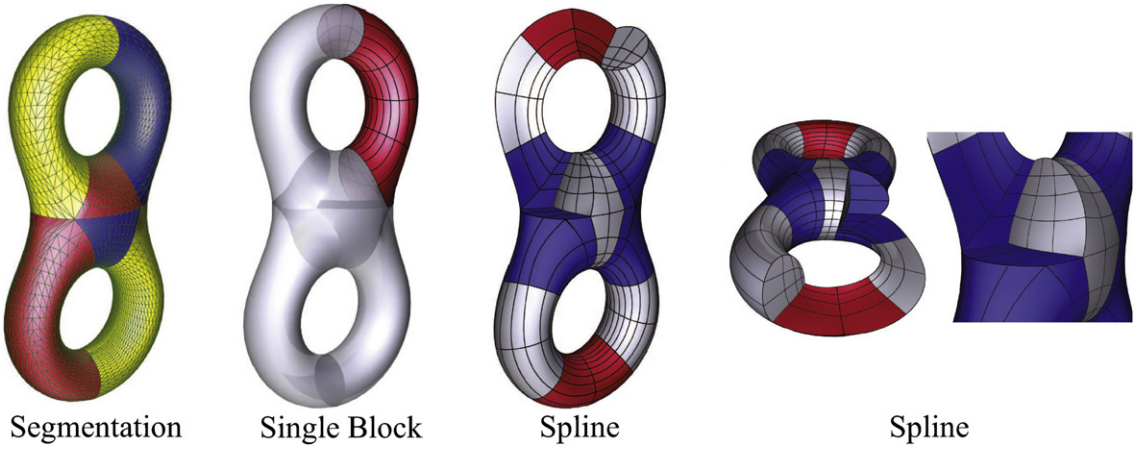

Fig. 21. The eight model.

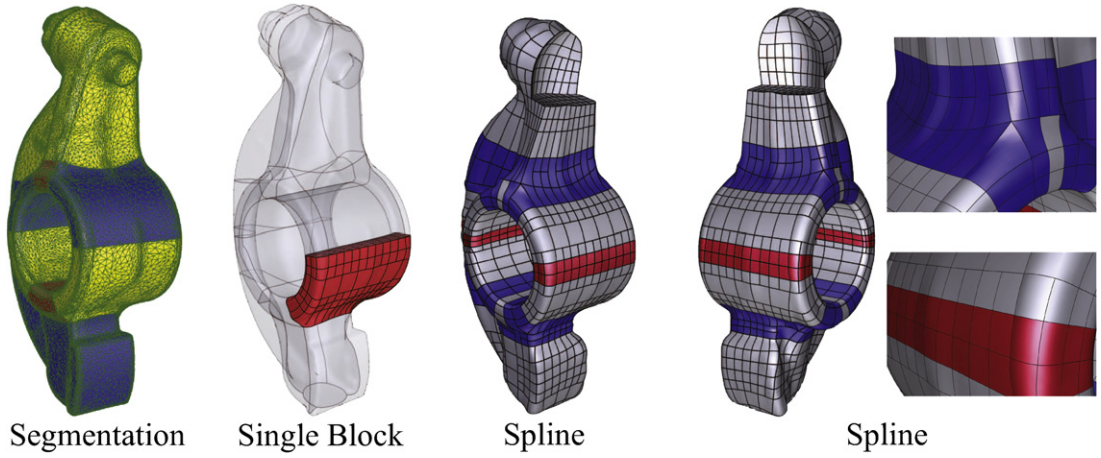

Fig. 22. The rockarm model.

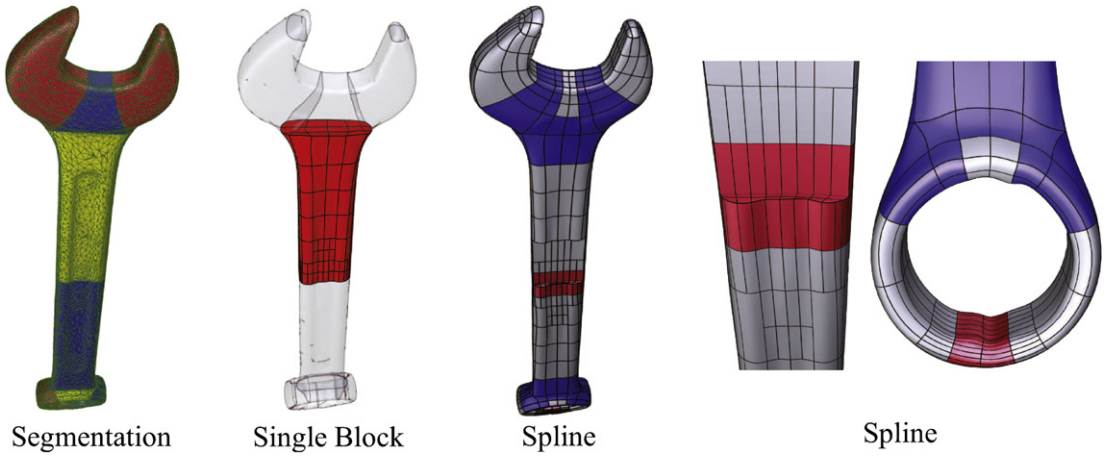

Fig. 23. The wrench model.

Table 3
Statistics of various test examples: N_c, # of control points; RMS, root-mean-square fitting error (10^{-3}). "bv1", "bv2", "ra" and "sd" represent the Beethoven (low and high resolution), rockarm, and screwdriver models.

Model	N_c	RMS	Model	N_c	RMS
Eight	2058	1.63	Wrench	3756	2.3
Kitten	2840	3.32	g3	2976	1.74
bv1	1001	1.8	bv2	3273	1.36
ra	4582	3.75	sd	1261	1.65

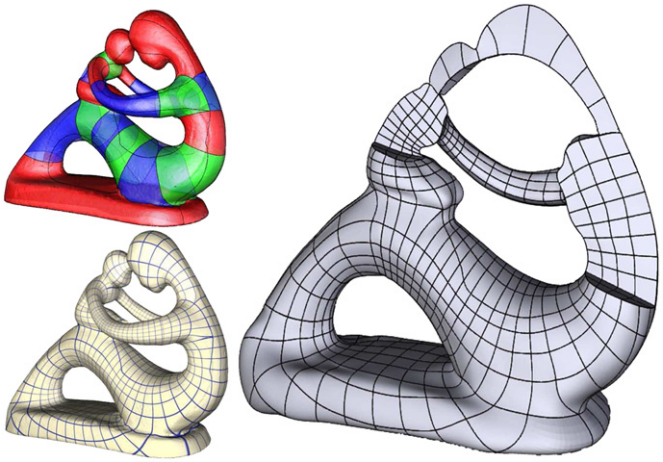

Fig. 24. Mesh smoothing: We convert the fertility model to a trivariate spline model and remesh it into a smooth result. Three figures show the components (with poly-edges), the globally aligned parameters, and the remeshing result (with the interior cutout view), respectively.

product construction. Furthermore, in order to preserve boundary restriction, we have to add duplicate knots around the corner and sacrifice smoothness only on the corner. For "Type-1" merging, we have to add two extra duplicate knots along two directions, leading to C^0 on this corner. Yet we still keep C^2 along the third direction. For "Type-2,3,4" we add two extra knots along all directions so we get C^0 on the corner. Away from these corners, our volume are C^2 everywhere.

Top-down vs. Divide-and-conquer schemes: In Table 4, we compare the performance between our divide-and-conquer framework with general T-splines using single integral domain in a traditional top-down approach. The most prestigious advantage of our divide-and-conquer framework is to easily handle models with bifurcations/highly twisted-shape/high-genus. For example,

Table 4
Comparison between our splines and general splines: space required by fitting; time to compute derivatives of basis functions; N_c, and number of cuboids.

Model	Our method			General method		
	Space	B' (s)	N_c	Space	B' (s)	N_c
Kitten	116 802	2.38	1	300 688	4.53	8
Eight	24 714	2.25	6	174 124	4.35	15
g3	18 952	2.17	16	314 832	4.23	46

a poly-cube like Fig. 1 designed using a top-down scheme is very complicated, with 46 cuboids and they are connected in various types, to mimic the shape of the g3 model. The poly-cube construction also requires tedious manual design. By comparison, its divide-and-conquer domain (Fig. 20) includes only 16 cuboids with two certain merging types. Second, we also compare the required spatial consumption between our divide-and-conquer scheme and the top-down scheme. In general, our memory cost is reduced to $1/n_s$, where n_s is the number of cuboids. Third, we compare the computation of B' between semi-standard T-splines and rational T-splines. We record the computation time on 10^4 samples for each model. The result shows that our method is at least twice as fast as rational T-splines. This is because the computation avoids division operation completely (see the difference between Eqs. (1) and (3)).

7. Conclusion and future work

In this paper, we have presented a novel framework to construct trivariate splines with arbitrary topology. Because of the divide-and-conquer scheme, our framework can naturally handle solid objects with high genus and complex bifurcations. We decompose the input surface model into several part-aware components so that we can fit each component. The proposed spline scheme supports local refinement and the global trivariate T-splines satisfy the attractive properties of semi-standardness and boundary restriction. These novelties have a broad appeal to both theoreticians and engineers working in the shape modeling and its application areas.

These modeling advantages naturally prompt us to explore its unchartered potential in the context of isogeometric analysis for computer-integrated engineering. Since our framework supports a regular spline structure, GPU-enabled scientific computing and image-driven shape processing become more desirable towards the design of more efficient algorithms. This framework can also be generalized by increasing the dimension of control points to

model not only geometry but also other physical attributes simultaneously.

Acknowledgments

This research is supported in part by US National Science Foundation Grants IIS-0949467, IIS-1047715, and IIS-1049448.

Appendix A. Supplementary material

Supplementary data associated with this article can be found in the online version at http://dx.doi.org.10.1016/j.cag.2012.03.007.

References

[1] Wang K, Li X, Li B, Xu H, Qin H. Restricted trivariate polycube splines for volumetric data modeling. IEEE Trans Vis Comput Graph 2012;18(5):703–16.
[2] Li B, Li X, Wang K, Qin H. Generalized polycube splines. In: SMI10: international conference on shape modeling and applications; 2010. p. 261–6.
[3] Myles A, Yeo Y, Peters J. GPU conversion of quad meshes to smooth surfaces. In: SPM'08: proceedings of the 2008 ACM symposium on solid and physical modeling; 2008. p. 321–6.
[4] Song W, Yang X. Free-form deformation with weighted t-spline. Vis Comput 2005;21(3):139–51.
[5] Hughes T, Cottrell J, Bazilev Y. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Comput Methods Appl Mech Eng 2005;194:4135–95.
[6] Rössl C, Zeilfelder F, Nurnberger G, Seidel H. Reconstruction of volume data with quadratic super splines. IEEE Trans Vis Comput Graph 2003;10(4): 397–409.
[7] Li W, Ray N, Lévy B. Automatic and interactive mesh to t-spline conversion. In: Symposium on geometry processing; 2006. p. 191–200.
[8] Wang H, He Y, Li X, Gu X, Qin H. Polycube splines. Comput Aided Des 2008;40(6):721–33.
[9] Zhou X, Lu J. NURBS-based Galerkin method and application to skeletal muscle modeling. In: ACM solid and physical modeling symposium; 2005. p. 71–8.
[10] Martin T, Cohen E, Kirby R. Volumetric parameterization and trivariate B-spline fitting using harmonic functions. Comput Aided Geometric Des 2009;26(6):648–64.
[11] Zhang Y, Bazilevs Y, Goswami S, Bajaj CL, Hughes T. Patient-specific vascular NURBS modeling for isogeometric analysis of blood flow. Comput Methods Appl Mech Eng 2007;196(29–30):2943–59.
[12] Martin T, Cohen E. Volumetric parameterizations of complex objects by respecting multiple materials. Comput Graph 2010;34(3):187–97.
[13] Sederberg T, Zheng J, Bakenov A, Nasri A. T-splines and t-NURCCs. ACM Trans Graph 2003;22(3):477–84.
[14] Sederberg T, Cardon D, Finnigan G, North N, Zheng J, Lyche T. T-spline simplification and local refinement. ACM Trans Graph 2004;23(3):276–83.
[15] Sederberg T, Zheng J, Sewell D, Sabin M. Non-uniform recursive subdivision surfaces. In: SIGGRAPH98; 1998. p. 387–94.
[16] Ipson H. T-spline merging, Master's thesis, Brigham Yong University; 2005.
[17] Sederberg T, Finnigan G, Li X, Lin H, Ipson H. Watertight trimmed NURBS. ACM Trans Graph 2008;27(3):1–8.
[18] Myles A, Pietroni N, Kovacs D, Zorin D. Feature-aligned t-meshes. ACM Trans Graph 2010;29(4):117:1–117:11.
[19] Loop C, Schaefer S. G2 tensor product splines over extraordinary vertices. In: Proceedings of the symposium on geometry processing, SGP '08; 2008. p. 1373–82.
[20] Peters J, Fan J. The projective linear transition map for constructing smooth surfaces. In: Proceedings of the 2010 shape modeling international conference, SMI '10; 2010. p. 124–30.
[21] Tarini M, Hormann K, Cignoni P, Montani C. Polycube maps. ACM Trans Graph 2004;23(3):853–60.
[22] Lin J, Jin X, Fan Z, Wang C. Automatic polycube-maps. In: Geometric modeling and processing; 2008. p. 3–16.
[23] Gregson J, Sheffer A, Zhang E. All-hex mesh generation via volumetric polycube deformation. Comput Graph Forum 2011;30(5):1407–16.
[24] Alliez P, Cohen-Steiner D, Yvinec M, Desbrun M. Variational tetrahedral meshing. ACM Trans Graph 2005;24:617–25.
[25] Yan D, Wang W, Lévy B, Liu Y. Efficient computation of clipped voronoi diagram. Comput Aided Des. http://dx.doi.org/10.1016/j.cad.2011.09.004, in press.
[26] Wang Y, Gu X, Chan T, Thompson P, Yau S. Volumetric harmonic brain mapping. In: IEEE symposium on biomedical imaging: macro to nano; 2004. p. 1275–8.
[27] Xia J, He Y, Yin X, Huan S, Gu X. Direct-product volumetric parametrization of handlebodies via harmonic fields. In: Shape modeling international conference; 2010. p. 3–12.
[28] Shapira L, Shamir A, Cohen-Or D. Consistent mesh partitioning and skeletonisation using the shape diameter function. Vis Comput 2008;24:249–59.
[29] Dey T, Li K, Sun J. On computing handle and tunnel loops. In: International conference on cyber worlds; 2007. p. 357–66.
[30] Li X, Gu X, Qin H. Surface mapping using consistent pants decomposition. IEEE Trans Vis Comput Graph 2009;15(4):558–71.
[31] Taubin G. A signal processing approach to fair surface design. In: SIGGRAPH '95: proceedings of the 22nd annual conference on computer graphics and interactive techniques; 1995. p. 351–8.
[32] Floater M. Mean value coordinates. Comput Aided Geometric Des 2003;20(1):19–27.
[33] Zhou K, Huang J, Snyder J, Liu X, Bao H, Guo B, et al. Large mesh deformation using the volumetric graph Laplacian. ACM Trans Graph 2005;24(3): 496–503.
[34] Electronic supplement 〈http://www.cs.stonybrook.edu/∼bli/smi12appendix〉.