Surface Mesh to Volumetric Spline Conversion with Generalized Polycubes

Bo Li, Xin Li, *Member*, *IEEE*, Kexiang Wang, and Hong Qin

Abstract—This paper develops a novel volumetric parameterization and spline construction framework, which is an effective modeling tool for converting surface meshes to volumetric splines. Our new splines are defined upon a novel parametric domain called generalized polycubes (GPCs). A GPC comprises a set of regular cube domains topologically glued together. Compared with conventional polycubes (CPCs), the GPC is much more powerful and flexible and has improved numerical accuracy and computational efficiency when serving as a parametric domain. We design an automatic algorithm to construct the GPC domain while also permitting the user to improve shape abstraction via interactive intervention. We then parameterize the input model on the GPC domain. Finally, we devise a new volumetric spline scheme based on this seamless volumetric parameterization. With a hierarchical fitting scheme, the proposed splines can fit data accurately using reduced number of superfluous control points. Our volumetric modeling scheme has great potential in shape modeling, engineering analysis, and reverse engineering applications.

Index Terms—Volumetric spline, generalized polycube, volumetric parameterization

1 Introduction and Motivation

The rapid advances in 3D scanning and acquisition techniques have given rise to the explosive increase of volumetric digital models in CAD environments in recent years. The engineering design industry frequently pursues data transformation from discrete 3D data to spline formulations because of their compactness and continuous representation. Compared with the commonly-used *"surface model to surface spline"* paradigm, volumetric splines can represent both boundary geometry and real volumetric and physical/material attributes. This property makes volumetric representation highly preferable in many physically based applications including mechanical analysis [1], shape deformation and editing, virtual surgery training, and so on. However, converting arbitrary meshes to volumetric splines is very challenging due to many unique requirements in parametric domain construction. These requirements, however, have not been thoroughly discussed and enforced in existing parameterization and spline construction techniques:

1. *Structural regularity.* Tensor-product splines (e.g., NURBS) are defined over regular "cube-like" domains. Compared with the unstructured domain (e.g., polygonal regions covered by tetrahedral meshes), regular domain supports more efficient evaluation and refinement, and GPU acceleration can also be applied directly to spline representation with regular structure. Also, spline-based physical analysis (e.g., isogeometric analysis [1]) has a preference for "cube-shaped" domain.
2. *Singularity free.* Singularity here means an inability to produce a locally consistent parameterization in the neighborhood. Specially in trivariate splines, a global volumetric model is locally parameterized onto several tensor-product charts. Like Figs. 1a and 1b, a singular point locates where local charts merge, if its valence number along one isoparametric plane is larger than four (note that from this definition, singularity in volumetric domain is of difference from surface geometry). Handling singularity with tensor-product splines is very challenging. Therefore, it is desirable to have a global one-piece spline defined on a globally connected singularity-free domain.
3. *Controllable Ill points.* In a volumetric parameterization over the polycube domain, we call the corner point in a concave corner of the polycube an ill point. On such a point, the basis function spans across nearby cubes through outside space (see Figs. 1c and 1d). Figs. 1e, 1f, 1g, and 1h illustrate all possible types of ill points in red (note that they are not singularities in volumetric parameterization but singularities in surface parameterization). Being harmless to usual parameterization-related applications, ill points, however, have an undesirable side effect on spline construction and subsequent tasks like physical analysis, boundary confinement, and partition-of-unity control (see [2], [3] for more details). Therefore, it is desirable to control the number and types of ill points. In practice, we hope to restrict ill points to "Type-1" only, as shown in Fig. 1e, since it is the easiest type and we can simply modify and restrict its "boundary" basis function [3].
4. *Shape awareness.* Each spline patch should abstract the shape in a geometrically meaningful way, reveal

• *B. Li, K. Wang, and H. Qin are with the Department of Computer Science, Stony Brook University, Stony Brook, NY 11794. E-mail: {bli, kwang, qin}@cs.stonybrook.edu.*
• *X. Li is with School of Electrical Engineering & Computer Science and Center for Computation and Technology, Louisiana State University, Baton Rouge, LA 70803. E-mail: xinli@cct.lsu.edu.*

Manuscript received 9 Nov. 2011; revised 31 May 2012; accepted 10 Aug. 2012; published online 24 Aug. 2012.
Recommended for acceptance by W. Wang.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2011-11-0279.
Digital Object Identifier no. 10.1109/TVCG.2012.177.

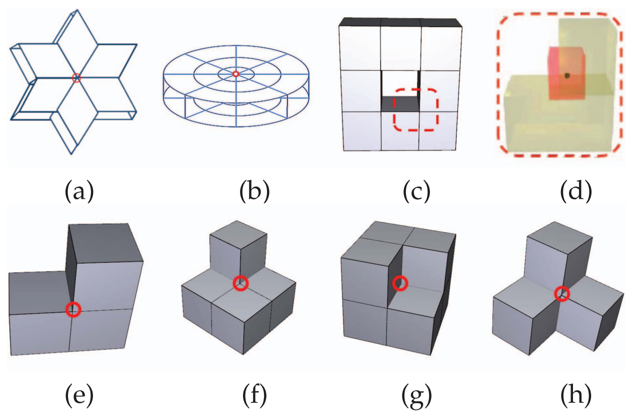


Fig. 1. Singularity and ill-point distribution in the volumetric domain is very critical to spline construction. (a-b) show two cases of singular points. (c) highlights one ill point. (d)This shows that the basis function around the ill point has influence outside the cube boundary. (e-h) show different types of ill-points: "Type-1" to "Type-4," which are the concave points in the domain.

the shape's key perceptual parts and topological structures (e.g., skeleton-like representation). Most importantly, spline construction on large volume data heavily depends on spline gluing in practice. Therefore, one desirable parameterization scheme should try to reduce patch number to cut off spline gluing processing.

Existing volumetric spline techniques generally follow two different trends: 1) Many recent methods [4], [5], [6] convert each part into splines defined on a cylinder/tube domain (e.g., Fig. 1b), because they can intuitively use the shape skeleton to produce a tube domain and reveal the global structure and topology. A severe limitation of such approaches is that points on the tube centerline are all singular. 2) In contrast, poly-cube splines [7], [8] are defined on domains assembled by multiple cubes, which avoid the central line singularity problem. Such splines are flexible to resemble the shape of the given mesh and are capable of capturing the large scale features with low-distortion mapping. However, gluing of many cubes may produce many uncontrollable ill points. Limitations from both categories of splines have inspired us to develop a new method that is superior to both types of splines.

The main contributions of this work are as follows: 1) We propose a novel concept of *Generalized polycube (GPC)* to serve as the parametric domain for spline construction. Particularly, GPC combines advantages of existing primitives to support splines: (a) GPC is powerful and flexible for representing complex models; (b) GPC provides a simple and regular domain with no singularity and controllable ill-point numbers/types, yet very spline-friendly domain structure. 2) We develop an effective GPC construction and parameterization framework to achieve all the above goals, while still respecting both the global structure and the geometric features. 3) We present a global "one-piece" volumetric spline scheme without stitching/trimming for general volumetric models. Unlike conventional spline schemes, our conversion does not require global coordinates everywhere, and piecewise local coordinates suffice. GPC, therefore, becomes an ideal parametric domain. We also design an efficient volumetric hierarchical spline fitting algorithm to support recursive refinement with improved accuracy and reduced number of control points.

2 Overview and Background

Our algorithmic pipeline mainly includes three steps: shape decomposition and abstraction (Section 4), volumetric parameterization (Section 5), and spline approximation (Section 6). In this section, we briefly review relevant background and existing techniques.

2.1 Shape Decomposition and Abstraction

As a thorough discussion, we refer readers to Shamir's great survey [9] on general segmentation research. To achieve part-aware decomposition, many methods have attempted to design appropriate part-aware metrics to align with human visual perception toward algorithm development, including shape diameter function [10], visual region difference [11], intrinsic symmetry [12], and so on. However, these algorithms are not appropriate for downstream spline construction without considering issues like patch numbers, singularity, and ill points.

A popular shape abstraction method is to use polycube [13]. In [13], the domain construction and mapping are computed through simple projection. Wang et al. [7] introduced an intrinsic approach that first maps the model and the polycube to a common canonical domain to guarantee bijectivity. Several methods have been developed to improve user control. Wang et al. [14] presented a technique to adjust locations and the number of corners of the polycube map. Xia et al. [15] allowed users to sketch curve constraints to control the polycube map. Automatic polycube construction is always very difficult. Lin et al. [16] used the Reeb graph to segment the surface and construct polycube map. However, their segmentation method may not work for shapes with complicated topology and geometry and does not guarantee bijectivity. He et al. [8] proposed a divide-and-conquer algorithm by slicing the model along an axis direction. Slicing along an axis produces very complex domain structure so Gregson et al. [17] proposed a deformation-based method, which is less prone to oversegmentation. These methods work best for axis-aligned geometric models without any twist, bend, and spiral.

2.2 Volumetric Parameterization

Shape parameterization is widely studied and we refer readers to comprehensive survey reports of [18], [19], and [20] for various surface parameterization techniques. More closely related to our work, volumetric parameterization has gained great interest in recent years. A few related techniques have been developed toward various applications such as shape registration [21], [22], volume deformation [23], [24], [25], and spline construction [5]. Wang et al. [21] parameterized solid shapes over a solid sphere by a variational finite element algorithm. This technique is utilized for constructing a shell mesh of the poly-cube domain [26]. However, this work only focuses on sphere-like solid shapes such as human brain. Ju et al. [23]

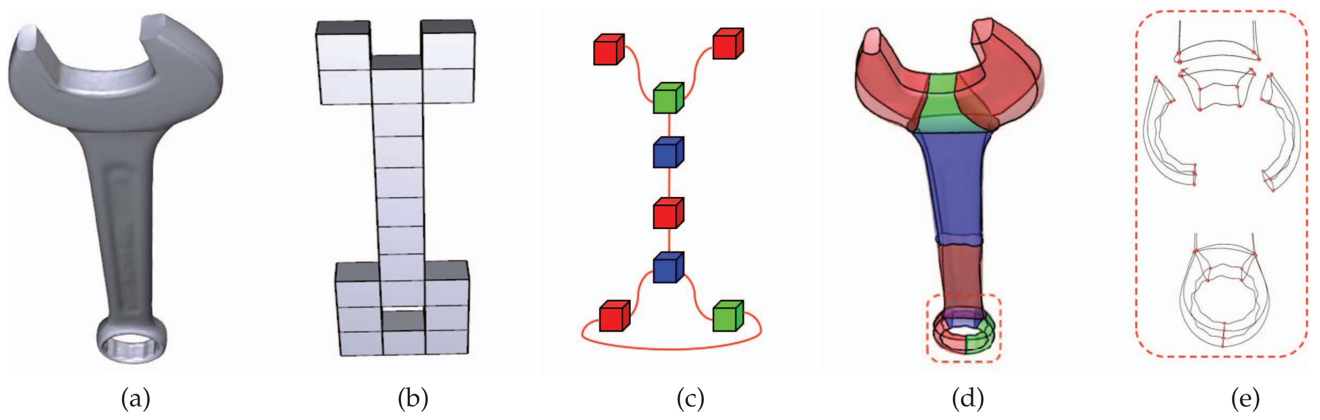


Fig. 2. GPCs: (a) The wrench model; (b) The CPC; (c) The GPC as a topological graph; (d-e) The cuboid edges are overlaid onto the model to visualize the GPC global structure.

generalized the mean value coordinates [27] from surface to volume for a smooth volumetric interpolation. Joshi et al. [24] presented harmonic coordinates for volumetric interpolation and deformation purposes. Their method guarantees the nonnegative weights and therefore leads to a more pleasing interpolation result in concave regions compared with that in [23]. Martin et al. [5] computed the precise (u, v, w) coordinates for genus-zero tetrahedral meshes, and the target domain is a cylinder. Li et al. [22] used the fundamental solution method to map solid shape onto a general target domain. Xia et al. [28] used harmonic field to map the solid shape onto a polycube domain. Nieser et al. [29] proposed the technique to generate volumetric mapping and hexahedral mesh, which are guided by a frame field.

2.3 Volumetric Splines

Instead of transforming a surface model to a surface spline [7], we seek for converting the whole volume space into a volumetric spline. Compared with simplex splines (e.g., [30], [31]), splines defined on regular tensor-product domain are more popular and widely used because of simple and regular structures. Song et al. [32] employed volumetric splines with nonuniform weights to model free-form deformation. A modeling technique [33] was developed to model skeletal muscle with anisotropic attributes and conduct FEM analysis directly on NURBS solid [1]. One challenge is to convert complex geometry models into regular volumetric splines. Zhang et al. [6] proposed a method to handle long branches. Their algorithm divides possible bifurcations of a vascular system into different cases to solve. Martin and Cohen [4] used a "mid-surface" in conjunction with harmonic functions to decompose the object. Yet, these methods cannot eliminate singularities.

Another challenge for volumetric B-splines is the huge number of control points for accurate geometry fitting. T-splines were introduced by Sederberg et al. [34] as a generalization of the traditional nonuniform B-spline surfaces. They also developed an algorithm to convert NURBS surfaces into T-splines while eliminating superfluous control points [35]. Zheng et al. [36] developed a technique for adaptively fitting T-splines to functional data.

3 Generalized Polycubes (GPCs)

Conventional polycube (CPC) is a shape composed of axis-aligned unit cubes that abut with each other. Cubes are glued and realized in a global 3D world coordinate system. CPC usually uses unit cubes as the building block. All cubes are glued together and embedded in the 3D space; any point in a cube is associated with a unique global coordinate. Fig. 2b shows an example of CPC constructed for a wrench model in Fig. 2a. Constructing effective (good approximation, coherent topology) CPC for volumetric models with relatively complicated geometry and topology usually requires extensive user involvement. Such a parametric domain is inadequate. A less tedious domain construction with reduced number of ill points is highly desirable.

GPC is composed of a set of cuboids glued together topologically. We allow any pair of two distinct cuboid faces to be glued together if these faces have the same size. Figs. 2c, 2d, and 2e show a GPC constructed for the wrench model (Fig. 2a).

From above definitions, GPC is less restrictive from CPC to be a better spline-friendly domain. First, GPC cuboid is not just a unit box. It can be a general cuboid with rectangular faces. Each cuboid has its local coordinate system; a cuboid is not axis-aligned but can deform (bend or twist) to glue with each other to form a global topological structure. Second, cuboids in GPC can be glued together through arbitrary two faces, and it is even possible that they are from the same cuboid. The topology of GPC can be represented using a topological graph, which we denote as a *GPC-graph* (each node represents a cuboid). Fig. 2c illustrates a GPC graph of Fig. 2d. To represent each cuboid, we project the 12 cuboid edges onto the model to visualize different faces (see Figs. 2d and 2e).

A less restrictive GPC has several advantages over CPC, which are very critical to trivariate spline construction: Controlled ill points, easier domain to simplify spline merging and more general shape modeling.

Ill-point controllability. First, the topological gluing can significantly reduce the number of ill points (due to the usage of fewer cuboids and simple gluing rules). In a simple shape like Figs. 3a and 3b, a torus' CPC generates 4 ill-points (in red circles) while a torus' GPC (see the kitten model, Fig. 14) has none. Second, our GPC construction algorithm

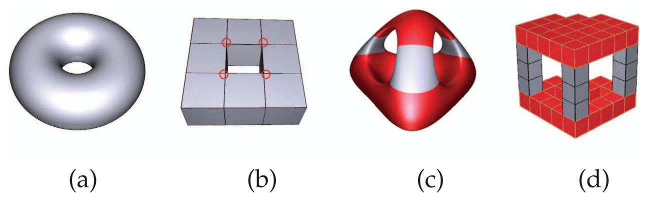


Fig. 3. (a) The torus model. (b) Its CPC uses at least eight cubes and generates four ill points. (c) The genus-3 model with narrow top and wide bottom regions. (d) Its CPC maps two regions onto the equal-sized parameterization domain, leading to large distortion.

will only generate *Type-1* (Fig. 1e) ill points. We can handle them much easier than other types of ill points [3].

Easier and better domain construction. Because of its topological simplicity and elegance, the construction of GPC is usually easier than that of CPC. Automatic GPC construction can be developed naturally following the part-aware decomposition of the model. From a spline practitioner's view, CPC requires many redundant cubes (to assemble topological handles in an axis-aligned way, like Fig. 3d). Cuboids in GPC are similar to the *"generalized cylinder"* so encodes the shape with less cuboids, which can significantly save the cost of spline merging.

When we consider parameterization distortion, less cuboids in GPC may lead to *less distortion* than CPC, because GPC is less restrictive (not axis aligned) and better mimics shape. For example, a CPC (Fig. 3d) can merely mimic the genus-3 model (with a narrow top and wide bottom region) in an axis-aligned domain. Consequently, two red-colored parts are parameterized onto the equally sized domain, introducing large distortion. A GPC (Fig. 5c) can fit the shape better and significantly improve the parameterization quality, benefitting the final spline construction.

Highly-twisted and high-genus shape. GPC can serve as the parametric domain for a more general category of solid shapes like the twisted or highly curved model, such as the twirl (Fig. 4a) and möbius band (Fig. 4d). Unlike axis-aligned CPC, GPC can twist them and glue adjacent cuboids in a topological way so that twisted global shape features can still be modeled as the cuboid edges (Figs. 4b and 4e), with a very small number of cuboids (Figs. 4 and 4f). For example, we can hardly construct a useful CPC domain for möbius band; But with GPC, only one cuboid is enough (Fig. 4f). Another category of models includes models with complex topology especially when handle loops/voids are relatively small, such as in the solid bucky model (Fig. 4g). For CPC, not only the above restrictive axis-aligned problem, small handles/voids also make the resulting CPC "overcomplex". A less restrictive GPC allows us to model the domain through a correct topological decomposition to small cuboids (Fig. 4h). The pattern of the bucky's GPC-graph around one handle can be decomposed as shown in (Fig. 4i).

The following three sections discuss the algorithmic pipeline to construct GPC and splines (also illustrated in Fig. 5). The input model is first decomposed into a few T-shapes. The final output is a global one-piece spline representation.

4 Model Partitioning

Suppose a solid region is bounded by a triangle-meshed surface ∂M (note that ∂M can be of high genus, but as the boundary of a solid object M, ∂M is a closed surface), this section focuses the computation of a group of curves $\{c\}$ on ∂M. These curves segment ∂M into subpatches ∂M_i, bounding subsolid regions M_i^s to be parameterized upon GPC cuboids. We denote these traced curves on ∂M as *polyedges*, as they will be mapped to edges of GPC cuboids. Our segmentation includes two main steps:

- Partitioning into T-shapes: we decompose the entire model into a group of T-shaped patches.
- T-to-cube decomposition: we generate polyedges on each T-shape and decompose it into four connected cube-like subpatches.

T-shapes are used as the basic primitive in our framework to decompose more complicated solid models. A T-shape, which represents the very simple 3-branched volume shape, has trivial topology and only contains Type-1 ill points.

4.1 T-Shape Segmentation

We use ∂T_i to represent a T-shape surface and T_i for its bounded volume. Our idea is to partition a given model M into several T-shaped subregions $\{T_i\}$. We achieve this segmentation through tracing curves on the boundary surface ∂M and partition it to subpatches ∂T_i or many simpler patches. This pipeline is illustrated in Fig. 6. The algorithm has following steps. Note that the challenge is how to ensure the segmented patch is geometrically similar to a "T" in 3D space, not just topologically.

Step 1. We first partition the input ∂M into several initial part-aware patches with nonintersecting cutting curves.

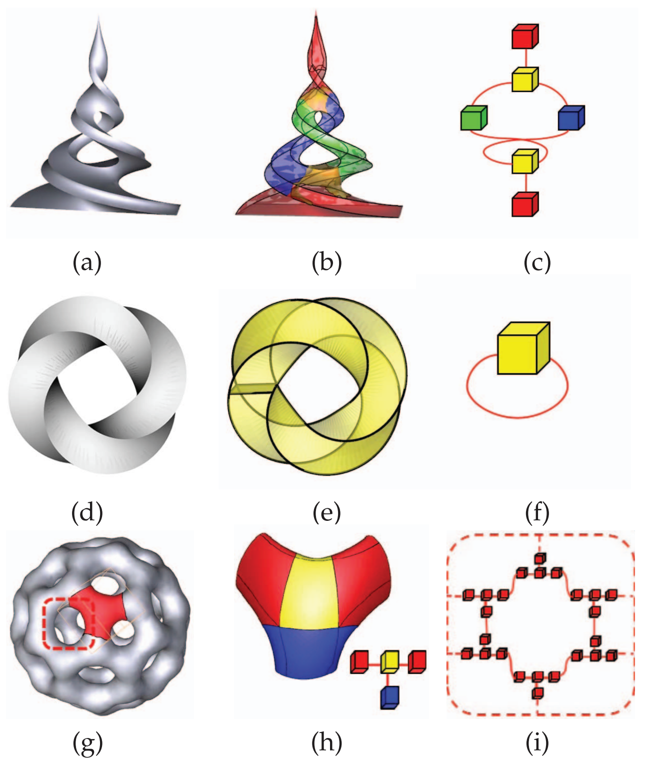


Fig. 4. GPC can handle more generalized models. Row 1: (a) The highly twisted swirl model, (b) Its GPC, and (c) Its topological graph. Row 2: (d) The non-axis-aligned möbius model, and (e,f) Its GPC and topological graph. Row 3: (g) The bucky model with complex topology, (h) It is decomposed into small "T-shapes" with four cuboids. (i) A subset of the GPC graph around the hole.

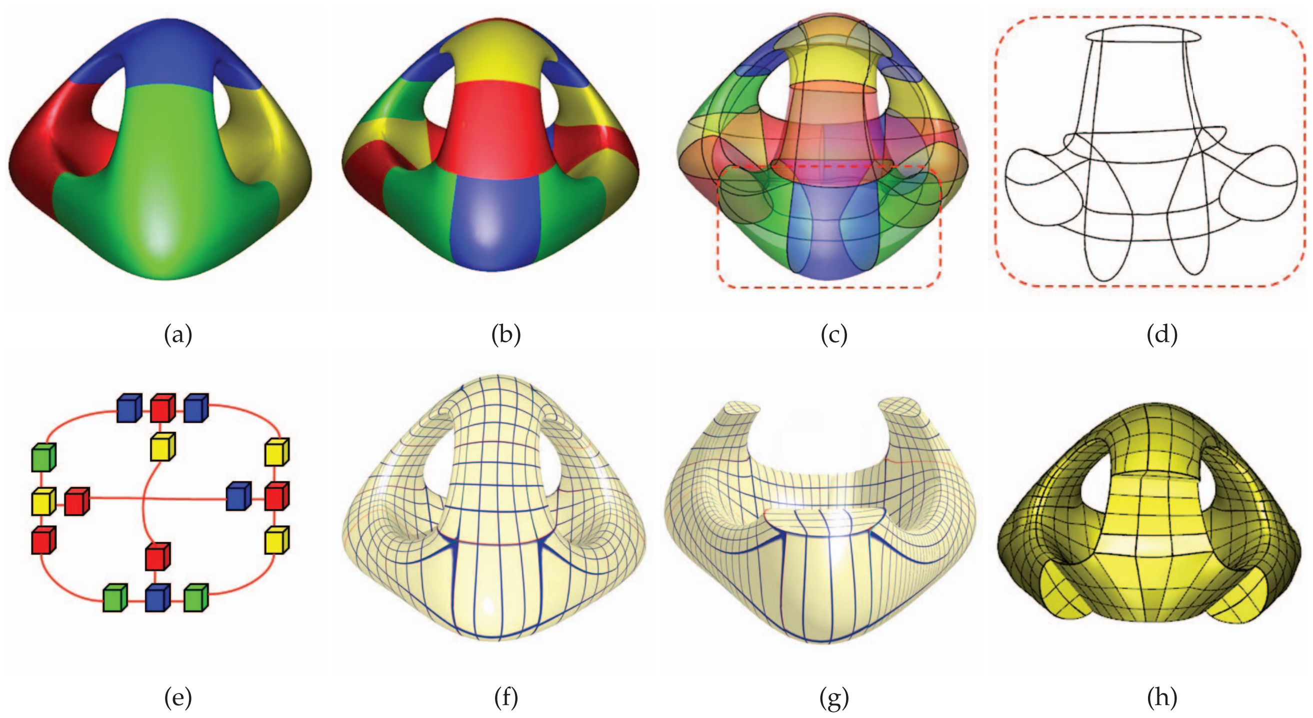


Fig. 5. GPC and spline construction pipeline. (a) The input genus-3 model is first decomposed into some "T-shape" patches. (b) Each "T-shape" is further decomposed into four cuboids. (c-d) Overlay all cuboid edges onto the model to visualize the global structure. (e) All cuboids comprise a topological GPC. (f-g) Construct the parametric mapping between the input model and its GPC. (h) Transform the model into a volumetric spline representation.

Any closed surface (the boundary of a solid model) can be partitioned in this way [37]. Different geometric criteria can be integrated in this unified partitioning framework. We choose volume-aware shape descriptors such as the shape diameter functions [10] to guide our cutting curve tracing.

Step 2. Upon a complete decomposition, we construct an abstraction graph: A node represents a patch, an edge connecting two nodes indicates their patch adjacency, and an edge connecting a node to itself indicates a handle loop. Fig. 6a shows a 4-torus with colored part-aware segmentation and the resulting abstraction graph.

Step 3. We modify each partitioned patch to a standard shape. It means that we split the abstraction graph's nodes with high valance until all nodes have ≤ 3 incident edges (a graph node with $d = 3$ represents a 3-branch patch, i.e.,

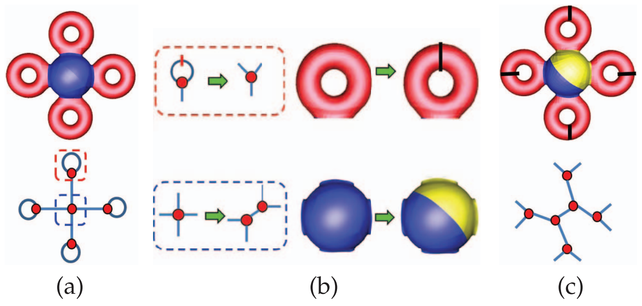


Fig. 6. Model segmentation into "T-shape" patches. (a) The part-aware segmentation and its abstraction graph. The nodes in the graph have different cases for edge connection (red and blue regions). (b) For each case, we have corresponding operations on the graph and input model. (c) Our operation guarantees that the resulting nodes in the graph are all degree $d = 3$, and the model is segmented into T-shapes.

T-shape, and $d = 1$ or $d = 2$ indicates the patch that bounds a tube). We partition every patch through analyzing all connected edges:

(3.1) *Handle loop* (see Fig. 6b, Row 1). We generate the shortest handle loop by [38] and then cut along it. In the abstraction graph, this partitioning cuts the loop into two edges.

(3.2) *High Valence* ($d > 3$) *branch* (see Fig. 6b, Row 2). We partition it to two connected nodes n_1 (valence-$d-1$) and n_2 (valence-3). Then, we repeat the split until all newly generated nodes are valence-3. To achieve this idea, we first choose two boundaries (a pair with the closest distance). Then, we utilize the technique in [37] to generate a cutting curve that covers two boundaries and avoids any intersection. This curve segments the patch into two patches, one with three boundaries (i.e., a T-shape) and another one with $d-1$ boundaries. We again execute the same partitioning method on the second patch until only three boundary patches exist. Row 2 shows an example of the cutting loop.

After repeating the above operations on every node, we can get a decomposition result where every node has its valence equivalent to 3 or less, as shown in Fig. 6c. Compared with existing partition techniques, our segmentation method is uniquely spline friendly: No prior segmentation result considers the critical issues in splines like singularities and ill points. Without addressing these issues, a segmentation is less suitable for spline conversion. Our T-shape-based segmentation, however, is completely singularity-free and ill-point controllable.

Cutting curve loops should be prevented from intersecting each other in our system. This can be ensured by not

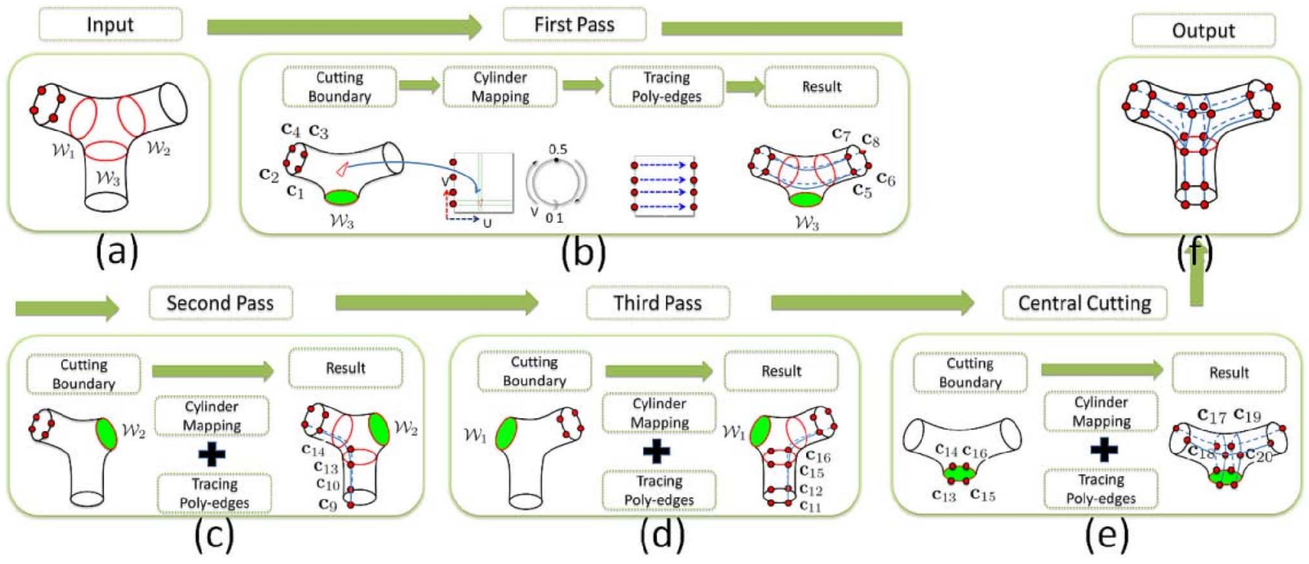


Fig. 7. Illustration of T-to-cube segmentation.

allowing a newly traced curve hitting (vertices of) existing loops. When two loops are very near and the triangle mesh is very sparse, triangles around this region will be subdivided to ensure the topologically correct tracing without intersection (for mesh refinement to ensure reliable curve tracing, please see [37] for details).

4.2 T-to-Cube Segmentation

We process a set of T-shapes ∂T_i or tube-shaped (cylinder) patches, one-by-one in an arbitrary order. ∂T_i is first partitioned into four subpatches ∂M_{ij}, then we generate corners and polyedges on each ∂M_{ij} (recall that polyedges are the traced curves that will be mapped to the edges of cuboid domains), as shown in Fig. 7. Meanwhile, for any simple tube-shaped patch, we can generate its corners and polyedges directly by the Step 2, the first pass, i.e., Fig. 7b. Finally, each resulting patch has eight corners and 12 polyedges like a cuboid. To guarantee corner alignment, when we determine one T-shape's result, we transfer its corners on the boundaries to the adjacent T-shapes if they are not processed yet.

Step 1. We generate three cutting lines $\mathcal{W}_1$, $\mathcal{W}_2$, and $\mathcal{W}_3$ (See Fig. 7a). We first find four corners on one boundary. We denote this boundary as "left" while arbitrarily denoting other two as "right" and "bottom." Positions of four corners are determined by its previously processed adjacent T-patch (except for the first processed T-shape, on which we manually set these four corners). To generate three cutting lines, we detect three branches of ∂T_i by extracting associated skeleton [39], with three resulting cutting lines.

Step 2. We generate all polyedges and corners on a T-shape ∂T_i, separately in three passes (Figs. 7b, 7c, and 7d). Each time we trace polyedges between two boundaries with three substeps.

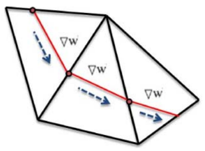
(2.1) We first remove the third long branches by cutting along its cutting lines (e.g., $\mathcal{W}_3$ in Fig. 7b). After filling the cutting hole [40], the resulting surface is a 2-boundary tube-shaped patch $\partial\mathcal{P}$.

(2.2) We map the tube shape to a cylinder domain $[\mathbf{u}, \mathbf{v}]$ following the approach of [5]. We shall briefly describe this algorithm: First, set $u = 0$ for vertices on one boundary and $u = 1$ for the other boundary, solve $\Delta u = 0$ by mean value coordinates [27]. Second, trace an iso-v curve along ∇u from an arbitrary seed vertex on the boundary $u = 0$ to the other boundary $u = 1$ and slice along this isocurve and get two duplicated boundary paths, then set $v = 0$ and $v = 1$ on them, respectively, and solve $\Delta v = 0$. The $\partial\mathcal{P}$ is, therefore, parameterized onto a cylinder domain.

(2.3) We generate polyedges between possible node pairs based on the cylinder-parameterized patch. For the first pass, we trace four edges from all corners on the left boundary to the right. For the second pass, we find two corners on the left boundary with shortest Dijkstra distance to the bottom $(\mathbf{c}_1, \mathbf{c}_2)$ as shown in Fig. 7c and trace two edges from them to the bottom. For the third pass, we choose pairing corners of $\mathbf{c}_1$ and $\mathbf{c}_2$ on the right boundary $(\mathbf{c}_5, \mathbf{c}_6)$ and trace two edges to the bottom (the possible node pairing/polyedge tracing algorithm is described below).

Step 3. We generate polyedges and corners for the central cuboid cutting. With four intersection corners (between the bottom cutting line and the traced paths) generated in the second and the third pass, now we trace polyedges between two intersection corners in each pass $(\mathbf{c}_{13}, \mathbf{c}_{14}$ and $\mathbf{c}_{15}, \mathbf{c}_{16})$.

Tracing polyedges. The above algorithm involves tracing edge $[\mathbf{c}_1, \mathbf{c}_2]$ on a cylinder parameterized patch $[\mathbf{u}, \mathbf{v}]$. According to the processing queue, $\mathbf{c}_2$'s location is either already determined by other precedent patches or is not yet known. For an unknown $\mathbf{c}_2$, we trace the polyedge from the starting corner $(\mathbf{c}_1)$ along the gradient direction $\nabla\mathbf{u}$ to another boundary at a new point $\mathbf{c}_2$. For a determined $\mathbf{c}_2$, we map both $\mathbf{c}_1$ and $\mathbf{c}_2$ to the cylinder domain $[\mathbf{u}, \mathbf{v}]$ and trace the straight line on the domain between them, then project this parametric straight line back to the patch and get the resulting polyedge. Note that none of polyedge is restricted to mesh edges. We allow them to cross and split the mesh triangles. This strategy enables more smooth path lines.



Node pairing. When we trace polyedges, it is very possible that all corners' locations on two boundaries are predetermined by other precedent patches. In such scenario, we desire to pair two boundaries' corners before tracing edges between unpaired corners. Intuitively, the traced path should be least deviated from the gradient of the harmonic field. Suppose, we are tracing paths between boundary b_1 and b_2. If corners on both b_1 and b_2 are predetermined, we trace the gradient line from b_1's corners and get ending nodes on b_2. Then, we compute and find the pairing between ending nodes and corners on b_2, satisfying that the sum of total distance between each pair is minimized. In this way, we can get the pairing between corners on b_1 and b_2; If corners on b_2 are not predetermined yet, we directly use the ending nodes as the new determined corners and, thus, get the pairing. In practice, we can merge edge tracing in the second/third pass (Figs. 7c and 7d) together: we determine the four node pairing together to avoid possible intersected polyedges generated between two passes.

Feature-preserving segmentation. Although the above automatic algorithm can handle most of models very well, sometimes users still expect to use several sharp features as the polyedges. For example, this choice is specially natural and meaningful on the strong symmetric man-made models with sharp features (e.g., CAD models in Figs. 4b, 4c, 4d, and 4e). Specifically, a scaling factor is applied to edges on feature curves, so they are considered shorter in the Dijkstra path tracing. Therefore, features will be on the traced curves and polyedges if we compute the shortest path between corners. Figs. 4a, 4d, 4e, and 2d show the results with feature-preserving polyedges. In practice, this method can only pick a few major feature lines (like in the twirl model, the polyedges are sharp features we pick). It is still difficult to handle more complex features. Instead, we can preserve the extra sharp features through the following spline fitting step.

5 PARAMETERIZATION

After the input model M is decomposed into subpatches $\{\partial M_{ij}\}$, bounding topological solid cuboids $\{M_{ij}\}$, we now perform cuboid parameterization of $\{M_{ij}\}$. We first map the patch boundary to the cuboid domain surface. Then, we use this mapping as boundary condition and compute the interior volumetric parameterization.

5.1 Surface Parameterization

The subpatch ∂M_{ij} computed previously has eight corners and 12 polyedges (see Fig. 8a), we partition ∂M_{ij} into six topological rectangles, then solve three harmonic mappings $\Delta u = 0$, $\Delta v = 0$, $\Delta w = 0$ on all rectangles. Each time we pick two opposite rectangles as two isoplane domains on one direction (e.g., $u = 0$ and $u = 1$). Then, we compute the parameters of this direction (u) on all other four rectangles. For example, to solve $\Delta u = 0$, we select eight polyedges on two opposite rectangles (see Fig. 8b). Four red polyedges bound an iso-u rectangle ($u = 0$) and the four blue polyedges bound another iso-u plane ($u = 1$). Then, we compute the approximated discrete harmonic map $\Delta u = 0$ [27] on other regions. Fig. 8c illustrates the computed u. Similarly, we can compute the harmonic scalar fields of v and w with $\Delta v = 0$ and $\Delta w = 0$, respectively. After solving three harmonic mappings, each vertex on the surface patch is mapped to a coordinate (u_0, v_0, w_0) on the cube surface. The surface parameterization is illustrated in Fig. 8d.

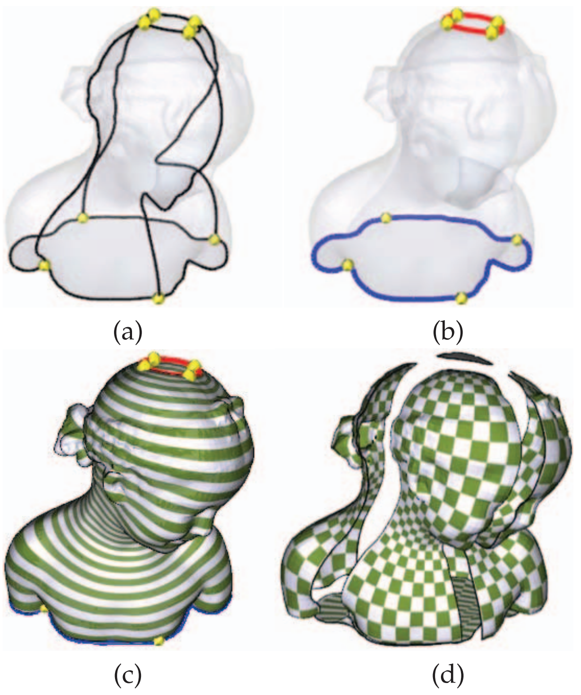


Fig. 8. Illustration of surface parameterization.

5.2 Volumetric Parameterization

We compute the volumetric parameterization of M_{ij} on a set of $n_0 \times n_1 \times n_2$ grid points. These grid points correspond to the uniformly sampled coordinates in the parametric space (u, v, w). This volumetric parameterization can be considered as finding the locations of these nodes within M_{ij}. Similarly, as we discussed in surface parameterization, we need to find the point locations that minimize the equations $\Delta u = 0, \Delta v = 0$, and $\Delta w = 0$ in 3D space.

The $n_0 \times n_1 \times n_2$ grid points include two categories: the surface grid points and interior points. We determine their positions as follows:

1. If the parameter of a grid point n falls on the domain surface, we can always find its location on $\partial\mathcal{M}$ by the parameter of n, where n's parameter always falls into a triangle $[v_1, v_2, v_3]$ of $\partial\mathcal{M}$ on the parametric domain with corresponding Barycentric coordinates $\lambda_1, \lambda_2, \lambda_3$, then its spatial location is interpolated as $\sum_{i=1}^{i=3} \lambda_i \mathbf{P}(v_i)$, where $\mathbf{P}(v)$ denotes the $3D$ position of vertex v.
2. Keeping the surface points fixed, we compute the interior point position by minimizing $3D$ Laplacian (1), where $\mathbf{n}_{ijk}$ and $\mathcal{N}_{ijk}^{\lambda}$ represent the node and its neighbor's spatial positions in (x, y, z) and w_λ is the point weight. In practice, each node is moved to the

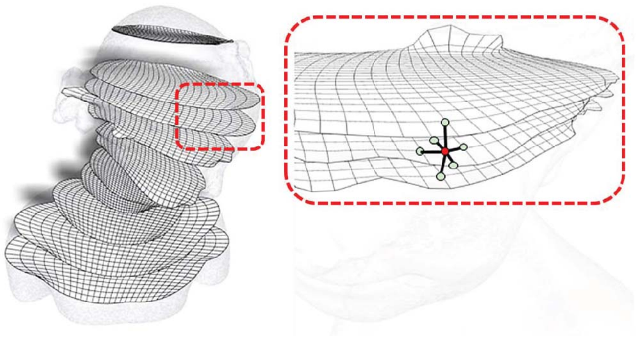


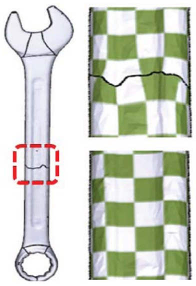
Fig. 9. Volumetric mapping. We extract sample points as a hexahedral model. Each node has six neighbors for solving 3D Laplacian in (1).

weighted mean center of their six neighbors. Here, the choice of weight w_λ has been studied in [21], [23]. In our implementation, we simply use the uniform weight $w_\lambda = 1/6$ as suggested in [40] and [41], as illustrated in Fig. 9:

$$\mathbf{E}(\mathbf{n}_{ijk}) = \sum_\lambda w_\lambda \times \|(\mathbf{n}_{ijk} - \mathcal{N}^\lambda_{ijk})\|, \lambda \in N_b(\mathbf{n}_{ijk}). \quad (1)$$

We move grid points iteratively. The update converges when changes of all node positions are smaller than a threshold during one iteration. Figs. 10a, 10b, and 10c show the computation results of the femur model after 20, 60, and 80 iterations.

Refinement across cutting boundary. Before merging, the parameterization of two adjacent subpatches are already computed separately. Along the cutting interface, only C^0 continuity is guaranteed and the cutting boundary is not smooth. We perform a refinement to improve this smoothness. To reduce computation time, we only extract a small region from each patch. For example, we pick a region from one patch within the parameter $(1-\alpha, 1) \times (0,1) \times (0,1)$, and $(0,\alpha) \times (0,1) \times (0,1)$ from another patch if two patches are connected along ∇u direction (α is a small scalar value). Gluing two extracted region together, the new patch also has eight new corners and 12 new polyedges (recall that polyedges between two adjacent patches are aligned along the boundary), thus we can recompute the surface mapping and volumetric mapping on the new patch. Meanwhile, this recomputing is subject to an extra constraint, on regions that connect to an extra third cuboid. We keep these region's parameter unchanged during recomputing, to avoid our modification destroying global parameter consistency.



6 GPC-Splines

Two challenging issues must be addressed when designing the mesh-to-spline transformation over GPC. First, allowing

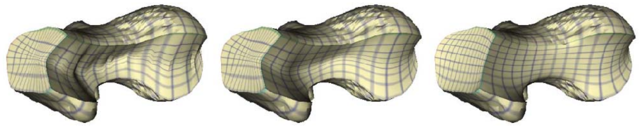


Fig. 10. Results of the cut-out view of the interior femur model by solving (1) after 20, 60, and 80 iterations.

adaptive refinement without significantly increasing control points is highly desirable since volumetric spline fitting usually requires a large number of control points when we seek high approximation accuracy. Second, unlike conventional B-splines that each control point and its knots are associated with global coordinates, GPC provides only locally defined parameters in each cuboid domain. This is because a global realization of GPC parametric domain in 3D euclidean space is oftentimes impossible on highly twisted/high genus models. Thus, we design a unique GPC-spline algorithm using a point-based scheme.

In principle, a volumetric cubic spline can be viewed as a point-based spline (PB-spline): Each control point C_i (located in parametric cube D^j with local coordinate $\mathbf{c}_i^j$) is associated with three knot vectors along three principal axes: $r = [r_1, r_2, r_3, r_4, r_5]$, $s = [s_1, s_2, s_3, s_4, s_5]$, $t = [t_1, t_2, t_3, t_4, t_5]$, where $\mathbf{c}_{ij} = (r_3, s_3, t_3)$. All knots can be determined using a *ray-tracing* strategy [34]. For any sample point with (u, v, w) as its local parameter, the blending function is

$$B_i(u,v,w) = N_r(u) \times N_s(v) \times N_t(w), \quad (2)$$

where N_r, N_s, and N_t are cubic B-spline basis functions associated with the knot vector r, s, and t, respectively. The formulation for PB-splines is

$$P(u,v,w) = \frac{\sum_0^n C_i B_i(u,v,w)}{\sum_0^n B_i(u,v,w)}. \quad (3)$$

We modify the above equation to construct GPC splines. The GPC domain comprises a collection of coordinate charts locally defined in individual cuboid. Adjacent local parametric coordinates are transformed coherently by transition functions, which can be encoded in a GPC-graph structure. Consequently, the global PB-splines are piecewise rational polynomials defined on GPC, whose transition functions between adjacent cuboids are compositions of simple cuboid translations and rotations of $n\pi/2$, where n is an integer.

In a cuboid D^j, given an arbitrary parameter $\mathbf{h}$, also denoted as $\mathbf{h}^j$, the spline approximation can be carried out as follows:

1. Find all the neighboring cubes $\{D^i\}$ that support $\mathbf{h}$ (i.e., it contains control points C_k that may support $\mathbf{h}$);
2. The spline function is:

$$P(\mathbf{h}) = \frac{\sum_{k=0}^n C_k^i B_k(\phi^{ij}(\mathbf{h}^j))}{\sum_{k=0}^n B_k(\phi^{ij}(\mathbf{h}^j))}, \quad (4)$$

where $\mathbf{h}^j$ is the local parametric coordinate of point $\mathbf{h}$ in the cube domain D^j, ϕ^{ij} is the transition function from cube domain D^j to D^i, and C_k^i denotes the control point k in the cube domain D^i.

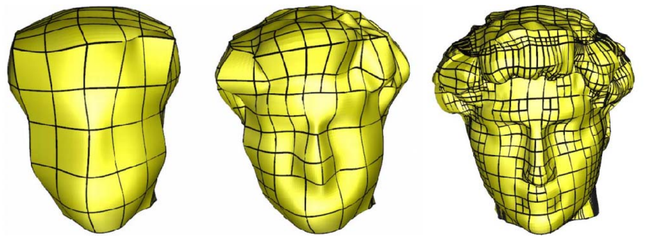


Fig. 11. Hierarchical spline fitting results at levels 0, 1, and 2, respectively.

In theory, a transition function ϕ^{ij} from cube domains D^j to D^i is a composition of translations and rotations following the shortest path from cube D^j to cube D^i in the GPC-graph. Suppose $\widetilde{D^iD^j} := D_1(= D^i) \rightarrow D_2 \ldots \rightarrow D_n(= D^j)$, and the transition function $\Phi_{(i,i+1)}$ (derived by way of cube gluing) from D_{i+1} to D_i is already known, then ϕ^{ij} is formulated by

$$\mathbf{h}^i = \phi^{ij}(\mathbf{h}^j) = \Phi_{1,2}(\Phi_{2,3}(\ldots \Phi_{n-1,n}(\mathbf{h}^j))).$$

In practice, because most control points only influence a very small local region and do not cut across nonadjacent cubes, we observed that only using a neighboring cube transition function is usually enough.

Along any merging region, two connected cubes share the same domain size along the merging face (i.e., we forbidden partial gluing between a large and small cubes). Therefore, when we merge two cuboids' control grid (with the same resolution), all the control points and intervals along the merging faces will merge coherently, without any T-junction before hierarchical fitting.

6.1 Hierarchical Fitting

Following above GPC-spline definitions, we develop a hierarchical fitting scheme to approximate volumetric models. For a sample point $f(\mathbf{h}_i)$ in the model whose parametric coordinate is $\mathbf{h}_i$ (defined by the volumetric parameterization computed in previous sections), $P(\mathbf{h}_i)$ is our GPC-spline representation. We minimize the following equation:

$$E_{dist} = \sum_{i=0}^{n} \|P(\mathbf{h}_i) - \mathbf{v}_i\|^2, \qquad (5)$$

which can be rewritten in matrix format

$$\frac{1}{2}\mathbf{C}^T\mathbf{B}^T\mathbf{B}\mathbf{C}^T - \mathbf{V}^T\mathbf{B}\mathbf{C}, \qquad (6)$$

where $\mathbf{C}$ is the vector of control points, $\mathbf{V} = \mathbf{v}_i$ is the vector of sample points, and $\mathbf{B} = B_i(\mathbf{h}_i)$ is the matrix of basis functions. This least square problem is not difficult to solve numerically. Given a sample parametric point h in GPC, to decide if we need to refine the approximation, we measure the root-mean-square error (RMS) $\sigma(\mathrm{h})$ between its spatial position $f(\mathrm{h})$ and its spline approximation $P(\mathrm{h})$. Algorithm 1 documents the main steps. The input includes all sample points and an initial control grid with control points. The initial control grid mimics the structure of GPC: Each cube corresponds to a local regular control grid. All local grids are topologically glued coherently following the GPC-graph, generating a one-piece global control grid. The function KnotVectors collects three direction knots for each control point. We use the same "ray-tracing" strategy in [34]. InfluencedSamples returns all sample points in the influenced region of a control point. Transition transports a local parameter from one cube to another cube. AssembleMatrix assembles the matrix for (6) and SolvingEquation solves it and determines the control point positions. FittingError returns the worst fitting result in a small grid. Subdivision divides a grid uniformly into eight smaller subgrids. Fig. 11 illustrates our hierarchical fitting results.

Algorithm 1. Hierarchical spline fitting.

```
Input: Initial control grid $\mathcal{L}_g$,
       List of sample points $\mathcal{L}_s$,
       List of control points $\mathcal{L}_c$,
       Fitting error threshold $\epsilon$
Output: all control points positions.
loop
  //Update control point knot vectors
  for all $\mathcal{L}_c$ do
    $\mathbf{c} = \mathcal{L}_c$ .next()
    c .knots = KnotVectors($\mathbf{c}, \mathcal{L}_g$)
    $\mathcal{L}'_s =$ InfluencedSamples ($\mathbf{c}, \mathcal{L}_s$)
    for all $\mathcal{L}'_s$ do
      $\mathrm{s} = \mathcal{L}'_s$ .next()
      s .ctrlist.push_back($\mathbf{c}$)
    end for
 end for
  //Compute basis functions for samples
  for all $\mathcal{L}_s$ do
    $\mathrm{s} = \mathcal{L}_s$ .next() $B_{total} = 0$
    $\mathcal{L}'_c =$ s .ctrlist $\mathcal{L}_B = \{\}$
    for all $\mathcal{L}'_c$ do
      $\mathbf{c} = \mathcal{L}'_c$ .next()
      param=Transition (s.cube#, c .cube#,
      c.param)
      B= BasisFunction (param,c.knots)
      $\mathcal{L}_B$.puch_back(B) $B_{total} = B_{total} + B$
   end for
   AssembleMatrix ($\mathcal{L}_B, B_{total}, \mathbf{s}$)
 end for
 //Fitting and evaluation
 SolvingEquation()
 for all $\mathcal{L}_g$ do
   $\mathbf{g} = \mathcal{L}_g$.next()
   if $FittingError(\mathbf{g}) > \epsilon$ then
     $\mathcal{L}'_g =$ Subdivision(g)
     $\mathcal{L}_g$.delete(g) $\mathcal{L}_g$ .insert($\mathcal{L}'_g$)
   end if
  end for
  Stop if no updated grid
 end loop
```

7 Implementation and Discussion

Our experimental results are implemented on a 3 GHz Pentium-IV PC with 4 Giga RAM. To demonstrate the versatility of our approach (therefore, the flexibility of our

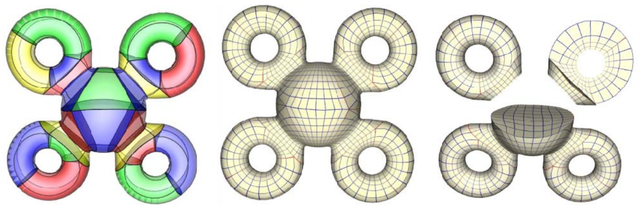


Fig. 12. The 4-sphere model visualized with cuboid organization, polyedge structure, surface parameterization, and volumetric parameterization.

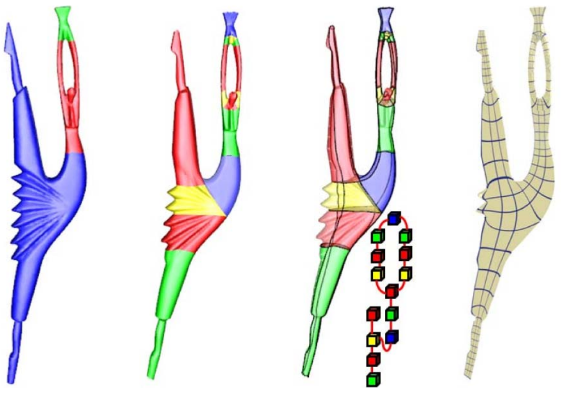


Fig. 13. The dancer model visualized with T-shape decomposition, cuboid organization, polyedge structure, GPC graph, and volumetric parameterization.

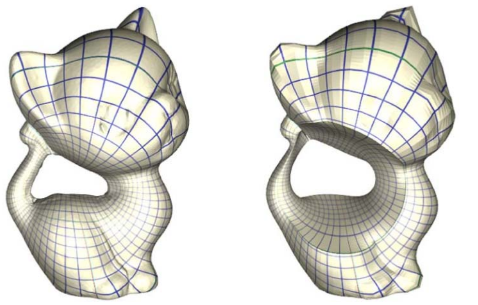


Fig. 14. The kitten model visualized with surface and volumetric parameterization.

TABLE 1
Statistics of Various Test Examples

Model	Genus	Twisted	# T-shape	# Cuboid	# Ill-points
genus-3	3	no	4	16	8
bucky	31	no	60	240	120
mobius	1	yes	1	1	0
twirl	1	yes	2	6	4
4-sphere	4	no	6	24	12
bimba	0	no	1	1	0
femur	0	no	1	1	0
wrench	1	no	2	8	4
dancer	1	no	3	14	6
david	3	no	4	12	24
greek	4	no	6	19	12

computational framework), we construct GPC splines for many models. Our experiments include models with twisted shape: twirl (Fig. 4, Row 1), möbius solids (Fig. 4, Row 2); and with complex topology: bucky (genus 31, Fig. 4, Row 3), genus-3 (Fig. 5), 4-sphere (genus 4, Fig. 12); and with complex conceptual parts: wrench (Fig. 2), dancer (Fig. 13), and greek and david (Fig. 15). Table 1 summarizes the statistics of the GPC construction, including every model's properties (genus, twisted/not twisted), the number of T-shapes, cuboids, and ill points.

It may be noted that our parameterization algorithm may not guarantee a globally minimized angle and volume distortion. However, since our algorithm decomposes the input into part-aware patches, each of which is parameterized on a geometrically similar cuboid, the distortion is satisfactory for our spline construction. The models of dancer, 4-sphere, kitten, greek and david (Figs. 12, 13, 14, and 15) demonstrate several surface and volumetric GPC parameterization results. Fig. 16 shows several volumetric spline approximation results. We overlay the control grid line

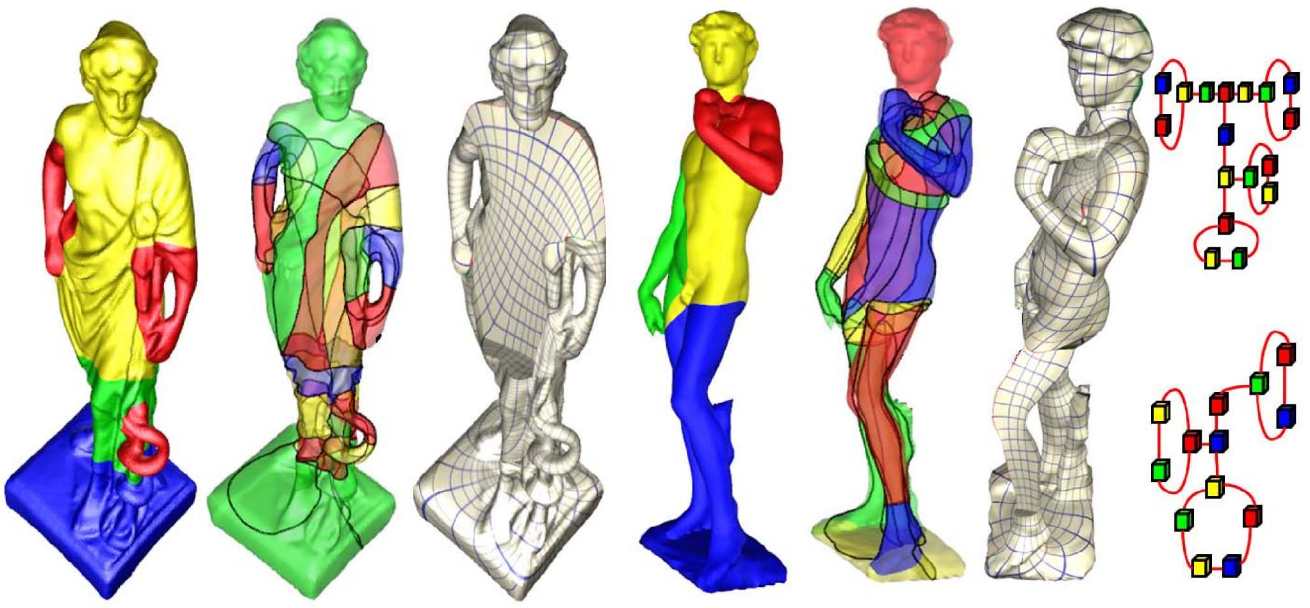


Fig. 15. The greek and david model visualized with T-shape decomposition, cuboid organization, polyedge structure, volumetric parameterization and their GPC graphs, respectively.

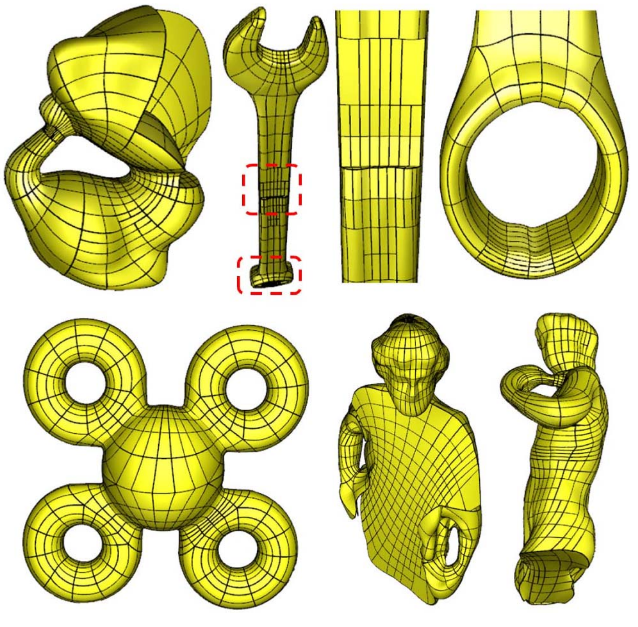


Fig. 16. The volumetric spline approximation results.

TABLE 2
Statistics of various spline examples

Model	#. Surface vertices	#. Volume vertices	#. Control points	RMS error	Running time
kitten	12403	40000	3020	0.35%	202s
wrench	7550	12000	2966	0.2%	105s
4-sphere	2042	22800	1088	0.2%	47s
genus-3	6632	51200	1280	0.17%	162s
david body	15572	81600	5956	0.37%	890s
greek body	20109	91900	7265	0.4%	1096s

(black lines) onto the fitting results, and the T-junctions on the control grid reduce the control point greatly while still preserving the shape details. The statistical results are given in Table 2. The table shows that the vertices' number increases dramatically when we convert a surface model into a volume data. Our spline scheme can significantly reduce control points for shape representation. In most of our experiments, approximation with good quality can be achieved within three levels of hierarchical refinement. The fitting qualities are measured by RMS errors normalized to the overall sizes of solid models.

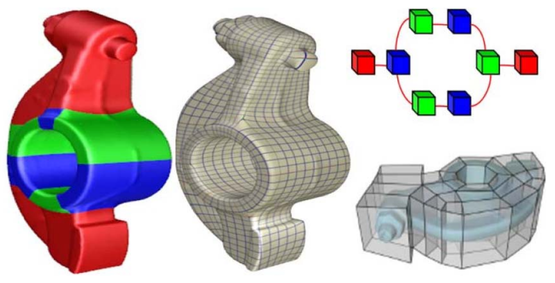


Fig. 17. Our segmentation/mapping result of the rocker-arm model (left/middle). Our GPC (right up) has only eight cuboids/no singularity, compared with 26 cubes/four singular points (right bottom, courtesy of [29]).

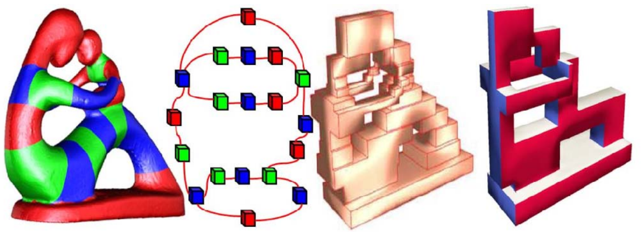


Fig. 18. Comparisons of different methods on the fertility model (courtesy of [8] and [17]). Our domain has significant improvement on cuboid and ill-point number.

Comparisons. We compare our method with other volumetric parametric domain construction and mapping approaches: [13], [8], [4], [6], [16], and [21]. As shown in Table 3 and Fig. 4, our method has advantages in the following aspects. First, our method works well for volumes with complex topology and structure. Second, our domain does not have any singularity and can control the type and number of ill points (which is highly desirable for spline construction). Our domain construction does not require tedious design, even for very complex shape input. Meanwhile, we can also flexibly edit the cube domain to better approximate the shape interactively.

We also test our system on the rocker-arm model, which also appears in other papers (e.g, CubeCover in [29]). As the comparison shown in Fig. 17, our parameterization has the same quality as in [29]. However, 26 cube domains and four singular points are used in [29], while we only have eight cuboids and no singularity. In Fig. 18, we compare our domain (Middle Left) with the methods in [8] (Middle Right) and [17] (Right) using the fertility model. Our domain significantly decreases the number of cuboids (19)

TABLE 3
Comparison with the existing approaches

Method	Tarini [13]	He [8]	Martin [4]	Zhang [6]	Lin [16]	Wang [21]	Hexahedral mesh [42]	Ours
Primitives	cube	cube	cylinder	cylinder	cube	sphere	numerous small cube no parameter	cube
Topology	axis-aligned	axis-aligned	symmetric	long branch	reeb-graph	genus-0	arbitrary	arbitrary
Twisted model	no	no	yes	yes	no	no	yes	yes
Singularity	no	no	center	center	no	center	large number	no
Ill-points	no control	large number	no	no	no control	no	large number	controllable
Domain construction	Artiest design	Axis scan	Simple	Simple	Simple	Sphere only	No domain existed	Simple
Editable domain	yes	no	no	no	no	no	no	yes

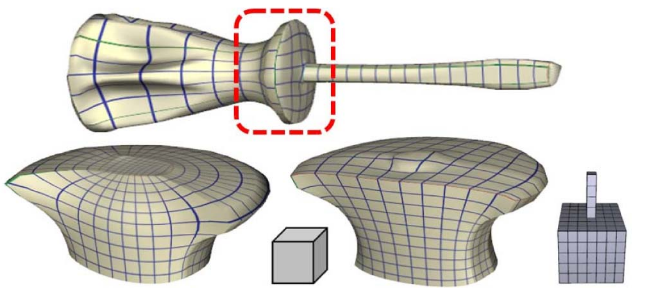


Fig. 19. Modified result of the screwdriver model (up). Mapping it to two separate domains (bottom right) instead of one cuboid domain (bottom left) can moderate distortion like extrusion round the handle top region.

as while as ill points (only on cuboids with more than two edges in the GPC-graph).

Discussions. Since singularity-free and ill-point simplification is the first priority in our spline-oriented system, this enforcement may lower mapping quality in certain region. According to users' requirement, we can always change it on the fly based on a hybrid system. Inside the current partitioning framework, we may further allow extra local segmentation to improve its geometry awareness. Upon initial partitioning, we detect long branches, and construct additional cuboids to parameterize these branches. For example, we map the axial shaft and handle of the screwdriver (Fig. 19) to separate cuboids. Compared with using only one cuboid, the distortion (e.g., the extrusion effect) around the handle top is significantly reduced. However, as mentioned above, this modified GPC decomposition will bring extra singularities, ill points, and merging cases. In this example, we add four extra"type-4" ill points.

Our system decomposes the input model mainly according to global shape and topology. This implies that it fails to handle the model with complex features if they are everywhere. Enforcing polyedges covering features (Section 4.2) can only recover major features, which are globally dominant. For spline construction, this is not a critical issue since we can always improve the fitting quality hierarchically around any sharp feature. However, many feature-based applications may require features to be retained. We will investigate how to preserve the feature as much as possible.

Our polyedge tracing algorithm cannot prevent them from intersecting with each other. Fortunately, our tracing algorithm can avoid intersection on a well partitioned part-aware patch. However, intersection may happen on a very poorly shaped T-shaped patch. We will develop an automatic method to detect degeneration and correct it.

8 Conclusion

We have presented a GPC spline framework for data transformation from surface meshes to continuous volumetric splines. The novelty of this paper lies at the systematic handling of GPC parametric domain without any strong assumption. Compared with CPC, GPC provides more generalized shape domain and better numerical stability to represent complicated models of arbitrary structure. We design a volumetric parameterization procedure based on GPC, which better handles solid objects with general topology and structure than existing volumetric parameterization techniques. We then devise a global "one-piece" volumetric spline based on GPC parameterization. The GPC construction enables a novel and desirable mechanism that facilitates the "one-piece" spline representation. Using local point-based strategy, global volumetric T-splines can be constructed on piecewise GPC because transition functions can be effectively computed from the GPC's topological structure. The entire spline framework affords hierarchical refinement and level-of-detail control. Our GPC volumetric splines have great potential in various shape design and physically based analysis applications. Our GPC is of great value to a wide range of geometry processing tasks, including volumetric isogeometric analysis [1], volume deformation, anisotropic material/texture synthesis.

Acknowledgments

This research is supported in part by US National Science Foundation Grants IIS-0949467, IIS-1047715, and IIS-1049448, Louisiana Board of Regents Research Competitiveness Subprogram (RCS) LEQSF(2009-12)-RD-A-06 and LA-BOR PFund: NSF(2011)-PFund-236; and the open program of the CAD&CG state lab, Zhejiang University, China.

References

[1] T.J. Hughes, J.A. Cottrell, and Y. Bazilev, "Isogeometric Analysis: Cad, Finite Elements, Nurbs, Exact Geometry and Mesh Refinement," *Computer Methods in Applied Mechanics and Eng.*, vol. 194, pp. 4135-4195, 2005.
[2] K. Wang, X. Li, H. Xu, and H. Qin, "Restricted Trivariate Polycube Splines for Volumetric Data Modeling," *IEEE Trans. Visualization and Computer Graphics,* vol. 18, no. 5, pp. 703-716, May 2012.
[3] B. Li and H. Qin, "Component-Aware Tensor-Product Trivariate Splines of Arbitrary Topology," *Computer and Graphics,* vol. 36, no. 5, pp. 329-340, 2012.
[4] T. Martin and E. Cohen, "Volumetric Parameterization of Complex Objects by Respecting Multiple Materials," *Computer and Graphics,* vol. 34, pp. 187-197, 2010.
[5] T. Martin, E. Cohen, and R. Kirby, "Volumetric Parameterization and Trivariate B-Spline Fitting Using Harmonic Functions," *Computer Aided Geometric Design,* vol. 26, no. 6, pp. 648-664, 2009.
[6] Y. Zhang, Y. Bazilevs, S. Goswami, C.L. Bajaj, and T. Hughes, "Patient-Specific Vascular Nurbs Modeling for Isogeometric Analysis of Blood Flow," *Computer Methods in Applied Mechanics and Eng.,* vol. 196, no. 29/30, pp. 2943-2959, 2007.
[7] H. Wang, Y. He, X. Li, X. Gu, and H. Qin, "Polycube Splines," *Computer Aided Design,* vol. 40, no. 6, pp. 721-733, 2008.
[8] Y. He, H. Wang, C. Fu, and H. Qin, "A Divide-And-Conquer Approach for Automatic Polycube Map Construction," *Computer and Graphics,* vol. 33, no. 3, pp. 369-380, 2009.
[9] A. Shamir, "A Survey on Mesh Segmentation Techniques" *Computer Graphics Forum,* vol. 27, no. 6, pp. 1539-1556, 2008.
[10] L. Shapira, A. Shamir, and D. Cohen-Or, "Consistent Mesh Partitioning and Skeletonisation Using the Shape Diameter Function," *The Visual Computer,* vol. 24, pp. 249-259, 2008.
[11] R. Liu, H. Zhang, A. Shamir, and D. Cohen-Or, "A Part-Aware Surface Metric for Shape Analysis," *Computer Graphics Forum,* vol. 28, no. 2, pp. 397-406, 2009.
[12] M. Ovsjanikov, J. Sun, and L. Guibas, "Global Intrinsic Symmetries of Shapes," *Computer Graphic Forum,* vol. 27, no. 5, pp. 1341-1348, 2008.
[13] M. Tarini, K. Hormann, P. Cignoni, and C. Montani, "Polycube Maps," *ACM Trans. Graphics,* vol. 23, no. 3, pp. 853-860, 2004.
[14] H. Wang, M. Jin, Y. He, X. Gu, and H. Qin, "User-Controllable Polycube Map for Manifold Spline Construction," *Proc. ACM Solid and Physical Modeling Symp. (SPM '08),* pp. 125-136, 2008.

[15] J. Xia, I. Garcia, Y. He, S. Xin, and G. Patow, "Editable Polycube Map for GPU-Based Subdivision Surfaces," *Proc. Symp. Interactive 3D Graphics and Games (I3D)*, pp. 151-158, 2011.

[16] J. Lin, X. Jin, Z. Fan, and C.C.L. Wang, "Automatic Polycube-Maps," *Proc. Fifth Int'l Conf. Advances in Geometric Modeling and Processing (GMP '08)*, pp. 3-16, 2008.

[17] J. Gregson, A. Sheffer, and E. Zhang, "All-Hex Mesh Generation via Volumetric Polycube Deformation," *Computer Graphics Forum*, vol. 30, no. 5, pp. 1407-1416, 2011.

[18] M. Floater and K. Hormann, "Surface Parameterization: A Tutorial and Survey," *Advances in Multiresolution for Geometric Modelling*. Springer, 2005.

[19] A. Sheffer, E. Praun, and K. Rose, "Mesh Parameterization Methods and Their Applications," *Foundations and Trends in Computer Graphics and Vision*, vol. 2, no. 2, pp. 105-171, 2006.

[20] K. Hormann, B. Lévy, and A. Sheffer, "Mesh Parameterization: Theory and Practice," *Proc. ACM SIGGRAPH '07 Courses*, 2007.

[21] Y. Wang, X. Gu, T.-F. Chan, P. Thompson, and S.-T. Yau, "Volumetric Harmonic Brain Mapping," *Proc. IEEE Int'l Symp. Biomedical Imaging: Macro to Nano (ISBI '04)*, pp. 1275-1278, 2004.

[22] X. Li, X. Guo, H. Wang, Y. He, X. Gu, and H. Qin, "Harmonic Volumetric Mapping for Solid Modeling Applications," *Proc. ACM Symp. Solid and Physical Modeling (SPM '07)*, pp. 109-120, 2007.

[23] T. Ju, S. Schaefer, and J. Warren, "Mean Value Coordinates for Closed Triangular Meshes," *Proc. ACM SIGGRAPH '05*, pp. 561-566, 2005.

[24] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki, "Harmonic Coordinates for Character Articulation," *ACM Trans. Graphics.*, vol. 26, no. 3, p. 71, 2007.

[25] K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo, and H.-Y. Shum, "Large Mesh Deformation Using the Volumetric Graph Laplacian," *ACM Trans. Graphics*, vol. 24, no. 3, pp. 496-503, 2005.

[26] S. Han, J. Xia, and Y. He, "Hexahedral Shell Mesh Construction via Volumetric Polycube Map," *Proc. 14th ACM Symp. Solid and Physical Modeling*, pp. 127-136, 2010.

[27] M. Floater, "Mean Value Coordinates," *Computer Aided Geometric Design*, vol. 20, no. 1, pp. 19-27, 2003.

[28] J. Xia, Y. He, X. Yin, S. Han, and X. Gu, "Direct-Product Volumetric Parameterization of Handle Bodies via Harmonic Fields," *Proc. Int'l Conf. Shape Modeling and Applications*, pp. 127-136, 2010.

[29] M. Nieser, U. Reitebuch, and K. Polthier, "Cubecover-Parameterization of 3d Volumes," *Computer Graphics Forum*, vol. 30, no. 5, pp. 1397-1406, 2011.

[30] J. Hua, Y. He, and H. Qin, "Multiresolution Heterogeneous Solid Modeling and Visualization Using Trivariate Simplex Splines," *Proc. ACM Symp. Solid Modeling and Applications*, pp. 47-58, 2005.

[31] C. Rössl, F. Zeilfelder, G. Nurnberger, and H. Seidel, "Reconstruction of Volume Data with Quadratic Super Splines," *IEEE Trans. Visualization and Computer Graphics*, vol. 10, no. 4, pp. 397-409, July/Aug. 2003.

[32] W. Song and X. Yang, "Free-Form Deformation with Weighted T-Spline," *The Visual Computer*, vol. 21, no. 3, pp. 139-151, 2005.

[33] X. Zhou and J. Lu, "Nurbs-Based Galerkin Method and Application to Skeletal Muscle Modeling," *Proc. ACM Symp. Solid and Physical Modeling (SPM '05)*, pp. 71-78, 2005.

[34] T. Sederberg, J. Zheng, A. Bakenov, and A. Nasri, "T-Splines and T-Nurccs," *ACM Trans. Graphics*, vol. 22, no. 3, pp. 477-484, 2003.

[35] T. Sederberg, D. Cardon, G. Finnigan, N. North, J. Zheng, and T. Lyche, "T-Spline Simplification and Local Refinement," *ACM Trans. Graphics*, vol. 23, no. 3, pp. 276-283, 2004.

[36] J. Zheng, Y. Wang, and H.-S. Seah, "Adaptive T-Spline Surface Fitting to Z-Map Models," *Proc. Third Int'l Conf. Computer Graphics and Interactive Techniques in Australasia and South East Asia (GRAPHITE '05)*, pp. 405-411, 2005.

[37] X. Li, X. Gu, and H. Qin, "Surface Mapping Using Consistent Pants Decomposition," *IEEE Trans. Visualization and Computer Graphics*, vol. 15, no. 4, pp. 558-571, July/Aug. 2009.

[38] T. Dey, K. Li, and J. Sun, "On Computing Handle and Tunnel Loops," *Proc. Int'l Conf. Cyber Worlds*, pp. 357-366, 2007.

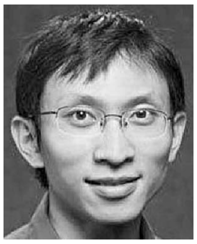
[39] D. Reniers and A. Telea, "Skeleton-Based Hierarchical Shape Segmentation," *Proc. Int'l Conf. Shape Modeling and Applications*, pp. 179-188, 2007.

[40] G. Taubin, "A Signal Processing Approach to Fair Surface Design," *Proc. 22nd Ann. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH '95)*, pp. 351-358, 1995.

[41] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel, "Laplacian Surface Editing," *Proc. Eurographics/ACM SIGGRAPH Symp. Geometry Processing (SGP '04)*, pp. 175-184, 2004.

[42] C. Carbonera and J. Shepherd, "A Constructive Approach to Constrained Hexahedral Mesh Generation," *Eng. with Computers*, vol. 26, pp. 341-350, 2010.

Bo Li received the BS degree in computer science from Zhongshan University (Sun Yet-sen University) and is working toward the PhD degree in the Department of Computer Science at Stony Brook University (SUNY). 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.



Xin Li received the BS degree in computer science from University of Science and Technology of China in 2003, and the MS and PhD degrees in computer science from Stony Brook University (SUNY) in 2008. He is an assistant professor in the School of Electrical Engineering and Computer Science, and the Center for Computational and Technology, at Louisiana State University. His research interests include geometric data modeling and processing, and their applications in graphics, vision, visualization, computational forensics, computational medicine, and robotics. He is a member of the IEEE and IEEE Computer Society. For more information, please visit http://www.ece.lsu.edu/xinli.

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 is currently a data analyst at Renaissance Technologies, LLC.

Hong Qin received the BS and MS degrees in computer science from Peking University, and the PhD degree in computer science from the University of Toronto. He is a professor of Computer Science in the Department of Computer Science at State University of New York at Stony Brook (Stony Brook University). In 1997, he 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. He served as the general cochair for Computer Graphics International 2005. He was the Conference cochair 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. He is currently 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.