Interactive Applications for Sketch-Based Editable Polycube Map

Ismael Garcia, Jiazhi Xia, Ying He, *Member*, *IEEE*, Shi-Qing Xin, and Gustavo Patow

Abstract—In this paper, we propose a sketch-based editable polycube mapping method that, given a general mesh and a simple polycube that coarsely resembles the shape of the object, plus sketched features indicating relevant correspondences between the two, provides a uniform, regular, and user-controllable quads-only mesh that can be used as a basis structure for subdivision. Large scale models with complex geometry and topology can be processed efficiently with simple, intuitive operations. We show that the simple, intuitive nature of the polycube map is a substantial advantage from the point of view of the interface by demonstrating a series of applications, including kit-basing, shape morphing, painting over the parameterization domain, and GPU-friendly tessellated subdivision displacement, where the user is also able to control the number of patches in the base mesh by the construction of the base polycube.

Index Terms—Digital geometry processing, surface parameterization, polycube map, GPU subdivision surface

1 Introduction

The motivations for this work come from the fact that most artists prefer modeling with quads, as quad geometry provides a better flow, tessellates cleaner, and deformations under animation are smoother, especially around joints [2]. Also, artists often perform intricate manipulations on the models during model creation to achieve a desired result. Operations like re-using model parts, painting, morphing, and preparing the model for current hardware pipelines can be complex without the appropriate tools.

A polycube is a natural generalization of a cubical space, being a useful parametric domain to ease some modeling operations of irregular input models and to create a regular map of the surface details. Tarini et al. [3] pioneered this concept to parameterize three-dimensional shapes with complex topology and geometry. Polycube maps have three characteristics that make them promising for graphics applications: First, the domain has a naturally *simple*, axis-aligned embedding in three-dimensional [4], that results in a *semiregular* quadrangulation [5] that can be easily constructed and visualized. A polycube is also an ideal domain for volume data processing [6], [7], [8]. Second, the cone singularities (polycube corners) have fixed structures, i.e., of valence 3, 5 or 6, which results in a small number of topological combinations—an important issue for modern tessellation hardware. Finally, their three-dimensional embedding is specially appealing from an *interface* point of view: It is easier and more intuitive to manipulate a three-dimensional embedded, axis-aligned polycube. We exploit this opportunity twice: To ease the user-driven, interactive construction of a polycube map (Section 3); and then to unlock several interactive applications of the resulting parameterized models (Section 4).

Constructing a polycube map is challenging: From the users' point of view, an ideal polycube mapping algorithm should have at least the following features:

- *Quality*: The map is a quasi bijection with low angle and area distortion.
- *User control*: The user can control the mapping by specifying optional features on the 3D model and their desired locations on the polycube domain.
- *Reuse*: The user can modify the polycube maps and reuse resources from existing maps.
- *Performance*: The algorithm is efficient, robust, automatic, and has the possibility of adding editable user-specified constraints to control the map.

There are several approaches to construct a polycube mapping [3], [9], [10], [11], [12], but none has all the desired features. For instance, Tarini et al. [3] does not guarantee a bijection, while others [9], [11], [12] do not allow user control. Wang et al. [10] require a lot of interaction to specify the polycube structure and is not suitable for large-scale models with complex geometry and topology. More importantly, none of them allow *editing* the polycube map in an easy and intuitive fashion. Editing on a polycube, which more closely resembles the gross structure of the model could be simpler, especially when allowed to control the rough shape of the mapping through simple sketches.

In this paper, we present the editable polycube map to overcome the limitations of existing approaches. See Fig. 1. Our method allows to construct the map in an intuitive and easy manner: Given a 3D model M and its polycube domain P, the user is able to approximately sketch features on M

- *I. Garcia and G. Patow are with the Departament d'Informàtica i Matemàtica Aplicada, Universitat de Girona, Edifici P-IV, Campus Montilivi, E-17071 - Girona (Girona), Spain. E-mail: {igarcia, dagush}@ima.udg.edu.*
- *J. Xia is with the Central South University, Computer Building, Room 411, Lushan Nan Load, Changsha 410083, China, and Nanyang Technological University, Singapore. E-mail: xiajiazhi@gmail.com.*
- *Y. He is with the School of Computer Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798. E-mail: yhe@ntu.edu.sg.*
- *S.-Q. Xin is with the Nanyang Technological University, Singapore, and the Institute of Computer Science & Technology, Ningbo University, Ningbo 315211, China. E-mail: xinshiqing@nbu.edu.cn.*

Manuscript received 21 Jan. 2012; revised 15 Aug. 2012; accepted 12 Nov. 2012; published online 21 Nov. 2012;
Recommended for acceptance by L. Kobbelt.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2012-01-0021.
Digital Object Identifier no. 10.1109/TVCG.2012.308.

1077-2626/13/$31.00 © 2013 IEEE Published by the IEEE Computer Society

(a) User sketches (b) Automatic segmentation (c) Polycube map(d) GPU subdivision displacement

Fig. 1. The proposed method allows the users to easily edit and control the polycube map by sketching the features/constraints on the 3D model and the polycube. The generated map is conformal and with low area distortion.

and P to specify feature correspondences. Then, our system automatically computes the map preserving the features on M to the user-specified locations on P. The user is always able to edit the features on M or P, providing precise control to mark and preserve fine-grained features for the different applications (e.g., kit-bashing), being easier to manipulate a 3D embedded, axis-aligned polycube.

A practical and useful application is *kit-bashing* [13], which refers to the widely used practice among artists of reusing previously made assets as accessories to quickly form a new model. With our scheme, the library of accessories and the resulting shape could be presented in a parameterized and tessellated form (see Section 4.1). We also introduce shape morphing as another practical application: It is possible to seamlessly blend or even obtain a smooth animation between shapes with a common polycube-based parameterization plus specific operators. See Section 4.2. We also present a tool that takes advantage of the dual parameterization of the polycube maps to help the user to easily paint complex models with concave surfaces that might be hard to reach (see Section 4.3). Finally, hardware-supported tessellation is already feasible and provides much faster model visualization than traditional methods [14], with specialized programmable units (Shader Model 5 hardware) [15]. The positive properties enumerated above allow an efficient implementation of GPU-based subdivision surfaces. Also, the reduced number of combinations, together with watertight sampling, allow for a continuous subdivision method that smoothly integrates with current production pipelines. Finally, we are able to provide coarse regular base meshes with a reduced memory footprint.

The process should be compared with the traditional work artists do on a model for an environment like a computer game. In general, if the new model is based on an existing one (e.g., from a laser scanning process), the generation implies manually performing retopology operations to create a base mesh (taking into account subdivision), to then transfer the original details by normal projection, which does not ensure bijectivity. Usually, artists start from a coarse base model, quite often sculpted using a rough shape like a polycube. Then, the model is imported into an application like ZBrush [16], and further subdivided with a couple of Catmull-Clark subdivision steps. Then, the artist has to model/transfer the fine-grained details onto the final model. Obviously, this is time consuming and quite redundant, even able to introduce fold overs in the parameterization. Our proposal is to quickly establish a quasi-bijective correspondence between the coarse polycube and the input model, and then let the user perform the different editing tasks. The resulting model will have all the enumerated properties and would be ready for use in a production environment.

The specific contributions of this paper include a method that, from a general mesh, creates a high-quality and user-controllable polycube map in an efficient and intuitive manner. Our method allows to easily modify and fine-tune the map through intuitive stroke-sketching operations. Usually, a coarse polycube and approximate strokes are usually enough. Compared to our preliminary work that mainly targets genus-0 models [1], the current framework works for a wider range of models with nontrivial topology and complicated geometry. Also, the simple, intuitive nature of the polycube map proves to be a substantial advantage from the point of view of the interface, opening the door to new possibilities and applications. Besides the GPU-friendly tessellated subdivision shown in [1], we demonstrate new applications, such as kit-bashing, shape morphing, and painting over the parameterization domain.

2 Related Work

Polycube mapping. Tarini et al. [3] pioneered the concept of polycube maps by approximating the input three-dimensional model by a polycube and constructing its dual space through six projection functions. However, two vertices on the same projection line share the same image, breaking bijectivity, a fundamental feature for many applications. Also, their method has strict requirements on the shape of the polycube, like that the dual space should enclose a slightly modified version of the input model in an intermediate coordinate space. Later, Wang et al. [9] mapped the three-dimensional model and the polycube to the canonical domain (e.g., sphere, euclidean plane or hyperbolic disc), looking for the map between them, which is guaranteed to be a diffeomorphism. However, in this scheme it is difficult to control where a feature on the three-dimensional model is mapped on the polycube. In their follow-up work, Wang et al. [10] proposed a map, where the users can specify the positions and the curvatures of all corner points., which works for shapes with simple geometry and topology, but is not feasible for complicated models because it is very tedious and error-prone manual process.

The polycube map quality (in terms of angle and area distortions) highly depends on its shape. Wan et al. [17] automatically optimized the polycube mapping in the sense

of area and angle distortion, at the price of not being controllable at all. Other authors [11], [12] construct the map by breaking the input model into smaller and simpler components and then use polycube primitives to approximate each one. For example, Reeb graphs [11] and harmonic functions [12] can be used to guide the shape segmentation. As heuristics are usually used in the segmentation and polycube approximation, these approaches may not work for complex models, and none of the existing algorithms allows to easily edit the maps.

A well designed polycube map may serve as a boundary constraint in volumetric parameterizations. Wang et al. [18] constructed trivariate polycube splines for volume data. Gregson et al. [19] generated an automatic all-hex mesh with a polycube domain. The parameterization quality is directly related to the quality of the surface polycube map.

Quadrangulation. Our work is also related to quadrangulation, with a number of excellent algorithms for the automatic or semiautomatic quadrangulation of arbitrary simplicial 2-manifolds [20], [21], [22], [23], [24], [25]. Myles et al. [26] describe a set of singularity alignment constraints that can design quadrangulations with a coarse base domain, providing a relatively simple user control. Tarini et al. [4] proposed a simple quad-domain generation method, which preserves the alignments and results in a good quality semiregular quad mesh. Ray et al. [27] proposed a periodic global parameterization that results in a semiregular quad mesh, introducing singularities to achieve a good quality quadrangulation. However, these methods do not preserve the intuitive 3D embedding of the given shape, and it is still not clear how to generate a cross domain among a set of shapes, not providing the features needed to allow the applications we propose. We refer interested readers to the survey by Bommes et al. [5].

Cross parameterization. Another related topic is cross parameterization, where the mapping between general surfaces of the same topology is built. A common approach is to parameterize the models over a common base mesh [28], [29],[30], [31], where meshes are split into matching patches that are parameterized on a common planar domain. A given set of matching feature points serve as patch corners and feature correspondences. In particular, Yeh et al. [32] proposed to place the correspondences interactively. All these approaches use points for feature correspondences, while our method supports sketches. In practice, we find it more intuitive to draw feature lines than to only place points.

Subdivision surfaces. Litke et al. [33] developed a technique to fit Catmull-Clark subdivision surfaces to a given base shape within a prescribed tolerance, based on the method of quasi-interpolation. In comparison, our method offers more flexibility in the mapping between the base and the target meshes. Cheng et al. [34] proposed a fitting method from the point cloud of a given surface. In contrast, we aim at generating a high-quality base mesh with a continuous parameterization for quad-based regular subdivision, something that no other method can provide. Recently, Panozzo et al. [35] presented a technique for the automatic construction of adaptive quad-based subdivision surfaces, based on a set of maps called *fitmaps*, which roughly estimate how well the mesh can be locally modeled by patches. This method is a "one-click" solution, while in ours the modeler has to sketch a base polycube to control the shape and connectivity of the coarse domain for subdivision. In the last few years, there has been a growing trend to use the tessellation capabilities of modern graphics hardware to generate high-resolution models from a coarse base mesh [36], [15]. Loop et al. [37], [14] presented a method for approximating subdivision surfaces with hardware-accelerated parametric patches. Our method presents a uniform, regular, and user-controllable quads-only mesh, which is a good basis for subdivision.

3 Editable Polycube Map

3.1 Overview

The objective of the proposed technique is, starting from a high-resolution model coming either from an artist or a three-dimensional scanner (plus its cleaning stage), to build a polycube map. This map will enable a set of user-friendly operations that take advantage, in a simple controllable way, using only quads, and without wasted space in texture space.

Our input is a high-resolution model plus a simple polycube representation, which is constructed manually with the sketch-based interface. By controlling the position and location of the cubes, the user has control over the number and location of the singular vertices in the resulting quads-only mesh. If the input model is non-genus-0, a preprocessing stage is performed to split the model into genus-0 patches (see Section 3.2.1). Then, we provide a guided feature-sketching interface enabling users to sketch controlling strokes in a natural manner (see Section 3.2.2). The user-specified features are sampled with points, from which we compute the shortest distance to each other, inducing a triangulation. To ensure the quality of the triangulation, we compute the geodesic triangulation on the polycube by using the correspondence of the user-specified features on the three-dimensional model and the polycube. This way, the resulting triangulation is smoother and more visually pleasing. After that, both shapes are segmented into genus-0 patches, keeping an identification between the segments in the polycube and in the high-resolution model. We compute the map between each pair of patches using a series of harmonic maps. Finally, a simple and effective global diffusion algorithm is applied to improve the map quality, at the price of introducing the possibility of small nonbijective deviations. Thus, we say the resulting map is quasi-bijective and satisfies the user-specified constraints.

As a result, the polycube map induced a quad-remeshed version of the high-resolution model, and the resulting patches come from tessellated square faces, which can generate a low-resolution version of the model, as shown in Fig. 2. This quad-only low-resolution model also has only vertices with a restricted valence number, which is important for GPU-subdivision displacement. See Section 4.4. It is important to mention that the user has full control over the process (shown in Fig. 5) through the sketched features and in further applications, as described in Section 4.

Fig. 2. Algorithmic pipeline. (a) On nongenus 0 surfaces, we require only two control points to guide an auto segmentation of the surfaces to genus 0 patches (Handles are cut by the corresponding autotunnels in M and P). (b) The user sketches a few strokes as constraints on the 3D model M and the polycube P (yellow dots, sample points on the sketched strokes; green dots, saddle points computed in the autotunnels). (c) Sample points are used as sources to compute the distance field (warm colors, shorter distances; cold colors, larger distances between sample points). (d) By the set of shortest paths between sample points, which do not cross the user strokes, and the autotunnels, a geodesic segmentation on M and P is computed, cutting both into *valid* single boundary genus-0 patches. (e) A constrained map is defined between each pair of patches, which allows to transfer the surface M_i to P_i, between all pairs of patches. (f) All P_i patches are seamlessly glued with a global smoothing operator. (g) The subdivision surface is reconstructed.

3.2 Constructing Polycube Map

Given the input model M and the corresponding user-built polycube P, we present a user-controllable framework to construct the map between them. See Fig. 2.

3.2.1 Topological Preprocessing

In this section, we introduce a topology-aware preprocessing algorithm to split M and P into genus-0 patches. The matching information is also computed in this stage.

The idea is motivated by the tunnel loop computation introduced by Dey et al. [38]. Given a closed surface of genus g, there are always g tunnel loops. In the following, we call tunnel loops simply as tunnels for short. Cutting along a tunnel eliminates the handle, so we can cut a high-genus model into a genus-0 surface by cutting all the handles along the corresponding tunnels. The geometry aware tunnel-computing methods [39] suggest that the tunnels can provide correspondence information in a surface map.

Given the input mesh M and its tunnels, if it has a genus greater than 1, the matching between the tunnels of the object and the polycube is a difficult challenge (see Section 5.4). We assume there is alignment and similarity between the object and the polycube, which in practice is reasonable as the polycube is constructed to mimic the input shape. In case both the object and the polycube are reasonably well aligned and not drastically twisted, we provide semiautomatic tunnel detection and matching. We adopt a hybrid strategy to achieve this matching between tunnels in the object and the polycube.

We compute the harmonic field in the object and polycube surfaces to depict their geometry shape. For this, the user only need to specify two corresponding control points on the input object and the polycube, as illustrated in Fig. 2a, which are also used as a part of the mapping constraints. In practice, we specify the two control points on two sides of the object and polycube shapes, to have all the tunnels between them. The computed harmonic fields arethree-dimensionalshape aware, with 0 value at one control point, and 1 at the other, smoothly increasing between them. Thus, the harmonic field values provide an ordering of the tunnel loops. Furthermore, two saddle points are computed for each tunnel loop corresponding to the loci of the control points (shown in green in Fig. 2b). These saddle points serve as additional accurate map constraints. Note that there could be tunnel loops at the same level of the harmonic field, under the mild assumption of alignment and similarity. We include the three-dimensional space coordinates to enhance the ordering of the tunnels to eliminate this ambiguity.

Our specific algorithm is as follows:

Step 1. Given the non-genus-0 model M and the corresponding polycube P, we compute the tunnels of each handle in the three-dimensional model and the polycube [39].

Step 2. Given the user-specified control points $p_0, p_1 \in M$, and the corresponding $p_0', p_1' \in P$, we compute the harmonic field f in M by setting $f(p_0) = 0, f(p_1) = 1$ as the boundary condition, and similarly for f' in P by setting $f'(p_0)' = 0, f'(p_1)' = 1$.

Step 3. We compute the two saddle points of each tunnel by selecting the points with minimum and maximum values of the computed harmonic fields. Under the assumption of similarity, the computed harmonic fields in the object and the polycube are similarly distributed.

Step 4. The user is required to provide *approximate* strokes on both the input model and the polycube, which may not match exactly. We match pairs of tunnels by ordering them in ascending order by the average harmonic field value of all their points. If there are tunnels in the same level of the harmonic field, a further ordering is performed by their 3D x, y or z space coordinates, in that order. We set the threshold based on the maximum number of blocks n in each axis as $1/n$. For instance, in our experiments with the Buddha model, the threshold was selected to be 0.05 times the maximum of the harmonic value, or the length of the maximum direction of the bounding box.

Step 5. We cut M and P along the tunnels, so models with genus g are cut into genus-0 surfaces with $2g$ holes. Each tunnel is split into two circles and the two saddle points in the tunnel are split into four points. We match the split circles and saddle points by their consistent orientation.

After this preprocessing stage, the tunnel loops also serve as strokes to segment the object and the polycube (see Fig. 2d), and the saddle points serve as sample points for the polycube mapping algorithm described in Section 3.2.3.

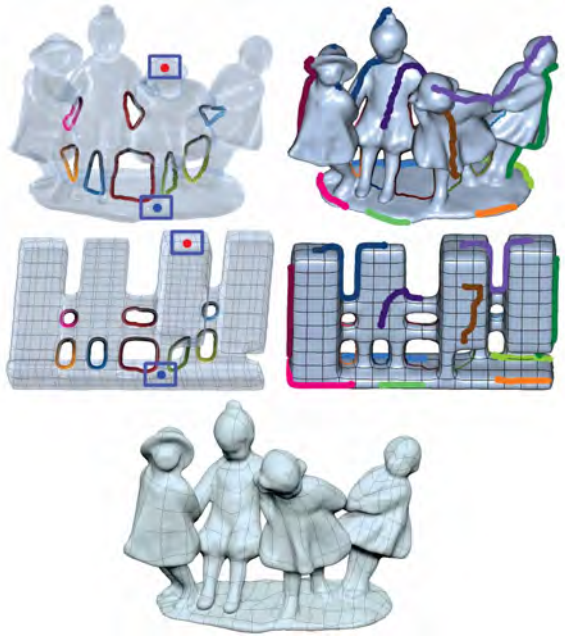


Fig. 3. Children model editing: Using two user points, and 13 user strokes. Bottom: The polycube map result.

3.2.2 Guided Stroke Drawing Interface

Feature Guided Stroke Drawing. In our framework, users are allowed to freely sketch the features on the models using a direct sketching interface, which allows a natural metaphor by projecting strokes from screen space onto the 3D surface. Observe that these strokes do not need to be very accurate, as our system uses them as *hints* in the parameterization process. However, the consistency of stroke order between the object and Polycube must be guaranteed, which can be checked with our drawing interface (see below). We should remark that the accuracy needed depends on the application. For example, in kit-basing the accuracy is more important near the part boundaries and less important in the interior.

On the object surface, the detailed geometry features are visualized and could be captured intuitively with strokes. Since the strokes serve as boundary conditions in the mapping stage, strokes on the shape features lead to more accurate mapping results that preserve the features well (see Fig. 4). The user sketches the same (reduced) number of features on the three-dimensional model M and on the polycube P.

Stroke Types. We support three types of stroke shapes: *curved line*, *closed loop*, and *crossed strokes*. *Curved line strokes* are the fundamental strokes for shape feature description to specify some features (e.g., see the strokes in Fig. 3). *Closed loop strokes* can be powerful for loop-shaped features on the objects' surface as they can be mapped onto loop shapes in the polycube domain (e.g., the mouth stroke in Fig. 9). *Crossed strokes* are a combination of several crossed *curved line strokes*. In practice, the strokes are split into connected *curved line strokes* by their crossing points (e.g., the eyes and nose strokes in Fig. 9).

Fig. 4. The users can take full control of the polycube map by simple sketches. The thumbnails show the sketched constraints.

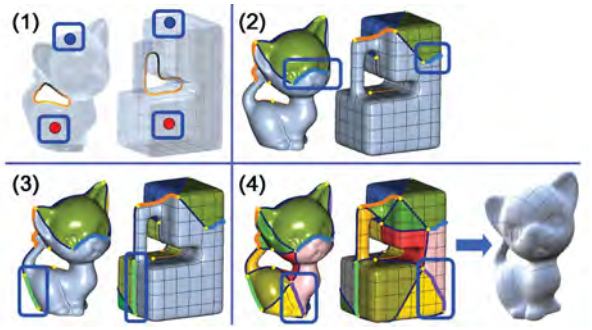


Fig. 5. Interactive editing of Kitten (genus 1): 1) Initially, the user draws two control points and the autotunnels are computed. 2) Then, he draws three strokes above the object and the polycube. As a result, a set of color parts get segmented as *valid* single boundary g0 patches. 3) The remaining surface, shown in gray, means that additional strokes are further required. 4) Finally, with five user strokes, all the surface of both, object and polycube, is consistently segmented and the polycube map is computed.

Consistency and Topology Guided Stroke Drawing. The feature stroke set in the object must be consistent with the three-dimensionalset in the polycube domain in the sense of having the same number of strokes and a similar spatial distribution. We present a consistency checking scheme to guide the consistency-aware stroke drawing. Specifically, after a pair of strokes are sketched in the object mesh and the polycube, our system performs the segmentation process described in Section 3.2.3 and validates the correspondence between the segmented patches. We perform the segmentation in an incremental manner that only computes the newly added geodesic paths in each iteration. If the segmentation results are not consistent, the system alerts that the added strokes could be nontopologically consistent. Thanks to our editable framework, the user is always given the possibility to edit the strokes dynamically with the consistency guidance.

Furthermore, the purpose of the segmentation stage (described in Section 3.2.3) is to divide the mesh into *genus-0 single-boundary patches*. In the following, we refer to a patch of genus-0 with a single-boundary as a *valid patch* (e.g., Fig. 2e illustrates a pair of *valid patches* M_i and P_i). On the interactive segmentation result, the *valid segmented patches* are shown with same consistent colors in the object and the polycube domain (see Fig. 5(2)). The remaining parts are still gray-blue colored, suggesting that additional strokes are required to segment this region. A complete example of the interactive stroke drawing process is illustrated in Fig. 5.

3.2.3 Segmentation

Our segmentation algorithm is as follows:

Step 1. Given the user-specified sketches, $\gamma_i \in M$, $\gamma_i' \in P$, $i = 1, \ldots, n$, we sample them with a set of points. Let $S = \{p_j\}_{j=1}^{m}$ and $S' = \{p_j'\}_{j=1}^{m}$ denote the sample points on M and P, respectively. Given the length of a stroke L and the

shortest distance between strokes l, m is computed as $m = floor(L/l) + 2$, to include both stroke endpoints.

Step 2. We use $p_j \in M$, $j = 1, \ldots, m$, as source points and compute the shortest distance for every point on M using a multisource Dijkstra's algorithm. This way, each vertex v is associated with a distance $d(v, p_j)$, where p_j is the closest sample point to v. Let $c(v) \in S$ be the closest sample point of vertex v.

Step 3. We consider each mesh edge $e_{ij} = (v_i, v_j)$, where v_i and v_j are neighboring mesh vertices. If $c(v_i) \neq c(v_j)$, let $s_1 = c(v_i)$ and $s_2 = c(v_j)$ be the two sample points. We mark the two sample points s_1 and s_2 as neighbors.

Step 4. For every pair of sample points s_i and s_j, which are marked as neighbors, we compute the geodesic path between them. It can be shown that two geodesic paths can only meet at the two ending sample points. Then on the polycube P, we compute the geodesic path between s_i' and s_j'. If the geodesic paths in object surface and polycube surface do not intersect with any user input strokes, we add them into the *segmentation path sets*.

Step 5. We segment M and P along the computed *segmentation path sets*.

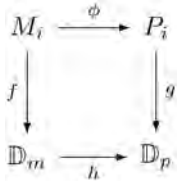
In the above algorithm, we compute a distance field on M using the user sketched constraints, naturally inducing a segmentation. See Fig. 2e. However, note that a few geodesic paths intersecting with user strokes and auto-tunnels are canceled, so not all the result patches are triangle-shaped.

3.2.4 Constrained Map

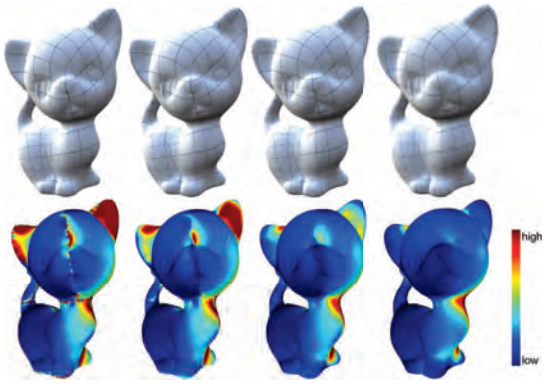
Let $P_i \in P$ and $M_i \in M$ be a pair of segmented patches, i.e., genus-0 surfaces with single boundaries. We want to find a bijective and smooth map $\phi : M_i \rightarrow P_i$. See Fig. 2e. We first parameterize M_i to the unit disc using a harmonic map, i.e., $f : M_i \rightarrow \mathbb{D}_m$ such that $\triangle f = 0$ and f maps the boundary of M_i to the boundary of $\mathbb{D}_m$ using the arc length parameterization, $f(\partial M_i) = \partial \mathbb{D}_m$. We also parameterize P_i to the unit disc using a harmonic map $g : P_i \rightarrow \mathbb{D}_p$. More details of discrete harmonic map can be found at [40].

We then seek a smooth map between the two unit discs $h : \mathbb{D}_m \rightarrow \mathbb{D}_p$. This map h is also computed using a harmonic map $\triangle h = 0$ and the boundary condition is set as follows: Let s_1, s_2, and s_3 be the sample points on ∂M_i, and $f(s_j) \in \partial \mathbb{D}_m$, $j = 0, 1, 2$ be the images on the boundary of unit disc. Similarly, let $g(s_j') \in \partial \mathbb{D}_p$ be the images of the sample points $s_j' \in P_i$. Then, we require the function h to map $f(s_j)$ to $g(s_j')$, i.e., $h \circ f(s_j) = g(s_j')$, $j = 0, 1, 2$. The images for the points between $f(s_j)$ and $f(s_{(j+1)\%3})$, $j = 0, 1, 2$, are computed using an arc length parameterization.

The polycube parameterization is given by the composite map $\phi = f \circ h \circ g^{-1}$ as shown in the following commutative diagram:



Finally, we glue the piecewise maps $\phi_i : P_i \rightarrow M_i$ together as illustrated in Fig. 2f.



(a) initial map 1.151 (b) 100 iterations 1.125 (c) 500 iterations 1.105 (d) 2000 iterations 1.087

Fig. 6. Smoothing the polycube map. The initial map has only C^0 continuity along the segmentation curves and user-specified features. Thus, one can clearly see the large distortion and lack of smoothness in (a). Using the Laplacian smoothing algorithm, the distortion smoothly spreads out over the entire model, see (b)-(d). The values are the angle distortions. The step length is $\delta = 0.05$.

3.2.5 Globally Smooth Map

With the above boundary conditions, the maps are consistent along the boundaries, which can be seamlessly glued. The resulting map $\cup_i \phi_i$ guarantees to be C^0 continuous along the segmentation boundaries.

We use Laplacian smoothing [41] to improve the continuity along the segmentation boundaries. Given the initial polycube map $\phi : P \rightarrow M$, let $p' = \phi(p) \in M$ denote the image of $p \in P$. We solve the following diffusion function:

$$\frac{\partial p'(t)}{\partial t} = -(\triangle p'(t))_{\|}, \tag{1}$$

where $\mathbf{v}_{\|} = \mathbf{v} - (\mathbf{v}, \mathbf{n})\mathbf{n}$ is the tangent component of $\mathbf{v}$, $\mathbf{n}$ is the normal vector, $(,)$ is the dot product, and t is the diffusion time.

Since the given polycube map ϕ is represented in a quadrilateral mesh induced by the tessellation of P in which each quad is a square, we use the following Laplace operator:

$$\triangle p' = p' - \frac{1}{m} \sum_{pq \; is \; edge} q', \tag{2}$$

where m is the valence of p, and q is the one-ring neighbor vertex of p. Observe that this is the only step in our pipeline that might break the bijectivity of the mapping, although we did not observe any problem in all our experiments.

The above diffusion equation can be easily solved using the Euler method. We set $\delta = 0.05$ in our experiments.

Following Degener et al. [42] and Tarini et al. [3], we measure the map quality in terms of angle and area distortions which integrate and normalize the values $\sigma_1\sigma_2 + 1/\sigma_1\sigma_2$ and $\sigma_1/\sigma_2 + \sigma_2/\sigma_1$, where σ_1 and σ_2 are the singular values of the Jacobian matrix of ϕ. $\epsilon_{angle} = \epsilon_{area} = 1$ when the map ϕ is isometric. As shown in Fig. 6, our method leads to visually pleasing results in only a few hundred iterations.

It is worth mentioning that the user constraints are not guaranteed to be preserved after the global smooth step, but, in our framework, the *approximate* feature stokes

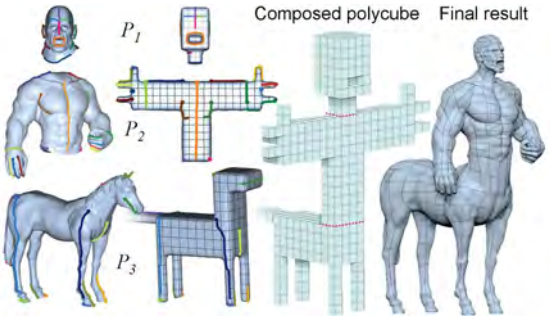


Fig. 7. Centaur kit-bashing composition example. Left: A horse and a male head and torso are polycube mapped. Right: The parts of interest are cut and assembled together to generate the centaur subdivision model.

are expected as soft guides. Thus, we consider that the proposed methods are good enough for our purpose. Better alternatives [43], [44], [25], [4], [26] could be employed, strictly guaranteeing the absence of fold overs. However, they were not specifically designed for polycube maps, so efforts should be made to integrate them into our editable framework.

4 Applications

In this section, we present some applications that are possible thanks to the inherent simplicity of the polycubes, the sketch-based correspondences, and the easier and more intuitive nature of the parameterization (i.e., polycubes are easy to glue on each other, to cut open, to visualize, and so on) to provide simpler, more intuitive ways of interacting with the model. Also, they benefit from the dual nature of the polycube parameterization, and the generated quad-based parameterized polycube representation $S_{[0-n]}$. Here, we present four of these applications, namely kit-bashing, shape morphing/animation, painting over the parameterization domain, and GPU-based subdivision displacement.

4.1 Kit-Bashing

Kit-bashing is a common practice among artists, who tend to accumulate models from previous projects and reuse parts of them to start building any new one. This technique is particularly used for human-like characters. Cut and paste methods are proposed in Funkhouser et al. [45] and Yu et al. [46]. Huang et al. [47] proposed a merging boundary optimization for better matching. Sharf et al. presented SnapPaste [48] enabling an interactive framework in a drag-and-snap manner. Kreavoy et al. [49] computed the cut boundary automatically after the merging operation is specified for better boundary shape. And *blocks* (generalized cuboid shapes) from Leblanc et al. [50] are introduced into this problem as a modeling primitive for composition operations to create complex objects.

With our polycube mapping technique, the artist can have a library of parts already mapped and textured, like for instance, the legs or arms from a previous model. Then, parts of the model (e.g., the torso and the legs) can be assembled in the polycube domain supporting integer scaling and rotation factors, and a free alignment in object space, together preserving the connectivity of the base mesh used for subdivision, as illustrated in Fig. 7.

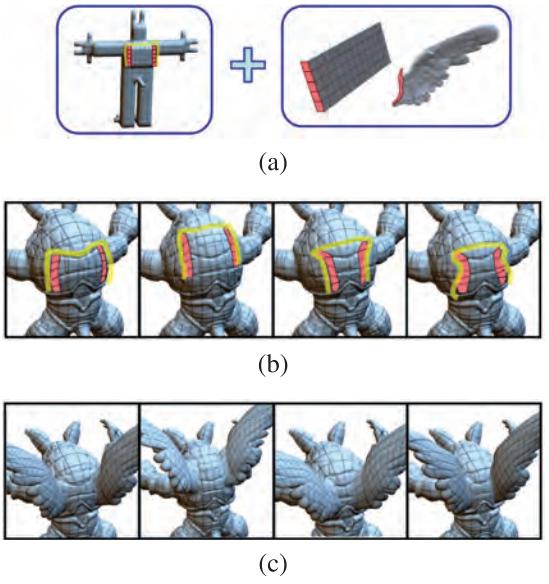


Fig. 8. Armadillo-wings kit-bashing composition: The user took the polycube mapped wings from the Lucy model and stitched them onto the back of the armadillo model. (a) The user selects the parts to be glued. (b) One curved stroke is added to fine-tune the wings placement. (c) A poisson-based smoothing is used between the gluing parts to obtain the final result.

As the parts to assemble have different atlases, creating the new unified atlas only means merging the respective patches from each texture map into a new single one, recomputing the seam boundaries, and adequately combining the new neighboring uv coordinates of each base level patch.

In Fig. 8, we can see an example where the user stitched a couple of wings from the Lucy model to the back of the Armadillo model. As the models were processed beforehand, the user just selected a few cube faces from both polycubes to cut the parts of interest, perfectly matching in parametric domain. Thanks to the quasi bijection between P and M, the edges that stitched to each other on the polycube maps are identified with their corresponding edges in the original 3D model. The next step is to align the two original parts. We minimized a simple energy function consisting of the l^2 distance between the vertices. If users are not satisfied with the final result, they are able to further adjust the piece positions. If a larger control is needed, the user is always free to place feature lines at any place in the object and the polycube, as illustrated in Fig. 8b, and these features would be used in the map editing step. Once aligned, we glue both parts by collapsing each pair of matching vertices, replacing them by the midpoint vertex. As this may cause crisp edges, we immediately perform a local poisson-based smoothing step [51], as shown in Fig. 8c.

4.2 Morphing

Another application of the editable polycube map approach is the possibility of morphing models to provide an intuitive way to define new models by fusing of a set of previous polycube-mapped models. Here, we use the polycube maps as help, as they are easier to manipulate and there are fewer possible polycube maps rather than generic quad-based domains. The proposed application is based on a same *shared* polycube base domain, where the user mapped

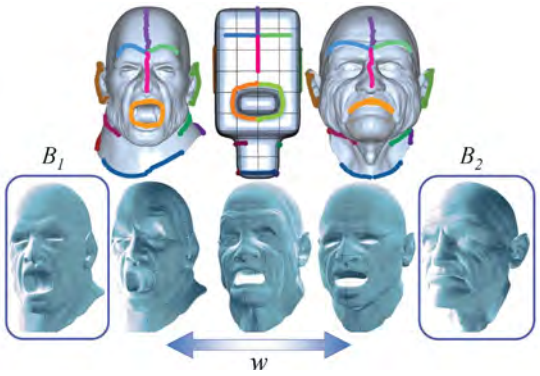


Fig. 9. Top: The input surfaces are polycube mapped with the same base *shared domain* to generate the morphs B_1 and B_2. Bottom: New models are obtained with brush operations to define the weight values for the linear combination of B_1 and B_2.

different input source shapes by a set of strokes as described in Section 3 and illustrated in Fig. 9. The polycube domain then acts as the *combination space* of previously mapped input models. The morphing formulation represents the surface as a linear combination of the set of shapes (i.e., like morph targets) as,

$$o = p + \sum_{i=0}^{n} B_i w_i, \tag{3}$$

displacing the original positions p of the reference polycube base domain vertices with respect to the set of morph displacement textures B_i. The weight textures w_i are used to obtain the morph result o as the combined vector displacements in a final texture map.

The morphs B_i and weight textures w_i, are unfolded in texture space on a per-face basis. The morph textures are combined by weights w_i defined by user brush strokes with alpha masks or smoothstep operators. The per-face morph texture implementation is encoded in a texture array, where we force the same texture values to all corners of the mesh with an irregular valence [52]. However, corners with valence different than 4 cannot be exactly matched with a bilinear interpolation. For that, we perform a simple process over the mesh connectivity to determine, which shared corners are irregular. Then, for each shared group of faces for each irregular corner, we fix the morph texture. First, by determining the correct value to be stored exactly at that corner in the shared per-face texture (e.g., by simple average). Then, the correction is propagated to every mipmap level, so that the shared corner has always the same value.

4.3 Painting over the Parameterization Domain

Another interesting application of our dual parameterization scheme is what we called *Painting over the parameterization domain*, where a user can paint the model directly over the three-dimensional model surface, the polycube, or any morph between the two. This can be considered a valuable addition to the tools proposed by Hanrahan and Haeberli [53], as the user can use the polycube map to easily access and paint concave regions that would be too difficult to paint otherwise (see Figs. 10 and 11). Another advantage is that cubes afford a semantic partition of the model so that users can expose occluded parts of the model by selecting hiding and unhiding cubes. Also, the parts to assemble for kit-bashing can be textured beforehand, and our painting tools help with the finishing touches, using the smoothing operation to blend the colors at the joints.

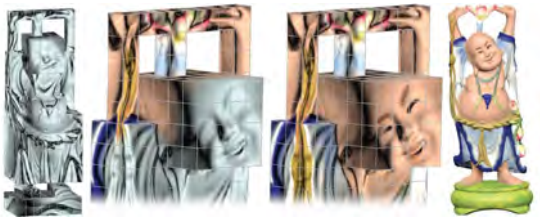


Fig. 10. Painting on Happy Buddha: Some parts (e.g., cloth) are very difficult to paint due to the occlusion and concave geometry, but the polycubed equivalent is more accessible and easier to paint.

4.4 GPU-Based Subdivision Displacement

We can generate a parameterized base mesh for subdivision, which can be effectively used in many graphics application like computer-generated movies or real-time applications.

As the quasi bijection has already been established between the original surface and the polycube, the surface can be processed to define a quad-based subdivision surface $S_{[0-n]}$. The base subdivision mesh S_0 will have the number of quad faces of the coarse polycube domain. The displacements defined between S_0 and the polycube-mapped mesh S_n are stored as a vector field *(direction + length)* in texture maps for the GPU-subdivision displacement applications described in Section 4. The following steps are performed:

Step 1. Reverse subdivision. First, we recover the coarse quads from the refined polycube-mapped surface S_n to extract the subdivision base mesh S_0 (see both in Fig. 16(bottom)): We reconstruct the Catmull-Clark Subdivision with the algorithm described in [54], which allows to obtain a valid scheme $S_{[0-n]}$.

Step 2. Packing in texture space. Here, we need to create a two-dimensional texture atlas, which contains all the externally visible polycube faces, which is an easy task as the previous steps already kept only the visible faces that are not shared by more than one cube. Then, all polycube map patches are packed as consecutive squares in the

Fig. 11. Painting on Arthur's Head: The parts inside the mouth are very difficult even impossible to paint due to the occlusion, but the polycubed equivalent is much more accessible and easier to paint. The packed textures are rendered in the bottom right.

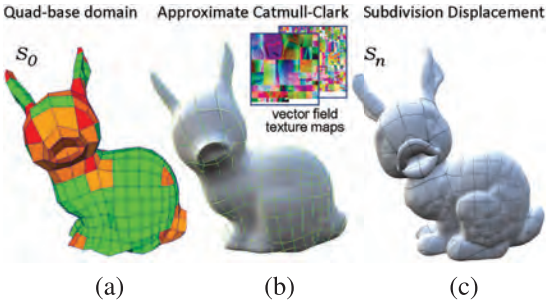


Fig. 12. Bunny-duck kitbashed model shown with GPU-based subdivision displacement. (a) Our base mesh (subdivision level 0). Green patches contain only valence four vertices, and the orange levels show the number of extraordinary vertices (red patches having the largest number). (b) Tessellation with approximate Catmull-Clark subdivision, with the normal and displacement vector field texture maps. (c) the surface with GPU-based subdivision displacement (top right: The isoparametric curves).

texture atlas. For each patch S_0, all the interior subpatches of S_n contained in a base patch of S_0 are assigned their respective texture coordinates in the atlas patch, as shown in Fig. 12b.

Step 3. Raster displacement data. We rasterize each high-resolution patch into its associated base patch as a vector field displacement in texture space, also saving the other required information as normals maps, occlusion maps, and so on.

For real-time applications, we use the approximation algorithm by Loop and Schaefer [37], following the implementation described by Castaño [55], [56], [57]. Polycube maps are well suited for this as we only generate vertices with valences 3, 4, 5 or 6, as shown in Fig. 12a. Geometry image-based tessellation is also an interesting option that becomes a subcase of our strategy, mainly intended for static and not too large objects. With geometry images, no texture coordinate-specific information is required, sufficing to compute the ownership data in the respective shaders.

5 Results and Discussion

5.1 Results

We tested our method with a wide range of models with various geometry and topology configurations, as shown in Fig. 17. Additionally, Table 2 shows the statistics of our

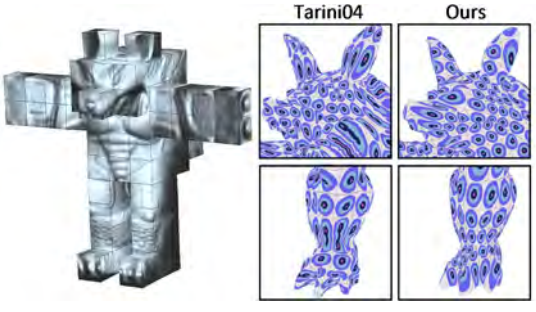


Fig. 13. Comparison of the bijectivity between the editable polycube map and the original polycube implementation. Left, the original proposal [3]. Top row: Insets of the Armadillo head. Bottom: Insets of its feet. Nonbijectivity in [3] results in a dependence of multiple mesh points with a single texture point.

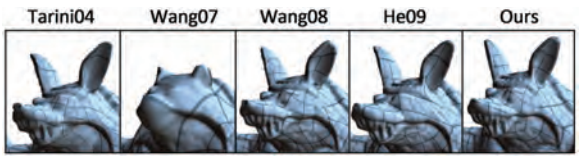
TABLE 1
Comparison of Polycube Map Construction Methods

Features	[3]	[9]	[10]	[11]	[12]	[19]	Our
Map quality	●	◐	◐	●	●	◐	●
Bijection	○	●	●	●	●	◐	●
User control	○	○	◐	○	○	○	●
Editing	○	○	○	○	○	○	●
PC construction	○	○	○	●	●	●	○
Automatic	●	○	○	●	●	●	●
Large models	●	○	○	○	◐	●	●
General topology	○	○	○	○	◐	●	◐

Symbols: ○ Good, ○ fair, and ○ poor.

experiments. A good global bijective parameterization is very important for modeling and texturing, as typical projections used in Tarini et al. [3] unavoidably have problems because two vertices on the same projection line share the same image, something that produces artifacts in texture mapping and painting, like the ones that are clearly visible in Fig. 13. Although our approach provides a quasi bijection, we have found that in practice all our experiments showed a perfect bijective mapping.

We also quantitatively compared our methods with existing methods [3], [9], [10], [12] and [19], as shown in Table 1 and Figs. 13 and 14. Tarini et al. [3] are efficient for large scale models, but it can not guarantee the bijectivity due to the used projections. Wang et al. [9] computed the polycube map in an intrinsic way by conformally parameterizing M and P to canonical domains and then seeking the map between them. For a genus-0 shape with complex geometry (like the Armadillo), the conformal spherical parameterization has very large area distortion on the elongated parts (e.g., arms, legs, and tail), so the induced map will show a large area distortion with uneven sampling. Wang et al. [10] required to manually specify the images of the polycube corners and edges on the three-dimensional model and then compute the map for each polycube face individually. The user-defined polycube structures on M can be considered as our constraints. However, it is very tedious and error prone to specify them manually if the polycube is complicated, which precludes its usage for large-scale models. Automatic approaches [11], [12] use heuristics and may not work well for complex models. In He et al. [12], both M and P are segmented by horizontal cutting planes, and the cutting locus serve as constraints. Thus, the generated map is orientation dependent. Due to the complex geometry of the Armadillo,



[angle, area] distortion:
[1.32, 1.22] [1.35, 88.62] [1.29, 1.24] [1.25, 1.23] [1.16, 1.18]

Fig. 14. Comparisons of our method with Tarini et al. [3], Wang et al. [9], Wang et al. [58], and He et al. [12], with the same polycube. Our method is more intuitive and flexible in terms of user control and editing, and can generate a better quality map. The values are the angle and area distortions.

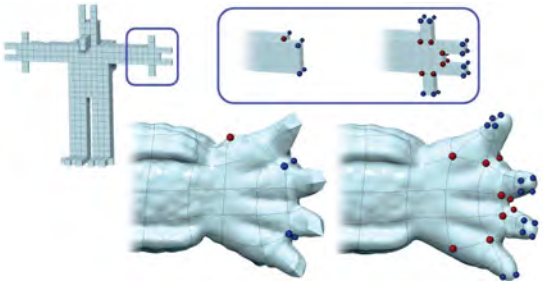


Fig. 15. Armadillo hand polycube editing: The control of the base polycube detail and the type and location of irregular vertices are both important for the final quality, because they are intrinsically linked to the geometric features on the surface. Irregular vertices shown in blue and red, are valence 3 and 5, respectively. Left: A coarser polycube hand without fingers has less irregular vertices but limit the quality. Right: A more detailed polycube introduces more irregular vertices but captures the details with higher fidelity.

e.g., the arms are not axis aligned, there are large distortions on the upper arms and shoulder (see Fig. 14). Gregson [19] proposed an automatic method to generate both the polycube and the mapping. However, this method does not produce a bijection, nor provides control on the parameterization fine details. Compared to the existing approaches, our quasi-bijective method is more intuitive and flexible in terms of control and editing, and can generate better quality polycube maps. Also, if the user provides a few intuitive guiding sketches, the whole process becomes automatic and practical for complex models.

We want to emphasize the differences between our method and He et al. [12]. They first segmented the three-dimensional model and polycube by horizontal planes, computing a map between each pair of segmented components, and finally smoothed the map by solving a harmonic map for the entire shape. Although using the same divide-and-conquer strategy, our method is completely different in all the following steps: First, because the cutting loci are also the constraints of the map, He et al. [12] can only map the horizontal, planar features from the three-dimensional model. Our segmentation allows the users to cut the model by arbitrary closed curves, resulting in more flexible and meaningful constraints. Furthermore, our method supports the user-control and editing operations not supported by them. Second, He et al. [12] mapped the segmented components to multiple connected rings by an uniformization metric, and then computed a harmonic map between the rings. It is known that computing this metric is a nonlinear time-consuming process. Our method computes the harmonic map between two topological disks, so it is much more efficient. Finally, rather than solving a harmonic map for the entire shape, we smoothed the whole map by an iterative method to diffuse the angle distortion. As shown in Fig. 6, our method is very effective and leads to a high-quality map with only a few hundred iterations, as only very simple vertex operations are involved in each iteration.

5.2 Tradeoff between Accuracy and Regularity

It is known that the distortions of polycube parameterizations highly depend on the shape of the polycube. In general, the more accurate its representation, the lower

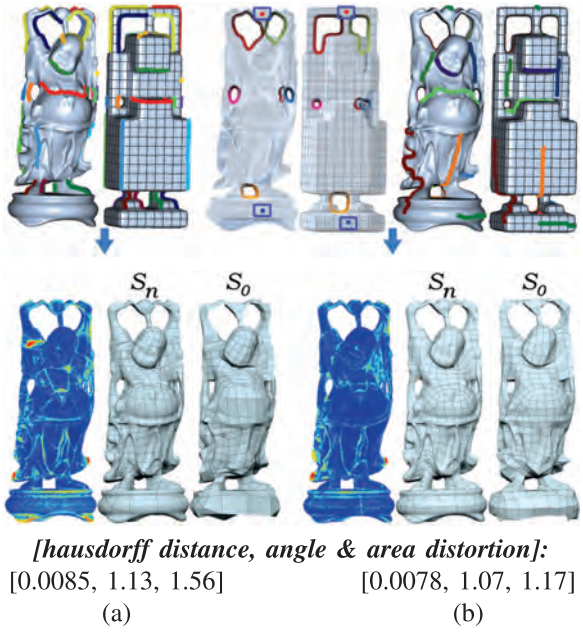


Fig. 16. Buddha model editing. Left: Without the handles autosegmentation, a novice user specified 34 strokes. Right: Using two user control points for the autosegmentation and only 15 strokes, the user obtained better quality results. Bottom: The subdivided model with Hausdorff w.r.t. bounding box diagonal between M and S_n, area, and angle distortion in S_n.

distortion of the map. However, the price to pay is the larger number of extraordinary points (polycube corners), so there is a tradeoff between quality and complexity. Through our experiments, we observed that, for shapes with extruding regions, like the Armadillo's fingers (see Fig. 15), it is a good idea to design an accurate polycube to model these features. In our framework, we leave the choice to the user. It is worth noting that compared to the results of other approaches, the polycube maps of our method have higher quality using the same coarse polycube base (See Fig. 14).

5.3 Stroke Drawing

The user input strokes play two important roles in our framework. First, they induce the segmentations: The shape of segmented patches is controlled by the shape of input strokes, which are validated during the mapping stage to ensure they provide a consistent topology in the object and the polycube. Second, the input strokes also serve as the boundary conditions in the mapping stage. Thus, the loci of input strokes brings to users full control of the mapping result (see Fig. 4).

The requirements and importance of the input strokes might challenge novice users. In our observation, fortunately, the correspondences match with user's perception and can be easily managed after a few trials. In general, users add strokes around relevant detailed features, like near the ears, eyes, and mouth of face models, which usually leads to satisfactory mapping results, which preserve well those features, as illustrated in Fig. 9.

5.4 Automatic Tunnel Splitting

The proposed topological preprocessing stage (described in Section 3.2.2 converts the input high-genus mesh into

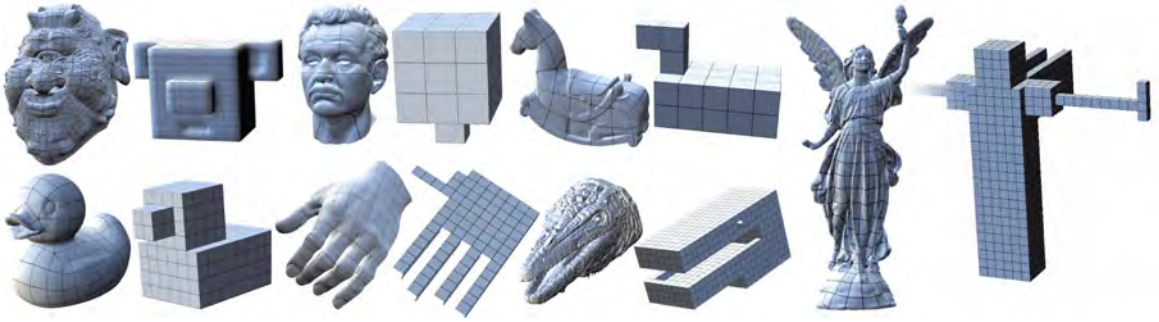


Fig. 17. More polycube mapping results: Ogre, Arthur's head, isidore horse, Lucy, duck, Arthur's hand, and tylo's head.

genus-0, but the user can work without it by carefully splitting all the handles. We evaluate the performance of automatic tunnel splitting by comparing polycube mappings with and without it (see Fig. 16). A novice user specified 34 strokes on Buddha without the autocomputed tunnel loops. Given the guidance of computed tunnel loops, 15 strokes are required by the same user. In general, users are required to use as few strokes as possible to keep the similarity, with higher priority given to the number of strokes. In the two cases, even when there is consistence between the two sets of strokes (e.g., on the chin or the belly of the Buddha model), the main difference comes from the automatic tunnel computation, which is often better than the user-defined strokes. Without autotunneling, the user is more concerned to segment the tunnels into topological disks. In comparison, with autotunneling, the user can focus on the quality of the guiding strokes because that all the tunnels have been cut and the model is already genus 0. In conclusion, the differences of the two cases suggests that autotunneling helps the user interaction a lot and improves the quality of the polycube map.

Our algorithm would fail if the alignment and similarity assumptions do not hold. If only the alignment assumption fails, the polycube or the input object could be aligned with simple rotation operations in our system. But the similarity assumption can fail if, for example, the input object is too much twisted, and the polycube can not be constructed keeping similarity and simplicity at the same time. In this case, tunnel loops are computed without the correspondence information, and users are required to manually input them by specifying one point at each tunnel loop in a corresponding order in the object and polycube. The specified points also serve as the boundary condition in tunnel loops map. Thus, $2g$ point specifying operations are needed for model with genus g. It is still much easier than drawing $2g$ loops strokes to split g tunnels manually. Furthermore, in this extremal case, there is no currently known perfect solution. For example, Reeb Graph [59] might be a potential solution. But it would fail in cases of symmetric shapes. More user interaction or information of 3D space alignments are also required. A complete classification of correspondence methods is described in [60]. For general case, we believe that the assumption of alignment and similarity are mild, and our algorithm would suffice to solve most cases of this problem.

6 Conclusions

We have proposed a new method that aims to improve user control by developing a very practical, intuitive and efficient system. From a general mesh with optional featured approximate sketches, we create a user-controllable quads-only mesh with a globally smooth parameterization. The method guarantees that the computed map is

TABLE 2
Statistics of Experimental Results

	genus	#△ in M	#□ in S_0	S_n levels	n_c	n_f	$time$	angle distortion	area distortion	hausdorff M - S_n
Armadillo	0	346K	1012	6	82	14	148	1.16	1.18	0.0054
Arthur's hand	0	307K	239	5	56	7	78	1.06	1.14	0.0091
Arthur's head	0	252K	57	7	12	5	76	1.02	1.13	0.0050
Buddha	6	114K	818	5	117	15	47	1.07	1.17	0.0078
Bunny	0	144K	452	5	22	6	35	1.01	1.13	0.0072
Children	8	269K	526	5	110	13	106	1.20	1.22	0.0089
Duck	0	101K	288	5	24	5	37	1.09	1.16	0.0041
El Oso head	0	100K	150	5	24	11	40	1.13	1.22	0.0076
Holes 3	3	64K	68	5	32	1	13	1.10	1.18	0.0045
Horse	0	67K	436	5	28	8	11	1.04	1.07	0.0069
Isidore horse	0	151K	74	8	14	6	48	1.01	1.07	0.0061
Kitten	1	274K	272	5	28	6	83	1.16	1.27	0.0055
Lucy	0	526K	980	5	38	15	210	1.12	1.23	0.0091
Ogre	0	478K	694	5	124	9	68	1.14	1.29	0.0089
Skull	0	52K	24	7	8	3	7	1.02	1.06	0.0035
Tylo head	0	500K	920	5	32	9	194	1.10	1.25	0.0083

Notation used in the table: genus: genus of M; #△ in M: # of triangles in M; # □ in S_0: # of squares in S_0; S_n levels: # of subdivision levels for the high-resolution polycube; n_c: # of corners in P; n_f: # of user-specified features; $time$: Time measured in seconds; angle distortion: Angle distortion metric; area distortion: Area distortion metric; hausdorff M - S_n: hausdorff w.r.t. bounding box diagonal between M and S_n.

quasi bijective and conformal except at a finite number of extraordinary points (the polycube corners). Actually, in all our experiments the resulting mapping is bijective, but the Laplacian-based smoothing step might potentially break this property. As listed in Table 1, our method has most of the user-desired features: The created mesh is uniform, regular, and is generated from the base polycube mesh in an automatic manner. During the editing stage, the user is able to control the number of patches in the base mesh of the subdivision by the construction of the base polycube. Then, the user still has the possibility of fully control the process by sketching approximate correspondence lines between the high-resolution model and the polycube. Given the quasi bijection defined between the model and the polycube and the intuitive nature of the later, this results in a number of applications that greatly benefit from this simplicity (e.g., the Lego-like nature of kit-bashing for assembling model parts). The simplicity of the polycubes, which are easier and more intuitive to glue on each other, to cut open, to visualize, and so on, allows a number of applications with a simplicity, which is unprecedented in the literature. On the more technical side, as a result of the reduced number of topology combinations, we are able to have both a small memory footprint and a reduced texture fetching bandwidth, which strongly improves runtime performance of the tessellated result with modern hardware using instanced tessellation or the programmable units in Shader Model 5 hardware.

6.1 Limitations and Future Works

The proposed framework has limitations. Sharp thin features, like the sharp boundaries of man-made mechanical objects may not be preserved well in our framework. There are various reasons: The sketched input strokes are only expected to approximate the sharp features. Also, the proposed global smoothing in Section 3.2.5 is not feature aware and the sharp features will be undesirably smoothed. A simple extension to improve the sketch interface would be to aid the user snapping strokes to the sharp features, and, at the same time, use these sharp stroke features as constraints in the global smoothing stage. The tunnel loops matching between the object and the polycube might fail if the input mesh has complicated geometry or topology, e.g., twisted shape with handles. In this case, it is hard to align the object and corresponding polycube. More precise polycube design is able to alleviate this problem. But designing a precise polycube requires more efforts and introduces more singularities. One direction of our future work is to investigate more robust tunnel loops matching algorithm.

Acknowledgments

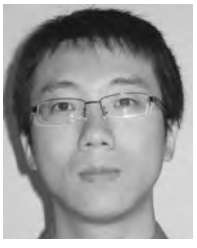
J. Xia, Y. He, and S.-Q. Xin are partially supported by AcRF 69/07 and NRF2008IDM-IDM004-006. J. Xia is partially supported by the freedom explore Program of Central South University(NO.2012QNZT058) and Doctoral Fund of Ministry of Education of China (NO. 20120162120019). I. Garcia and G. Patow are partially supported by Grant TIN2010-20590-C02-02 from Ministerio de Ciencia e Innovación, Spain. The authors would like to thank the anonymous reviewers for their comments, I. Castaño (NVIDIA Corporation) and F. González for support, and the Stanford three-dimensional Scanning Repository, AIM@Shape, K. Crane, H. Alvarado, and J. Johnson-Mortimer for the three-dimensional models. Part of this work has been presented in ACM Symposium on Interactive three-dimensional Graphics and Games (I3D '11) [1]. Jiazhi Xia is the corresponding author.

References

[1] J. Xia, I. Garcia, Y. He, S.-Q. Xin, and G. Patow, "Editable Polycube Map for GPU-Based Subdivision Surfaces," *Proc. Symp. Interactive 3D Graphics and Games (I3D '11)*, pp. 151-158, 2011.
[2] G. Oliverio, *Maya 8: Character Modeling*. Jones & Bartlett Publishers, 2006.
[3] M. Tarini, K. Hormann, P. Cignoni, and C. Montani, "PolyCube-Maps," *Proc. ACM SIGGRAPH '04*, pp. 853-860, 2004.
[4] M. Tarini, E. Puppo, D. Panozzo, N. Pietroni, and P. Cignoni, "Simple Quad Domains for Field Aligned Mesh Parametrization," *ACM Trans. Graphics*, vol. 30, no. 6, pp. 142:1-142:12, Dec. 2011.
[5] D. Bommes, B. Lvy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, and D. Zorin, "State of the Art in Quad Meshing," *Eurographics STARS*, 2012.
[6] X. Li, X. Guo, H. Wang, Y. He, X. Gu, and H. Qin, "Meshless Harmonic Volumetric Mapping Using Fundamental Solution Methods," *IEEE Trans. Automation Science and Eng.*, vol. 6, no. 3, pp. 409-422, July 2009.
[7] K. Wang, X. Li, B. 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.
[8] 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.
[9] H. Wang, Y. He, X. Li, X. Gu, and H. Qin, "Polycube Splines," *Proc. ACM Symp. Solid and Physical Modeling (SPM '07)*, pp. 241-251, 2007.
[10] H. Wang, M. Jin, Y. He, X. Gu, and H. Qin, "User-Controllable Polycube Map for Manifold Spline Construction," *Proc. ACM Symp. Solid and Physical Modeling (SPM)*, pp. 397-404, 2008.
[11] 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.
[12] Y. He, H. Wang, C.-W. 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.
[13] Z. Petroc, K. Lanning, and T. Baysal, *Character Modeling 2*. Ballistic Publishing, 2007.
[14] C. Loop, S. Schaefer, T. Ni, and I. Castaño, "Approximating Subdivision Surfaces with Gregory Patches for Hardware Tessellation," *ACM Trans. Graphics*, vol. 28, no. 5, pp. 1-9, 2009.
[15] N. Tatarchuk, "Advanced Topics in GPU Tessellation," *Proc. Gamefest '08*, 2008.
[16] Pixologic, "Zbrush," http://www.pixologic.com/, 2010.
[17] S. Wan, Z. Yin, K. Zhang, H. Zhang, and X. Li, "A Topology-Preserving Optimization Algorithm for Polycube Mapping," *Computer & Graphics*, vol. 35, no. 3, pp. 639-649, 2011.
[18] K. Wang, X. Li, B. Li, H. Xu, and H. Qin, "Restricted Trivariate Polycube Splines for Volumetric Data Modeling," *IEEE Trans. Visualization and Computer Graphics*, to be Published 2011.
[19] 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.
[20] S. Dong, P.-T. Bremer, M. Garland, V. Pascucci, and J.C. Hart, "Spectral Surface Quadrangulation," *ACM Trans. Graphics*, vol. 25, no. 3, pp. 1057-1066, 2006.
[21] Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun, "Designing Quadrangulations with Discrete Harmonic Forms," *Proc. Fourth Eurographics Symp. Geometry Processing (SGP '06)*, pp. 201-210, 2006.
[22] F. Kälberer, M. Nieser, and K. Polthier, "Quadcover - Surface Parameterization Using Branched Coverings," *Computer Graphics Forum*, vol. 26, no. 3, pp. 375-384, 2007.
[23] J. Huang, M. Zhang, J. Ma, X. Liu, L. Kobbelt, and H. Bao, "Spectral Quadrangulation with Orientation and Alignment Control," *ACM Trans. Graphics*, vol. 27, no. 5, p. 147, 2008.

[24] N. Ray, B. Vallet, L. Alonso, and B. Lévy, "Geometry Aware Direction Field Processing," *ACM Trans. Graphics,* vol. 29, no. 1, article 1, 2009.
[25] D. Bommes, H. Zimmer, and L. Kobbelt, "Mixed-Integer Quadrangulation," *ACM Trans. Graphics,* vol. 28, no. 3, pp. 1-10, 2009.
[26] A. Myles, N. Pietroni, D. Kovacs, and D. Zorin, "Feature-Aligned T-Meshes," *ACM Trans. Graphics,* vol. 29, no. 4, pp. 117:1-117:11, July 2010.
[27] N. Ray, W.-C. Li, B. Lévy, A. Sheffer, and P. Alliez, "Periodic Global Parameterization," *ACM Trans. Graphics,* vol. 25, no. 4, pp. 1460-1485, 2006.
[28] A.W.F. Lee, D. Dobkin, W. Sweldens, and P. Schröder, "Multiresolution Mesh Morphing," *Proc. ACM SIGGRAPH '99,* pp. 343-350, 1999.
[29] T. Michikawa, T. Kanai, M. Fujita, and H. Chiyokura, "Multiresolution Interpolation Meshes," *Proc. Ninth Pacific Conf. Computer Graphics and Applications (PG '01),* pp. 60-69, 2001.
[30] E. Praun, W. Sweldens, and P. Schröder, "Consistent Mesh Parameterizations," *Proc. ACM SIGGRAPH '01,* pp. 179-184, 2001.
[31] V. Kreavoy and A. Sheffer, "Cross-Parameterization and Compatible Remeshing of 3D Models," *ACM Trans. Graphics,* vol. 23, pp. 861-869, 2004.
[32] I.-C. Yeh, C.-H. Lin, O. Sorkine, and T.-Y. Lee, "Template-Based 3D Model Fitting Using Dual-Domain Relaxation," *IEEE Trans. Visualization and Computer Graphics,* vol. 17, no. 8, pp. 1178-1190, Aug. 2011.
[33] N. Litke, A. Levin, and P. Schröder, "Fitting Subdivision Surfaces," *Proc. Conf. Visualization (VIS '01),* pp. 319-324, 2001.
[34] K.-S. Cheng, W. Wang, H. Qin, K.-Y. Wong, H. Yang, and Y. Liu, "Design and Analysis of Optimization Methods for Subdivision Surface Fitting," *IEEE Trans. Visualization and Computer Graphics,* vol. 13, no. 5, pp. 878-890, Sept./Oct. 2007.
[35] D. Panozzo, E. Puppo, M. Tarini, N. Pietroni, and P. Cignoni, "Automatic Construction of Adaptive Quad-Based Subdivision Surfaces Using Fitmaps," *IEEE Trans. Visualization and Computer Graphics,* vol. 17, no. 10, pp. 1510-1520, Oct. 2011.
[36] M. Bunnell, "Adaptive Tessellation of Subdivision Surfaces with Displacement Mapping," *GPU Gems 2,* Addison Wesley, 2005.
[37] C. Loop and S. Schaefer, "Approximating Catmull-Clark Subdivision Surfaces with Bicubic Patches," *ACM Trans. Graphics,* vol. 27, no. 1, pp. 1-11, 2008.
[38] T.K. Dey, K. Li, and J. Sun, "On Computing Handle and Tunnel Loops," *Proc. IEEE Int'l Conf. Cyberworlds NASAGEM Workshop (NASAGEM '07),* pp. 357-366, 2007.
[39] T.K. Dey, K. Li, J. Sun, and D. Cohen-Steiner, "Computing Geometry-Aware Handle and Tunnel Loops in 3d Models," *ACM Trans. Graphics,* vol. 27, pp. 45:1-45:9, Aug. 2008.
[40] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, "Multiresolution Analysis of Arbitrary Meshes," *Proc. ACM SIGGRAPH '95,* pp. 173-182, 1995.
[41] D.A. Field, "Laplacian Smoothing and Delaunay Triangulations," *Comm. Applied Numerical Methods,* vol. 4, pp. 709-712, 1988.
[42] P. Degener, J. Meseth, and R. Klein, "An Adaptable Surface Parameterization Method," *Proc. 12th Int'l Meshing Roundtable (IMR '03),* pp. 201-213, 2003.
[43] A. Khodakovsky, N. Litke, and P. Schröder, "Globally Smooth Parameterizations with Low Distortion," *ACM Trans. Graphics,* vol. 22, pp. 350-357, 2003.
[44] N. Pietroni, M. Tarini, and P. Cignoni, "Almost Isometric Mesh Parameterization through Abstract Domains," *IEEE Trans. Visualization and Computer Graphics,* vol. 16, no. 4, pp. 621-635, July/Aug. 2010.
[45] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin, "Modeling by Example," *ACM Trans. Graphics,* vol. 23, no. 3, pp. 652-663, 2004.
[46] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum, "Mesh eDiting with Poisson-Based Gradient Field Manipulation," *Proc. ACM SIGGRAPH '04,* pp. 644-651, 2004.
[47] X. Huang, H. Fu, O.K.-C. Au, and C.-L. Tai, "Optimal Boundaries for Poisson Mesh Merging," *Proc. ACM Symp. Solid and Physical Modeling (SPM '07),* 2007.
[48] A. Sharf, M. Blumenkrants, A. Shamir, and D. Cohen-Or, "SnapPaste: An Interactive Technique for Easy Mesh Composition," *Visual Computer,* vol. 22, pp. 835-844, Sept. 2006.
[49] V. Kreavoy, D. Julius, and A. Sheffer, "Model Composition from Interchangeable Components," *Proc. 15th Pacific Conf. Computer Graphics and Applications (PG '07),* pp. 129-138, 2007.
[50] L. Leblanc, J. Houle, and P. Poulin, "Modeling with Blocks," *Visual Computer,* vol. 27, no. 6-8, pp. 555-563, June 2011.
[51] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum, "Mesh Editing with Poisson-Based Gradient Field Manipulation," *ACM Trans. Graphics,* vol. 23, pp. 644-651, 2004.
[52] J. McDonald Jr., and B. Burley, "Per-Face Texture Mapping for Real-Time Rendering," *Proc. ACM SIGGRAPH Studio Talks (SIGGRAPH '11,)* pp. 3:1-3:1, 2011.
[53] P. Hanrahan and P. Haeberli, "Direct WYSIWYG Painting and Texturing on 3D Shapes," *ACM SIGGRAPH Computer Graphics ,* vol. 24, no. 4, pp. 215-223, 1990.
[54] S. Lanquetin and M. Neveu, "Reverse Catmull-Clark Subdivision," *Proc. Int'l Conf' Central Europe Computer Graphics, Visualization and Computer Vision (WSCG '06),* 2006.
[55] I. Castaño, "Tessellation of Subdivision Surfaces in Direct3D 11," *Proc. Gamefest '08,* 2008.
[56] I. Castaño, "Next-Generation Rendering of Subdivision Surfaces," *Proc. ACM SIGGRAPH '08,* 2008.
[57] I. Castaño, "Ownership-Based Zippering," http://castano.ludicon.com/blog/2009/01/10/ownership-based-zippering/, 2009.
[58] H. Wang, Y. He, X. Li, X. Gu, and H. Qin, "Polycube Splines," *Computer Aided Design,* vol. 40, no. 6, pp. 721-733, 2008.
[59] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas, "Robust on-Line Computation of Reeb Graphs: Simplicity and Speed," *ACM Trans. Graphics,* vol. 26, article 58, July 2007.
[60] O. van Kaick, H. Zhang, G. Hamarneh, and D. Cohen-Or, "A Survey on Shape Correspondence," *Proc. Eurographics State-of-the-Art Report,* pp. 1-24, 2010.

Ismael Garcia is working toward the PhD degree at the Geometry and Graphics Group, Universitat of Girona. His advisor is Dr. Gustavo Patow. His research is concerned with parallel efficient data structures for visualization and processing of spatial data, such as triangle meshes and implicit surfaces.



Jiazhi Xia received the BS and MS degrees in computer science from Zhejiang University, China, and the PhD degree in computer science from Nanyang Technological University. He is currently a lecturer at the School of Information Science and Engineering, Central South University, China. His research interests include computer graphics, multimedia, and human-computer interaction.

Ying He received the BS and MS degrees in electrical engineering from Tsinghua University, and the PhD degree in computer science from the Stony Brook University. He is currently an associate professor at the School of Computer Engineering, Nanyang Technological University. His research interests fall in the broad area of visual computing. He is particularly interested in the problems that require geometric computation and analysis. He is a member of the IEEE.

Shi-Qing Xin received the PhD degree in Zhejiang University in 2009. After that, he has been working as a research fellow at Nanyang Technological University in Singapore. In the past several years, he focused on discrete geodesic and its applications. His research interests include computer graphics, computational geometry and mobile computing.

Gustavo Patow received the PhD degree in computer science from Universitat Politècnica de Catalunya. He is an associate professor at Universitat de Girona's Computer Science and Applied Math Department. His research interests include modeling, texturing, GPU-based rendering, and procedural modeling.