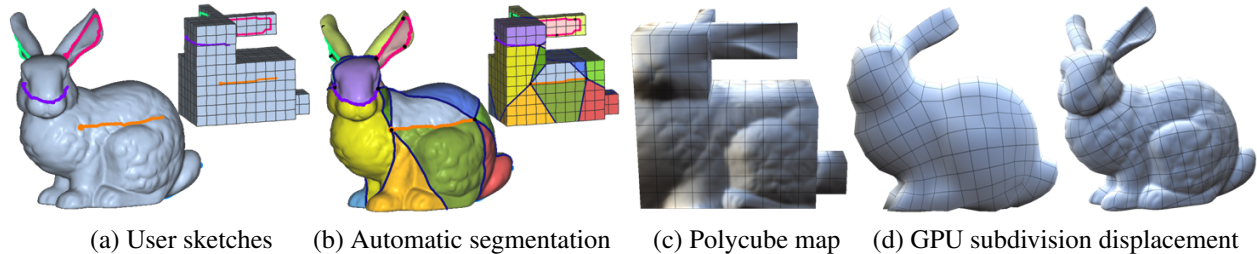


# Editable Polycube Map for GPU-based Subdivision Surfaces

Jiazhi Xia<sup>†\*</sup> Ismael Garcia<sup>‡\*</sup> Ying He<sup>†</sup> Shi-Qing Xin<sup>†</sup> Gustavo Patow<sup>‡</sup>  
<sup>†</sup>School of Computer Engineering, Nanyang Technological University, Singapore.  
<sup>‡</sup>Geometry and Graphics Group, University of Girona, Spain.



**Figure 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 polycube map is conformal and with low area distortion, thus facilitates some graphics applications such as GPU-based displacement mapping.

## Abstract

In this paper we propose an editable polycube mapping method that, given an arbitrary high-resolution polygonal mesh and a simple polycube representation plus optional sketched features indicating relevant correspondences between the two, provides a uniform, regular and artist-controllable quads-only mesh with a parameterized subdivision scheme. The method introduces a global parameterization, based on a divide and conquer strategy, which allows to create polycube-maps with a much smaller number of patches, and gives much more control over the quality of the induced subdivision surface. All this makes it practical for real-time rendering on modern hardware (e.g. *OGL 4.1* and *D3D11* tessellation hardware). By sketching these correspondence features, processing large-scale models with complex geometry and topology is now feasible. This is crucial for obtaining watertight displaced Catmull-Clark subdivision surfaces and high-quality texturing on real-time applications.

**Keywords:** Digital geometry processing, surface parameterization, polycube map, GPU subdivision surface.

## 1 Introduction

The motivations for this work come from two different, but related origins. On one side, it is a well known fact that most artists prefer modeling with quads, as quad geometry provides a better flow, tessellates cleaner and deformations under animation that are noticeably smoother, especially around joints [Oliverio 2006]. On the other side, hardware-supported tessellation is already feasible and provides much faster model visualization than traditional methods [Loop et al. 2009], and graphics card hardware designers have added specialized programmable units for the task (Shader Model 5 hardware) [Tatarchuk 2008].

\*J. Xia and I. Garcia contributed equally to this project.

(c) ACM, 2009. This is the author’s version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version will be published in I3D’2011.

Tarini et al. [2004] pioneered the concept of polycube maps as a technique to parameterize 3D shapes to the polycube domain, which is a natural generalization of the cube space and can be useful for the parametric domain of shapes with complicated topology and geometry. Compared to other global surface parameterization techniques, a polycube map has two unique features that make them promising for graphics applications. First, the parametric domain has a regular structure that naturally supports quadrangulation. The parametric domain can be easily constructed and visualized. Second, the singularities (polycube corners) have fixed structures, i.e., of valence 3, 5 or 6. The reduced number of possible singularities results in a small number of topological combinations.

Constructing a polycube map is a challenging work. From the users’ point of view, an ideal polycube mapping algorithm should have at least the following features: *Quality*: the map is a bijection with low angle and area distortion. *User control*: the user can easily control the mapping by specifying optional features on the 3D model and their desired locations on the polycube domain. *Performance*: the algorithm is efficient, robust, and automatic except for the editable user-specified constraints to control the map.

There are several approaches to construct a polycube mapping [Tarini et al. 2004; Wang et al. 2007; Wang et al. 2008; Lin et al. 2008; He et al. 2009]. Unfortunately, none of them has all the desired features. For instance, [Tarini et al. 2004] does not guarantee a bijection, [Wang et al. 2007; Lin et al. 2008; He et al. 2009] do not allow user control. [Wang et al. 2008] requires a large amount of user interaction to specify the polycube structure on the 3D model and is not suitable for large-scale models with complicated geometry and topology. More importantly, none of them allows the users to *edit* the polycube map in an easy and intuitive fashion: In general, current tools for editing in planar parametric domains can be awkward to use. On the other hand, editing on a polycube, which more closely resembles the gross structure of the model, could indeed be simpler, especially where the user is allowed to control the rough shape of the mapping.

In this paper, we present the editable polycube map to overcome the limitations of the existing approaches. See Figure 1. Our method allows the users to construct the polycube map in an intuitive and easy manner: given a 3D model  $M$  and its polycube domain  $P$ , the user is able to sketch features on  $M$  and  $P$  to specify feature correspondences. Then, our system will automatically compute the

map in such a way that the features on  $M$  are mapped to the user-specified locations on  $P$ . Later on, the user is allowed to edit the features on  $M$ ,  $P$  or both, providing fine-grained control over the mapping. This way, the editable features can help to *mark* and preserve sharp features on the polycube-mapped subdivision scheme.

We demonstrate the proposed editable polycube map framework to GPU-friendly interpolative subdivision surfaces. The positive properties enumerated above allow an extremely 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 whole process should be compared with the traditional work artists do to create a model to be used in an environment like a computer game, which usually requires a subdivision scheme to be used with modern tessellation hardware. In general, if the new model is based on an existing model, for instance obtained from a laser scanning process, the final model generation implies manually performing re-topology operations on a subdivision-based representation, and then transfer corresponding details by normal projection, which do not ensure bijectivity. Usually, artists start from a coarse base model, quite often sculpted using a polycube as base mesh. Then, the model is imported into an application like ZBrush [2010] and further subdivided with a couple of steps of Catmull-Clark subdivision. From this moment on, the artist has to model/transfer the fine-grained details onto the final model. Obviously, this is a time-consuming and quite redundant procedure. Our proposal is to avoid this by quickly establishing a bijective correspondence between the coarse polycube and the input model, and then creating the subdivision scheme on the polycube-based model by transferring the details of the high-resolution input model. The resulting model will have all the enumerated properties and would be ready for usage in a production environment with a minimal user interaction.

The specific contributions of this paper include:

- We present a method that, from a general mesh, creates a high-quality and artist-controllable polycube map in an efficient and intuitive manner. Our method allows the users to easily modify the map and fine-tune the mapping. The user is also able to control the number of patches in the base mesh of the subdivision scheme by the construction of the base polycube. The provided polycube doesn't need to accurately resemble the shape of the object. In fact, coarse polycubes are enough.
- We provide a subdivision scheme specially built for quad patch-based tessellation at the GPU for object and character rendering. Also, this scheme provides a reduced number of topology combinations thanks to the low number of valence possibilities (only 3, 5 or 6), which is very important in terms both of memory footprint and of the texture fetching bandwidth, which strongly affects performance.

The remaining of the paper is organized as follows: Section 2 reviews related works on polycube map, surface quadrangulation and hardware based tessellation. Then, Section 3 presents the details of our editable polycube map framework. After that, Section 4 shows the experimental results and discussions. Finally, Section 5 draws the conclusion and discusses our future work.

## 2 Related work

Tarini *et al.* [2004] pioneered the concept of polycube maps. They roughly approximated the input 3D model by a polycube, and then constructed its dual space. By carefully examining the different

configurations that can occur in a non-empty cell, they designed six projection functions that map the points inside a cell of the dual space to the polycube surface. These projection functions are elegantly designed to guarantee a globally continuous map. However, their method does not produce a bijective mapping since two vertices on the same projection line share the same image, which is a fundamental feature for a large range of applications. Furthermore, their method has strict requirements on the shape of the polycube, like that the dual space should completely enclose a slightly modified version of the input model in an intermediate coordinate space. Rather than projecting the 3D surface to the polycube, Wang *et al.* [2007] introduced an intrinsic approach that first maps the 3D model and the polycube to the canonical domain (e.g., sphere, euclidean plane or hyperbolic disc), and then seeks the map between the two canonical domains. The resulting polycube map is guaranteed to be a diffeomorphism. However, in this scheme it is difficult to control the polycube map, i.e., a feature on the 3D model may not be mapped to a desired location on the polycube. In their follow-up work, Wang *et al.* [2008] proposed the user-controllable polycube map where the users can specify the pre-images of the polycube corners. Their method works well for shapes with simple geometry and topology, but is not feasible for complicated models since it is very tedious and error-prone to specify the polycube structure manually. Instead, the method presented here provides much more user control, which allows us to create polycube maps with a much smaller number of patches, and gives much more control over the quality of the induced subdivision surface, which is what makes this method practical for real-time rendering on modern hardware.

It is known that the polycube map quality (in terms of angle and area distortion) highly depends on the shape of the polycube. There have been some research efforts that aim to construct the polycube automatically [Lin *et al.* 2008; He *et al.* 2009]. These techniques usually break down the input model into smaller and simpler components and then use polycube primitives to approximate each one. For example, the Reeb graphs [Lin *et al.* 2008] and harmonic functions [He *et al.* 2009] 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 models of complicated geometry and topology. As we mentioned before, none of the existing algorithms allows the users to easily edit the maps.

Our work is also related to quadrangulation which has been studied extensively in the past few years. There are a number of excellent algorithms for the automatic or semi-automatic quadrangulation of arbitrary simplicial 2-manifolds, such as [Dong *et al.* 2006], [Tong *et al.* 2006], [Kälberer *et al.* 2007], [Huang *et al.* 2008], [Ray *et al.* 2009], [Bommes *et al.* 2009]. These constructed quads can nicely align the principal directions. Litke *et al.* [2001] developed a technique to fit Catmull-Clark subdivision surfaces to a given shape within a prescribed tolerance, based on the method of quasi-interpolation. However, the subdivision scheme must be known beforehand, and it does not represent a continuous parameterization, as our method does.

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 [Bunnell 2005] [Tatarchuk 2008]. Loop *et al.* [2009] and Castaño [2008b] presented a method for approximating subdivision surfaces with hardware-accelerated parametric patches, but their method presents some disparity between the surface mesh they created and the one that is rendered in realtime, while our method presents a uniform, regular and artist-controllable quads-only mesh with a parameterized subdivision scheme, which fits nicely into their pipeline because of the low number of combinations it generates, but also provides a number of already-mentioned benefits for the artist.

Another related topic is cross parameterization. Recently, many algorithms are developed for building the mappings between general surfaces of same topology. A common approach is to parameterize the models over a common base mesh [Lee et al. 1999; Michikawa et al. 2001; Praun et al. 2001; Kraevoy and Sheffer 2004]. In this approach, the meshes are split into matching patches, each set of which is then parameterized on a common planar domain. A given set of matching feature points serve as the corners of patches and the feature correspondences. In particular, Yeh *et al.* [2010] proposed an interactive interface for correspondence placement. However, all the above approaches use points as the feature correspondence, while our method supports user sketches. In practice, we find it more intuitive to draw feature lines than to place points.

### 3 Editable Polycube Map

#### 3.1 Overview

As mentioned, the objective of the proposed technique is, starting from a high-resolution model coming either from an artist or a 3D scanner (plus its cleaning stage), to build a new subdivision scheme that allows user control over a reduced number of singularities, and have that model ready for seamless texturing, watertight displacement, etc. This should be done in an artist-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 by the users<sup>1</sup>. By controlling the position and location of the cubes, and by providing additional controlling sketches, the user has control at all times over the number and location of the singular vertices in the resulting quads-only mesh. Then the user-specified features are sampled with points, from which we compute the shortest distance for each other sampled point using multi-source Dijkstra’s shortest path algorithm. Although the computed distance field induces a triangulation on the 3D model, the path between two points is not straight and the resulting patch can have a complex non-triangular shape. 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 3D model and the polycube. This way, the resulting triangulation is smoother and more visually pleasing than the one that could be obtained directly from the multi-source Dijkstra computation. 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. By setting the boundary conditions carefully, the computed map is guaranteed to be continuous along the cutting boundaries. Finally, a simple and effective global diffusion algorithm is applied to improve the map quality. The resultant map is bijective and satisfies the user-specified constraints.

The polycube map induces a quad-remeshed version of the high-res model. An immediate benefit is that the patches of the resulting remeshed model come from tessellated squared faces, so a very low-resolution version of the model can be built from the original polycube faces. This low-resolution model not only has quads as its only primitive, but also has only vertices with a small and restricted valence number, only 3, 4, 5 and 6. This low-resolution model is the basis of our subdivision scheme, see Section 3.4.

Then, polycube texture mapping is performed, working only with the polycube version of the model, and consists of creating a 2D

<sup>1</sup>Many commercial softwares, such as Maya and 3ds Max, allow the users to do this easily.

texture atlas which contains all externally visible polycube faces, which is an easy task as the previous step already kept only the visible faces that are not shared by more than one cube. To build the atlas, we just placed each face in consecutive squares in the final texture atlas.

Finally, in runtime, only the very low-resolution model needs to be sent to the GPU, along with the texture maps built in the pre-processing, to generate a continuous subdivision scheme with all the benefits mentioned in Section 1. As the very low-resolution model can be animated, its run-time tessellated version also can. It is important to mention that the user/artist has full control over the process through the segmented input polycubes, the sketched features and the segmentation step, and can introduce further tweaking in the processed mesh without the need to re-parameterize the model. The entire pipeline is illustrated in Figure 2.

#### 3.2 Constructing Polycube Map

The input model  $M$  for the method presented in this paper comes from two kinds of possible input models: a model from a 3D scanner, after a cleaning pass, or a high-resolution model created directly by an artist.

Along with the high-resolution model, the artist/modeler should also provide a polycube surface  $P$  that follows the shape of the original model. This polycube model can be coarse or fine, both being able to create a good dual parameterization of the original mesh, but depending on how fine  $P$  model is, it would be much easier to capture high frequency details in the final tessellated model.

##### 3.2.1 Segmentation

The divide-and-conquer approach [He et al. 2009] constructs the polycube map by breaking down the model into genus-0 patches and then computing the piecewise map independently. Although their method is able to divide the model in automatically, all cutting planes are horizontal. Thus, the segmentation highly depends on the orientation of the model and may result in too many small patches, and the cutting boundary may not represent any feature.

In our framework, the users are allowed to sketch the features freely on the models. Here we assume that the user-specified features are consistent. For example, as shown in Figure 2(b), the user sketches a few features on the 3D model  $M$ , and the same number of features must be specified on the polycube  $P$ . Furthermore, the spatial relation of the features should be consistent in the sense that they are aligned in a similar order, otherwise, the induced harmonic maps may not be matched.

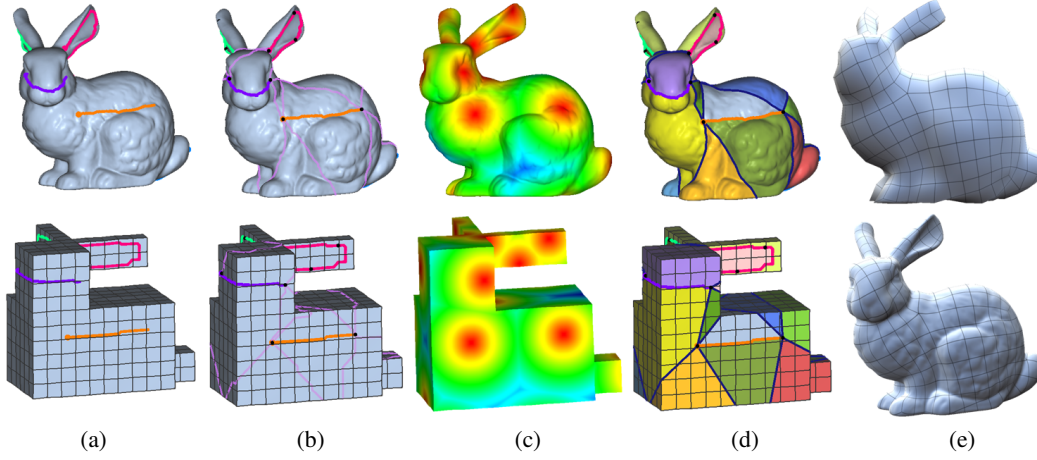
Our segmentation algorithm is as follows:

**Step 1.** Given the user-specified sketches,  $\gamma_i \in M$ ,  $\gamma'_i \in P$ ,  $i = 1, \dots, n$ , we sample the sketches with 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.

**Step 2.** Use  $p_j \in M$ ,  $j = 1, \dots, m$ , as source points and compute the shortest distance for every point on  $M$  using multi-source Dijkstra’s algorithm. So 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.** 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. 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, find the shortest path between  $s_1$  and  $s_2$ . It can be

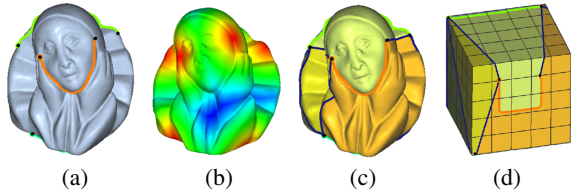


**Figure 2:** Algorithmic pipeline. (a) The user sketches constraints on the 3D model  $M$  and polycube  $P$ . (b) Black dots are sample points on these constraints. (c) We compute the distance fields using the samples as sources. (d) The distance fields induce a triangulation on  $M$ . Using the correspondence of the sketches, we compute the geodesic triangulation on  $P$ , so the model can be cut into genus-0 patches. We compute the constrained map between each pair of patches such that constraints are mapped consistently. With a carefully designed boundary condition, the two maps can be glued seamlessly. (e) The iso-parametric line on  $M$ : On top the base level of the subdivision surface.

shown that two shortest paths can only meet at the regions of the two ending sample points. Then on the polycube  $P$ , compute the geodesic between  $s_i$  and  $s'_j$ .

**Step 5.** Segment  $M$  and  $P$  along the computed shortest path and geodesic path.

In the above algorithm, we compute a distance field on the 3D model  $M$  using the user sketched constraints. As shown in Figure 3, this distance field naturally induces a triangulation on  $M$ .



**Figure 3:** Distance field based on multi-source Dijkstra's algorithm and its induced triangulation. Black dots in (a) are sample points on the user sketches (color curves). Using these samples as sources, we compute the Dijkstra's based distance field on the model, as shown in (b) (warm colors mean small distance to the sources, cold colors are large distances). Curves in (c) are the shortest paths between sample points, which induces a triangulation on  $M$ . By using the correspondence of the user sketches, in (d) we construct the geodesic triangulation on the polycube  $P$ .

### 3.2.2 Constrained map

Let  $P_i \in P$  and  $M_i \in M$  be the pair of segmented patches, each of which is a genus-0 surface with only one boundary. We want to find a bijective and smooth map  $\phi : M_i \rightarrow P_i$ . Rather than computing the map directly, we first parameterize  $M_i$  to the unit disc using harmonic map, i.e.,  $f : M_i \rightarrow \mathbb{D}$  such that  $\Delta f = 0$  and  $f$  maps the boundary of  $M_i$  to the boundary of  $\mathbb{D}$  using arc length parameterization,  $f(\partial M_i) = \partial \mathbb{D}$ . Similarly we also parameterize  $P_i$  to the unit disc using harmonic map  $g : P_i \rightarrow \mathbb{D}$ . More details of discrete harmonic map could be found at [Eck et al. 1995].

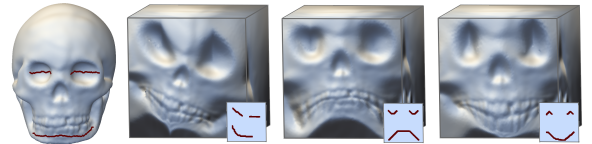
Then we seek a smooth map between two unit discs  $h : \mathbb{D} \rightarrow \mathbb{D}$ . This map  $h$  is also computed using harmonic map  $\Delta h = 0$  and

the boundary condition is set as follows: Let  $s_1, s_2$  and  $s_3$  be the sample points on  $\partial M_i$ , and  $f(s_j) \in \partial \mathbb{D}$ ,  $j = 0, 1, 2$  be the images on the boundary of unit disc. Similarly, let  $g(s'_j) \in \partial \mathbb{D}$  be the images of the sample points  $s'_j \in P_i$ . Then we require the function  $h$  maps  $f(s_j)$  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 arc length parameterization.

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

$$\begin{array}{ccc}
 M_i & \xrightarrow{\phi} & P_i \\
 f \downarrow & & \downarrow g \\
 \mathbb{D} & \xrightarrow{h} & \mathbb{D}
 \end{array}$$

Figure 4 demonstrates the results of constrained map. User can take full control of the polycube map using simple sketches as the boundary condition.



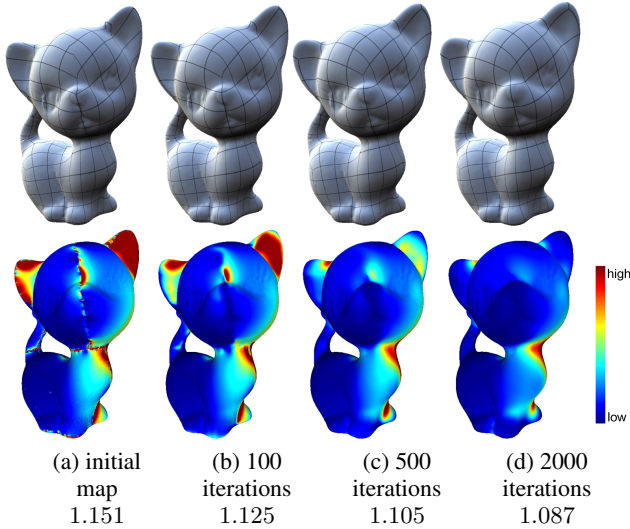
**Figure 4:** The users can take full control of the polycube map by simple sketches. The thumbnails show the sketched constraints.

### 3.2.3 Globally smoothing map

We glue the piecewise maps  $\phi_i : P_i \rightarrow M_i$  together. With the above boundary conditions, the maps are consistent along the boundaries which can be glued seamlessly. The resulting map  $\cup_i \phi_i$  guarantees to be  $C^0$  continuous along the segmentation boundaries.

We use Laplacian smoothing [Field 1988] 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  $p \in P$ . Then





**Figure 5:** 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 unsmoothness 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  $\delta = 0.05$ .

we solve the following diffusion function:

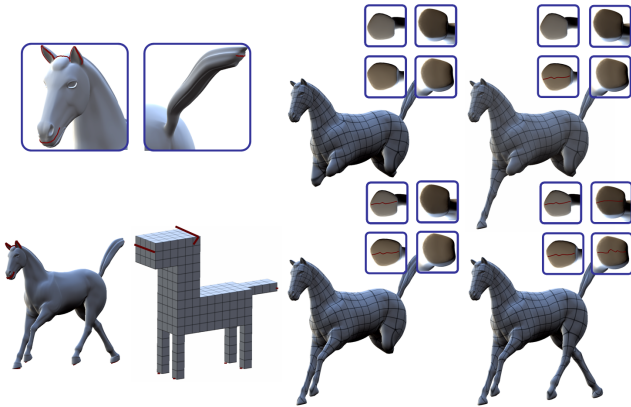
$$\frac{\partial p'(t)}{\partial t} = -(\Delta p'(t))_{\parallel}, \quad (1)$$

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

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

$$\Delta p' = p' - \frac{1}{m} \sum_{pq \text{ is edge}} q', \quad (2)$$

where  $m$  is the valence of  $p$ .



**Figure 6:** Interactive editing of the polycube map. Initially, the user only draws three features on the mouth, ears and tail. As a result, the horse head, body and tail are parameterized well, but the four legs are poorly sampled. Then the user can improve the map quality by specifying additional features on the feet. Finally, the whole model is parameterized well.

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

Following [Degener et al. 2003; Tarini et al. 2004], 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  $\sigma_1$  and  $\sigma_2$  are the singular values of the Jacobian matrix of  $\phi$ .  $\epsilon_{angle} = \epsilon_{area} = 1$  when the map  $\phi$  is isometric. As shown in Figure 5, our method leads to visually pleasing results in only a few hundred iterations. Khodakovsky *et al.* [2003] introduced a globally smooth parameterization method. Particularly, our globally smooth algorithm is designed for polycube mapping, while their method is for mapping between triangle shaped patches.

### 3.3 Texture Mapping and run-time data generation

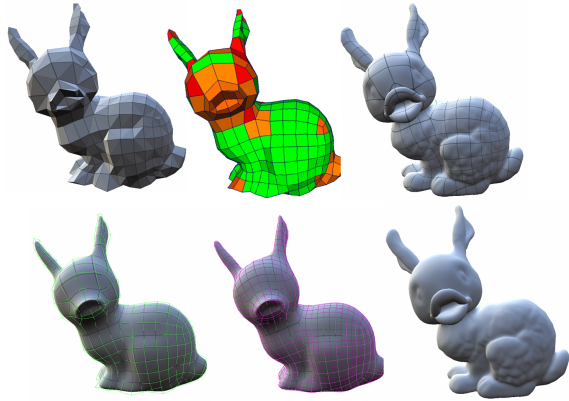
For any further processing, we should provide a map from the polycube space to regular texture space, a process that only needs the polycube texture coordinates of the model. With them, a 2D texture atlas is created containing the textures corresponding to all *visible* polycube faces, in a similar way to the proposal by Cohen et al. [1998]. As already mentioned, this is an easy task as the previous steps already discarded internal faces shared by more than one cube. Once the visible faces are found, they are placed in consecutive squares in the final texture atlas. Each vertex in the model is assigned its respective texture coordinates in the atlas, thus smoothly integrating texturing to the polycube mapping process.

Once the mapping stage has finished, we have a bijective mapping between the tiles in texture space (one for every patch) and the artist-provided high-resolution 3D geometry. Probably, the artist also provided normal maps, color maps, specular maps, etc. We have developed a completely automatic scheme that transfers (bakes) that information to texture tiles, and encodes them along with the subdivision information. Traditionally, applications like ZBrush [2010] perform an explicit projection over different resolution versions of the same model, but this can have serious problems with an arbitrary simplification scheme. In our case, as we have defined a *dual parameterization*, this is done in an automatic way without losing any detail in the process. To do it, we rasterize each high-resolution patch into its associated texture space, saving the required information (normals, occlusion maps, etc). Special care should be taken when doing this process with the displacement maps, either if they are scalar displacement along the surface normal or full 3D displacements, as we also need to have access to neighboring patches to correctly generate this information. The whole process can easily be done with tools like XNormal [2010].

### 3.4 GPU-based subdivision displacement

The previous sections explained how to build the bijective mapping between the model and the polycube-map. Now, we need to explain how the induced subdivision surface is built. In general, we can generate a parameterized subdivision scheme which can be effectively used in any graphics application like computer-generated movies, as well as real-time applications. In our implementation, as the bijection has already been established, we only need to recover the coarse quads from the polycube-mapped model. In order to do that we (re-)constructed the Catmull-Clark Subdivision scheme with the algorithm described in [Lanquetin and Neveu 2006], which allows to obtain a valid Catmull-Clark subdivision scheme for filming rendering and prepare the approximate Catmull-Clark [Loop and Schaefer 2008] data structures (textures & base mesh) for the real-time subdivision.

Later on, in real time applications, we use the subdivision algorithm presented by Loop and Schaefer [2008], following the implemen-



**Figure 7:** GPU-based subdivision displacement. Top left: our base mesh (level 0 of the subdivision scheme). Top middle: patch structure we associate with the subdivision surface (green patches contain only valence 4 vertices, and the darkness of the orange levels shows the number of extraordinary vertices, darker patches having the greater number). Bottom left: the base mesh superimposed to the approximate Catmull-Clark subdivision surface in grey. Bottom middle: In lilac, the subdivision surface. Right Column: the subdivision surface with GPU-based subdivision displacement (the iso-parametric curves can be seen in the top right).

**Table 1:** Comparison of polycube map construction methods. Symbols: ● good, ◐ fair, ○ poor.

| Features              | Tarini [2004] | Wang [2007] | Wang [2008] | Lin [2008] | He [2009] | Our method |
|-----------------------|---------------|-------------|-------------|------------|-----------|------------|
| Map quality           | ●             | ◐           | ◐           | ●          | ●         | ●          |
| Bijection             | ○             | ●           | ●           | ●          | ●         | ●          |
| User control          | ○             | ○           | ◐           | ○          | ○         | ●          |
| Editing               | ○             | ○           | ○           | ○          | ○         | ●          |
| Polycube construction | ○             | ○           | ○           | ●          | ●         | ○          |
| Automatic             | ●             | ○           | ○           | ●          | ●         | ●          |
| Large models          | ●             | ○           | ○           | ○          | ◐         | ●          |
| Arbitrary topology    | ○             | ○           | ○           | ○          | ◐         | ◐          |

tation described by Castaño [2008b]. The method presented here is particularly well suited for this implementation as we only generate vertices with valences 3,4,5 or 6. In this implementation, domain shaders (or vertex shaders when using instanced tessellation) are responsible for providing the final position of each new vertex from the tessellated mesh.

In order to have watertight sampling of the displacement map, we define, for each patch, the one who “owns” every single edge and corner [Castaño 2008b; Castaño 2008a]. This way, all patches are coordinated with respect to what texture coordinate to use when sampling the displacement map. In practice, this amounts storing, for every edge and for every corner, the texture coordinates of the owner of those features (4 texture coordinates per vertex). At runtime, only a single texture sample is needed; and the corresponding texture coordinate can be selected with a simple calculation.

Geometry image tessellation is also an interesting option that becomes subcase of our strategy, but it is a technique mainly intended for static and not too large objects. With geometry images, no texture coordinate-specific information is required, sufficing just the ownership data to compute everything in the respective shaders.

## 4 Results and Discussion

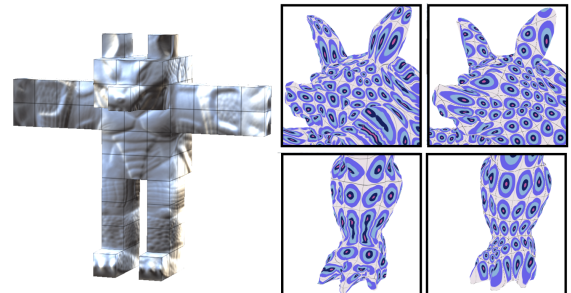
We tested our method with a wide-range of models with various geometry and topology configurations as shown in Fig. 10. Table 2

**Table 2:** Statistics of experimental results.  $\#\triangle$ : # of triangles in the input mesh;  $\#\square$ : # of squares in the polycube of base level;  $l$ : # of subdivision levels for the high-resolution polycube;  $n_c$ : # of corners in  $P$ ;  $n_f$ : # of user-specified features;  $T$ : time measured in seconds;  $\epsilon_1$ : angle distortion;  $\epsilon_2$ : area distortion.

|               | $\#\triangle$ | $\#\square$ | $l$ | $n_c$ | $n_f$ | $T$ | $\epsilon_1$ | $\epsilon_2$ |
|---------------|---------------|-------------|-----|-------|-------|-----|--------------|--------------|
| Armadillo     | 346K          | 1012        | 6   | 82    | 28    | 148 | 1.16         | 1.18         |
| Arthur        | 252K          | 57          | 7   | 12    | 5     | 76  | 1.02         | 1.13         |
| Bunny         | 144K          | 452         | 4   | 22    | 6     | 35  | 1.01         | 1.13         |
| Horse         | 67K           | 436         | 5   | 28    | 8     | 11  | 1.04         | 1.07         |
| Lucy          | 526K          | 980         | 5   | 38    | 15    | 210 | 1.12         | 1.23         |
| Skull         | 52K           | 24          | 7   | 8     | 3     | 7   | 1.02         | 1.06         |
| Tylo head     | 500K          | 920         | 5   | 32    | 12    | 194 | 1.10         | 1.25         |
| Isidore Horse | 151K          | 74          | 8   | 14    | 6     | 48  | 1.01         | 1.07         |

shows the statistics of our experiments. Figure 6 shows an example of interactive editing of the polycube map of the horse model.



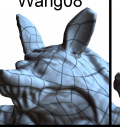

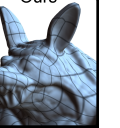
A good global bijective parameterization is very important for modeling and texturing, as typical projections as used in [Tarini et al. 2004] unavoidably have problems since two vertices on the same projection line share the same image, something that produces artifacts that are clearly visible in Figure 8.



**Figure 8:** Comparison of the bijectivity between the editable polycube map and the original polycube implementation, at the left the original [Tarini et al. 2004]. Top row: insets of the Armadillo head. Bottom row: insets of its feet. As we can see, non-bijection in [Tarini et al. 2004] results in a dependence of multiple points on the mesh with a single point in texture space: painting this single point stains other places than the one originally intended.

We also quantitatively compared our methods with existing methods like [Tarini et al. 2004], [Wang et al. 2007], [Wang et al. 2008] and [He et al. 2009] as shown in Table 1 and Figures 8 and 9. [Tarini et al. 2004] is efficient for large scale models, but it can not guarantee the bijectivity due to the projection of the 3D model to the polycube, and may result in some undesired artifacts in texture mapping and painting, see Figure 8. [Wang et al. 2007] 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). Thus, the induced map also has a large area distortion with uneven sampling. [Wang et al. 2008] required to manually specify the images of the polycube corners and edges on the 3D model and then computed 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. Furthermore, the user can not specify other features or constraints in [Wang et al. 2008]. Automatic ap-

proaches as [Lin et al. 2008; He et al. 2009] use heuristics and may not work well for complex models. In [He et al. 2009], 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, e.g., the arms are not axis aligned, there are large distortions on the upper arms and shoulder, see Figure 9. Compared to the existing approaches, our method is more intuitive and flexible in terms of user control and editing, and can generate better quality polycube-maps.

| Tarini04  | Wang07  | Wang08  | He09  | Ours  |
|---|---|---|---|---|
|  |  |  |  |  |
| 1.32  | 1.35  | 1.29  | 1.25  | 1.16  |
| 1.22  | 88.62   | 1.24  | 1.23  | 1.18  |

**Figure 9:** Comparisons of our method with [Tarini et al. 2004], [Wang et al. 2007], [Wang et al. 2008] and [He et al. 2009]. We use the same polycube to make the comparison fair. Our method is more intuitive and flexible in terms of user control and editing, and can generate polycube map of better quality. The values below each figure are the angle and area distortions.

**Discussions** We want to emphasize the differences between our method and [He et al. 2009]. They first segmented the 3D model and polycube by horizontal planes, computed 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, since the cutting loci are also the constraints of the map, [He et al. 2009] can only map the horizontal, planar features from the 3D 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 that are not provided in [He et al. 2009]. Second, [He et al. 2009] mapped the segmented components to the multiply 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 computed a harmonic map between two topological disks, so it is more efficient than [He et al. 2009]. 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 Figure 5, 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.

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 distortion of the map. However, the price to pay is the larger number of extraordinary points (polycube corners). Thus, there is a tradeoff between quality and complexity. Through our experiments, we observed that for shapes with extruding regions, e.g., the Armadillo’s fingers and toes, it is usually a good idea to design an accurate polycube to model these features. In our framework, we leave the choice to the users.

## 5 Conclusions

We have proposed a new method that aims to improve the existing ones and develop a very practical and efficient system. From a general mesh with optional featured sketches, we create an artist-controllable quads-only mesh with a globally smooth parameterization. The method guarantees that the computed map is bijective

and conformal except at a finite number of extraordinary points (the polycube corners). 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 preprocessing stage, the artist/user is able to control the number of patches in the base mesh of the subdivision scheme by the construction of the base polycube. Then, the artist still has the possibility of fully controlling the process by sketching correspondence lines between the high-resolution model and the polycube.

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 run-time performance. The result can be tessellated with modern hardware using instanced tessellation, or with the new programmable units in Shader Model 5 hardware.

**Limitations** The limitation of the proposed framework lies in the segmentation step. For models with genus more than one, the users are required to sketch a loop on each handle explicitly. Then our algorithm can cut along the user-specified loops and then automatically segment the model into genus-0 patches. We conducted experiments for models with non-trivial topology (such as Kitten of genus-1 and Happy Buddha of genus-6). In the future, we will improve our framework by computing the canonical homology basis automatically, which can be used as the constraints for segmentation.

## Acknowledgements

J. Xia, Y. He and S.-Q. Xin are partially supported by AcRF 69/07 and NRF2008IDM-IDM004-006. I. Garcia and G. Patow are partially supported by grant TIN2007-67120 from Ministerio de Educación y Ciencia, 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 3D Scanning Repository, AIM@Shape, K. Crane, H. Alvarado, and J. Johnson-Mortimer for the 3D models.

## References

- BOMMES, D., ZIMMER, H., AND KOBELT, L. 2009. Mixed-integer quadrangulation. *ACM Trans. Graph.* 28, 3.
- BUNNELL, M. 2005. *GPU Gems 2*. Addison Wesley, ch. Adaptive Tessellation of Subdivision Surfaces with Displacement Mapping.
- CASTAÑO, I., 2008. Next-generation rendering of subdivision surfaces. *SIGGRAPH 2008*.
- CASTAÑO, I., 2008. Tessellation of subdivision surfaces in direct3d 11. *Gamefest 2008*.
- COHEN, J., OLANO, M., AND MANOCHA, D. 1998. Appearance-preserving simplification. In *SIGGRAPH '98*, 115–122.
- DEGENER, P., MESETH, J., AND KLEIN, R. 2003. An adaptable surface parameterization method. In *IMR '03*, 201–213.
- DONG, S., BREMER, P.-T., GARLAND, M., PASCUCCI, V., AND HART, J. C. 2006. Spectral surface quadrangulation. *ACM Trans. Graph.* 25, 3, 1057–1066.
- ECK, M., DE ROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. 1995. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH '95*, 173–182.
- FIELD, D. A. 1988. Laplacian smoothing and delaunay triangulations. *Communications in Applied Numerical Methods* 4, 709–712.



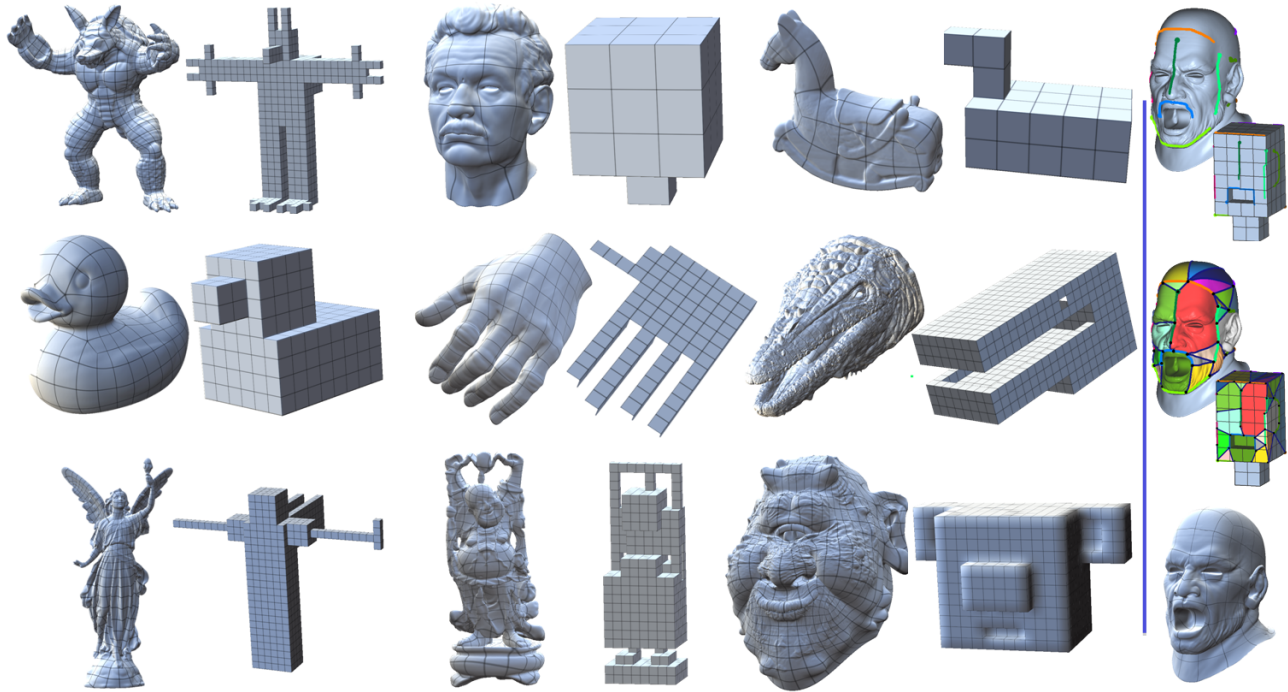


Figure 10: More polycube mapping results.

- HE, Y., WANG, H., FU, C.-W., AND QIN, H. 2009. A divide-and-conquer approach for automatic polycube map construction. *Computer & Graphics* 33, 3, 369–380.
- HUANG, J., ZHANG, M., MA, J., LIU, X., KOBELT, L., AND BAO, H. 2008. Spectral quadrangulation with orientation and alignment control. *ACM Trans. Graph.* 27, 5, 147.
- KÄLBERER, F., NIESER, M., AND POLTHIER, K. 2007. Quad-cover - surface parameterization using branched coverings. *Comput. Graph. Forum* 26, 3, 375–384.
- KHODAKOVSKY, A., LITKE, N., AND SCHRÖDER, P. 2003. Globally smooth parameterizations with low distortion. *ACM Trans. Graph.* 22, 350–357.
- KRAEVOY, V., AND SHEFFER, A. 2004. Cross-parameterization and compatible remeshing of 3d models. *ACM Trans. Graph.* 23, 861–869.
- LANQUETIN, S., AND NEVEU, M. 2006. Reverse catmull-clark subdivision. In *WSCG '06*.
- LEE, A. W. F., DOBKIN, D., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresolution mesh morphing. In *SIGGRAPH '99*, 343–350.
- LIN, J., JIN, X., FAN, Z., AND WANG, C. C. L. 2008. Automatic polycube-maps. In *GMP'08*, 3–16.
- LITKE, N., LEVIN, A., AND SCHRÖDER, P. 2001. Fitting subdivision surfaces. In *VIS '01*, 319–324.
- LOOP, C., AND SCHAEFER, S. 2008. Approximating catmull-clark subdivision surfaces with bicubic patches. *ACM Trans. Graph.* 27, 1, 1–11.
- LOOP, C., SCHAEFER, S., NI, T., AND CASTAÑO, I. 2009. Approximating subdivision surfaces with gregory patches for hardware tessellation. *ACM Trans. Graph.* 28, 5, 1–9.
- MICHIKAWA, T., KANAI, T., FUJITA, M., AND CHIYOKURA, H. 2001. Multiresolution interpolation meshes. In *PG '01*, 60–69.
- OLIVERIO, G. 2006. *Maya 8: Character Modeling*. Jones & Bartlett Publishers.
- ORGAZ, S., 2010. Xnormal. <http://www.xnormal.net/>.
- PIXOLOGIC, 2010. Zbrush. <http://www.pixologic.com/>.
- PRAUN, E., SWELDENS, W., AND SCHRÖDER, P. 2001. Consistent mesh parameterizations. In *SIGGRAPH '01*, 179–184.
- RAY, N., VALLET, B., ALONSO, L., AND LÉVY, B. 2009. Geometry aware direction field processing. *ACM Transactions on Graphics* 29, 1.
- TARINI, M., HORMANN, K., CIGNONI, P., AND MONTANI, C. 2004. PolyCube-Maps. In *SIGGRAPH '04*.
- TATARCHUK, N., 2008. Advanced topics in GPU tessellation. *Gamefest'08*.
- TONG, Y., ALLIEZ, P., COHEN-STEINER, D., AND DESBRUN, M. 2006. Designing quadrangulations with discrete harmonic forms. In *SGP '06*.
- WANG, H., HE, Y., LI, X., GU, X., AND QIN, H. 2007. Polycube splines. In *Proceedings of ACM symposium on Solid and Physical Modeling*, 241–251.
- WANG, H., JIN, M., HE, Y., GU, X., AND QIN, H. 2008. User-controllable polycube map for manifold spline construction. In *Proceedings of ACM symposium on Solid and Physical Modeling*, 397–404.
- YEH, I.-C., LIN, C.-H., SORKINE, O., AND LEE, T.-Y. 2010. Template-based 3d model fitting using dual-domain relaxation. *IEEE Transactions on Visualization and Computer Graphics accepted*.