PolyCut: Monotone Graph-Cuts for PolyCube Base-Complex Construction

Marco Livesu
Universita' di Cagliari

Nicholas Vining
University of British Columbia

Alla Sheffer
University of British Columbia

James Gregson
University of British Columbia

Riccardo Scateni
Universita' di Cagliari

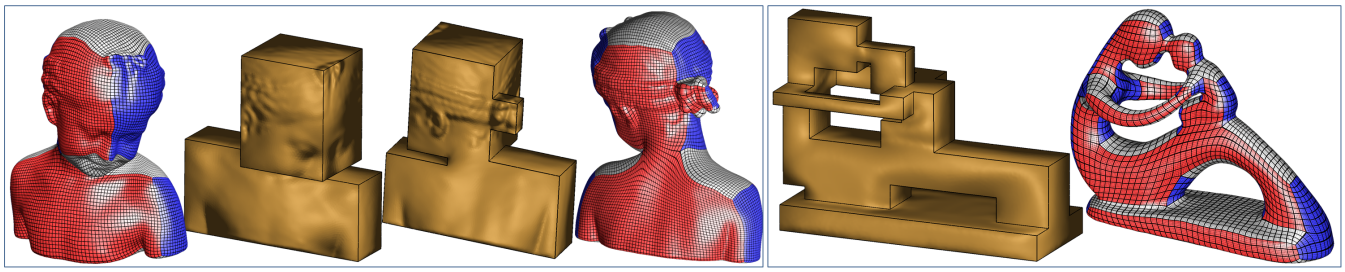


Figure 1: *PolyCut generated PolyCubes combine a small singularity count with low distortion. (Bimba con Nostra mesh provided courtesy of INRIA by the AIM@SHAPE model repository; fertility idol provided courtesy of Frank ter Haar by the AIM@SHAPE model repository.)*

Abstract

PolyCubes, or orthogonal polyhedra, are useful as parameterization base-complexes for various operations in computer graphics. However, computing quality PolyCube base-complexes for general shapes, providing a good trade-off between mapping distortion and singularity counts, remains a challenge. Our work improves on the state-of-the-art in PolyCube computation by adopting a graph-cut inspired approach. We observe that, given an arbitrary input mesh, the computation of a suitable PolyCube base-complex can be formulated as associating, or labeling, each input mesh triangle with one of six signed principal axis directions. Most of the criteria for a desirable PolyCube labeling can be satisfied using a multi-label graph-cut optimization with suitable *local* unary and pairwise terms. However, the highly constrained nature of PolyCubes, imposed by the need to align each chart with one of the principal axes, enforces additional *global* constraints that the labeling must satisfy. To enforce these constraints, we develop a constrained discrete optimization technique, *PolyCut*, which embeds a graph-cut multi-label optimization within a hill-climbing local search framework that looks for solutions that minimize the cut energy while satisfying the global constraints. We further optimize our generated PolyCube base-complexes through a combination of distortion-minimizing deformation, followed by a labeling update and a final PolyCube parameterization step. Our *PolyCut* formulation captures the desired properties of a PolyCube base-complex, balancing parameterization distortion against singularity count, and produces demonstrably better PolyCube base-complexes then previous work.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—;

Keywords: PolyCube, mesh parameterization, mesh segmentation

Links: DL PDF

1 Introduction

PolyCubes, or orthogonal polyhedra, are volumes bounded by axis-aligned planes. Recent research has highlighted the advantages of using PolyCubes as base-complexes for parametrizing closed surfaces and volumes for applications including surface texture mapping [Tarini et al. 2004; Yao and Lee 2008], hexahedral meshing [Gregson et al. 2011; Xia et al. 2010], trivariate spline fitting [Wang et al. 2007], and volumetric texturing [Chang and Lin 2010]. However, construction of quality PolyCubes that provide an optimal trade-off between parametrization distortion and chart or singularity counts remains a challenge. Consequently, most of the works above rely on manually or semi-manually constructed PolyCubes.

We improve on the state-of-the-art in automatic PolyCube base-complex computation by recasting the problem as one of mesh segmentation, or labeling, followed by PolyCube geometry extraction and subsequent parameterization. We observe that, by construction, a PolyCube parameterization re-orients triangle normals on the input mesh, aligning each normal with a signed principal axis direction. These per-triangle alignment choices define a segmentation of the surface mesh into charts, which map to axis-aligned faces of the PolyCube. We therefore formulate PolyCube base-complex extraction as a labeling computation which associates each input mesh triangle with one of six possible orientations minimizing the labeling cost subject to global constraints arising from the requirements to minimize distortion and ensure PolyCube validity. To compute the desired labeling we use a discrete optimization framework, we name *PolyCut*, that embeds multi-label graph-cut optimization within a local search algorithm that resolve these global constraints.

We demonstrate our new method on a large number of inputs and compare our results to the state-of-the art [Lin et al. 2008; He et al. 2009; Gregson et al. 2011], highlighting the improvement in terms of parameterization distortion, compactness, and singularity counts

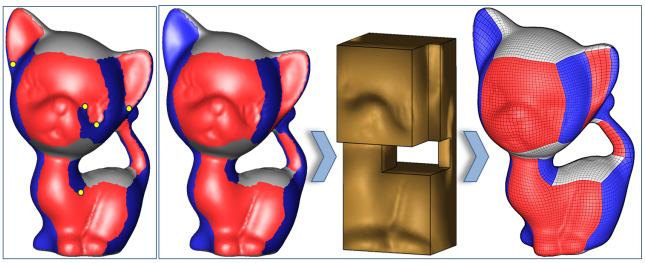


Figure 2: *(left) Segmentation with non-monotone boundaries (locations where boundary orientations change, or* turning points*, highlighted in yellow) leads to extreme mapping distortion; our alternative with all monotone boundaries (center) (color corresponds to axis direction: $\pm X$ - blue, $\pm Y$ - red, $\pm Z$ - gray) allows low distortion parameterization (right). (Kitten model provided courtesy of Frank ter Haar by the AIM@SHAPE model repository.)*

(Section 5). On average, our method reduces the number of singularities by 30%, while simultaneously improving the parameterization stretch. To demonstrate the advantages of using our better base-complexes, we use the resulting PolyCubes for hex-meshing, leading to better mesh quality than that provided by the most recent PolyCube based approach [Gregson et al. 2011].

Our key contributions that make this improvement possible are casting PolyCube computation as a multi-label graph cut problem, and introducing a discrete optimization algorithm capable of generating the desired constrained labelings that balance mapping distortion against singularity counts.

1.1 Problem Statement and Overview

Our main challenge is to compute a labeling that induces a low-distortion parametrization between the input model and the base-complex, while keeping both the number of singularities (chart corners), and the number of charts, low. In addition to reducing the number of singularities, coarser, or more **compact** segmentations, also allow users to reduce the element count for applications such as hex-meshing, or volume and surface fitting.

Segmentation compactness must be weighed against parameterization distortion. To predict the distortion for a given labeling, we consider both chart orientation and chart boundary shape. We expect charts to map to planar axis-aligned polygons with low parametric distortion, and to have ninety degree dihedral angles along chart boundaries. Consequently, as observed by Gregson et al. [2011], given a suitable coordinate system a local proxy of the distortion can be provided by measuring the angle between the normal of each triangle and the oriented axis it is associated with. We refer to this metric as geometric **fidelity**. Fidelity serves as a proxy for estimating the distortion caused by flattening each chart and rotating them so as to form the ninety degree dihedral angles. In PolyCube parameterization, an additional source of distortion comes from the need to map chart boundaries to the axis-aligned straight edges of the PolyCube. Thus, both shape and direction of the boundaries must be taken into account during labeling. In particular each boundary must have a uniquely defined direction with respect to the corresponding axis, defined by the cross-product of the normals associated with the adjacent charts. We therefore need to avoid *non-monotone* chart boundaries, ones where the direction as computed on the input segmentation switches sign (Figure 2, left), as in these cases "straightening" the boundary to map it to the corresponding PolyCube edge would require extreme distortion. We therefore aim to compute **all-monotone** labelings with no *turning points*, i.e. locations along boundaries where the boundary direction switches sign.

Finally, the chart connectivity has to define a **valid** PolyCube topology, or structure. As pointed out by Eppstein and Mumford [2010] the necessary and sufficient set of topological conditions on PolyCube validity remains unknown. Hence we only enforce the set of criteria listed in [Eppstein and Mumford 2010] as sufficient to provide valid PolyCube topology for genus zero objects. As they show, a segmentation will not produce a valid PolyCube if charts associated with opposite orientations of the same axis share a boundary, or if any chart has less than four neighbors. Additionally, one cannot guarantee that a PolyCube embedding of a graph exists unless each vertex in the graph has valence three, i.e. all segmentation corners must have valence three.

Overview Our *PolyCut* labeling algorithm is explicitly designed to address these requirements. We first observe that most of the properties above can be encoded locally: we can express fidelity via per-triangle labeling preferences, and can use binary terms relating pairwise label correlations between adjacent triangles to encode the preference for compact segmentations, ones that have shorter inter-chart boundaries, as well as some of the validity terms. A labeling that optimizes a weighted function of these terms can be computed via a graph-cut based multi-label optimization framework (Section 3.1), yielding segmentations which prove a good balance between compactness and fidelity. By minimizing chart boundary length, this computation also tends to reduce boundary curvature. However, it can, and does, introduce non-monotonicity along chart boundaries since, depending on input geometry, non-monotone boundaries (Figure 2, left) may better satisfy local fidelity (e.g. on the kitten's nose) and sometimes also be shorter (e.g. on his right ear) than monotone alternatives. While locally optimal, such boundaries are clearly undesirable. Since the graph-cut approach is purely local, it can also introduce charts with less than four boundaries.

To leverage the advantages of the graph-cut approach while satisfying the global monotonicity and boundary count constraints, we embed the graph-cut optimization as a local step within a discrete constrained optimization framework (Section 3, Figure 4). We start with an unconstrained solution and then use a specialized local search algorithm, based around the hill-climbing optimization technique [Hoos and Sttzle 2004], which searches for valid all-monotone segmentations while remaining close to the initial labeling. Each step of the algorithm locally perturbs the optimized fidelity term in the vicinity of turning points on the detected non-monotone boundaries, and re-applies the graph-cut optimization with a set of constraints aimed at minimizing deviation from the best obtained solution. The results of the relabeling are used to update the constraints, and to guide the location and magnitude of the next perturbation. The process terminates once all the non-monotone boundaries are resolved, and all charts are valid. We use the resulting constrained solution to characterize the axis orientation for each chart boundary, and modify the boundaries to better align with these orientations. The final base-complex PolyCube and the cross-parameterization are then generated using a combination of mesh deformation and distortion minimization (Section 4).

2 Related Works

Our work is related to research in several areas reviewed below.

PolyCube Parameterization and its Applications: PolyCube parameterization was introduced to graphics by Tarini et al. [2004] as a method for extending cube mapping to general shapes. PolyCube maps allow geometry and texture data to be efficiently stored as square or rectangular images, generalizing Geometry Images

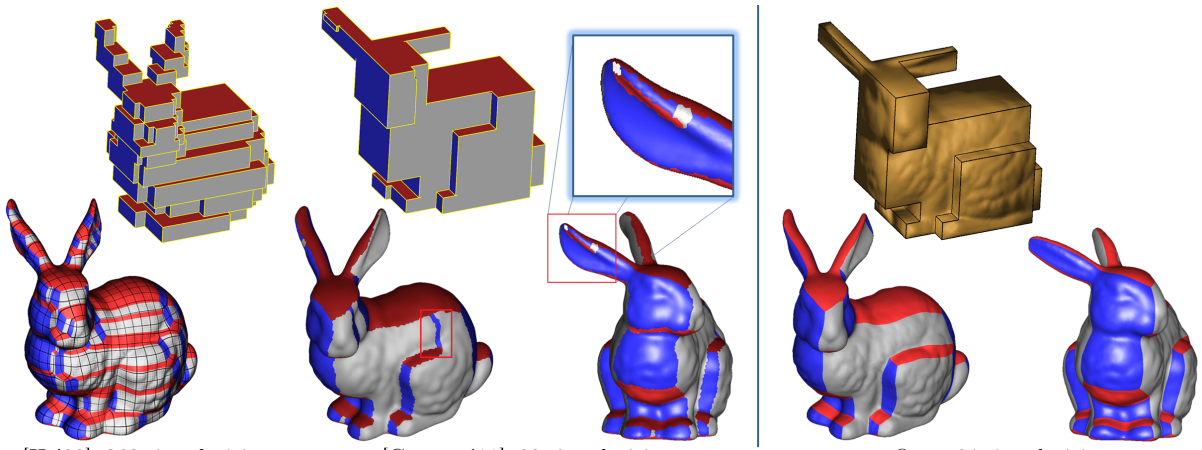


Figure 3: *Both He et al. [2009] and Gregson et al. [2011] generate PolyCubes with redundant or poorly placed charts (frames and zoom-in highlight unnatural/spurious charts) which introduce artifacts in downstream applications such as hex-meshing (Figure 14). Our method successfully balances compactness and mapping distortion. Statistics for all models included in Table 1. (Bunny model courtesy of Stanford Model Repository.)*

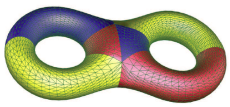
[Gu et al. 2002]. The advantages of PolyCube maps over irregular meshes include a more regular and compact representation, cache-friendly data access, easy texture filtering, and smooth chart boundaries for texturing. PolyCubes also make ideal base domains for GPU subdivision and multiresolution representation [Xia et al. 2011], and can be effectively used in quad and hex remeshing [Xia et al. 2010; Han et al. 2010; Gregson et al. 2011] as they can be trivially meshed with a regular quad or hex grid.

Given a PolyCube base-complex and a corresponding segmentation of the input model, a low distortion mapping between the PolyCube and the input can be computed via a number of both general [Tarini et al. 2004; Wang et al. 2007; Wan et al. 2011; Wang et al. 2008] and application-specific methods [Xia et al. 2010; Han et al. 2010; Li et al. 2010b].

Mesh Segmentation and Base-Complex Construction: Mesh segmentation is a well researched problem [Shamir 2008] with many methods aiming for some trade-off between an appropriate data, or fidelity, term and a related compactness metric. Similarly, parameterization over different base-complexes has a long history in mesh processing, see [Sheffer et al. 2006] and [Bommes et al. 2013] for surveys of construction of base-complexes with triangular and quad faces respectively.

What distinguishes our problem from those addressed by these bodies of work is the highly constrained nature of PolyCubes imposed by the need to align charts with one of the three principal axes, and consequently the unique validity and monotonicity constraints it enforces. To the best of our knowledge, global constraints such as those have not been tackled in previous segmentation or base-complex construction setups.

PolyCube Base Complexes: Despite the variety of applications that benefit from PolyCubes, most of the parameterization methods reviewed above rely on manually constructed base-complexes (i.e. [Xia et al. 2011]), with only a handful of methods addressing automatic construction. The first automatic PolyCube construction method [Lin et al. 2008] uses the Reeb graph of the input object and tends to produce overly coarse base domains that lack geometric fidelity resulting in excessive parameterization distortion. The voxelization approach of Wan et al. [2011] exhibits similar artifacts. Both methods lack means to refine the PolyCubes. The divide and conquer approach [He et al. 2009] captures the geometric features, leading to low distortion, but introduces multiple redundant singularities (Figure 3, left). Gregson et al. [2011] (Figure 3, center) generate PolyCubes using a combined deformation/segmentation algorithm based on rotating surface normals towards the principal axes. Although this method improves the domains considerably, the number of singularities remains unnecessarily high, largely due to a heuristic post-process the authors employ to improve monotonicity. As the authors acknowledge, the method is not guaranteed to produce a valid or all-monotone boundary PolyCube and in our experiments failed to do so on a number of models (e.g see the girl statuette in Figure 13). Our PolyCut method is explicitly designed to generate compact segmentations while satisfying fidelity, monotonicity and validity (Figure 3, right), generating better PolyCubes in terms of distortion and singularity counts when compared to these methods (Section 5, Table 1).



A related line of work [Li et al. 2010a; Li and Qin 2012; Li et al. 2012] considers coarse block-decomposition of low-frequency models for tri-variate spline fitting. These blocks define a quad base mesh of the model's surface, but in general do not correspond to a valid PolyCube embedding (left).

3 PolyCut Segmentation

Our algorithm takes a triangle mesh as input and outputs a corresponding PolyCube base-complex and a cross-parametrization between the two. Our key insight is that while directly optimizing for a PolyCube base-complex that minimizes distortion is hard, it suffices to optimize for the four criteria defined in Section 1, using fidelity and monotonicity as a reasonable proxy for mapping distortion. To this end, we decouple the problem of finding the best distortion-minimizing parameterization from the generation of the PolyCube segmentation and treat them as two separate processes (Figure 4). It is instructive to think of the first step, described here, as a *topological* step, which produces a PolyCube segmentation, and the second step as a *geometric* step which takes the segmentation and uses it to generate the actual PolyCube geometry and the parameterization with respect to the input mesh (Section 4).

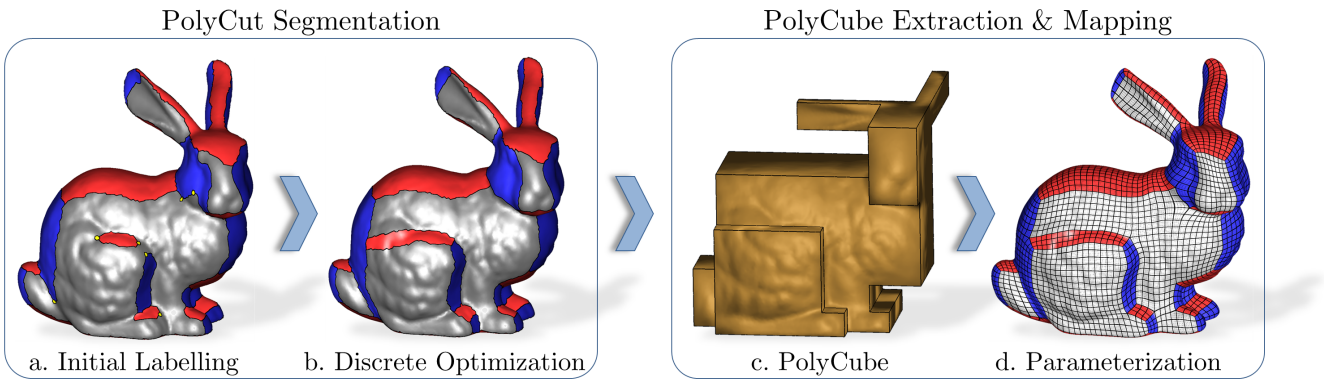


Figure 4: *Algorithm Stages: The first, topological, step of our method produces a good quality PolyCube segmentation of the input mesh (left), and is followed by a geometric step which defines the PolyCube vertex positions and a cross-parameterization between this PolyCube and the input mesh (right). Our topological step first uses multi-label optimization to compute a segmentation that satisfies our compactness, fidelity, and local validity constraints, but can contain non-monotone boundaries (turning points highlighted in yellow). It then uses discrete optimization to resolve these boundaries and generate a valid segmentation.*

We observe that most of the requirements for a good PolyCube segmentation can be expressed locally, and involve a condition on either one triangle, or two adjacent triangles. In particular, we note that two of the Steinitz criteria for orthogonal polyhedra [Eppstein and Mumford 2010] can be expressed as purely local constraints. Unfortunately, monotonicity is not a condition that can be formulated locally, as detecting a change in a boundary's orientation with respect to the corresponding axis requires at the very least considering two adjacent mesh edges. Moreover, such purely local evaluation may provide multiple false positives as it will depend on the local mesh connectivity. Once we have a labeling, however, the situation improves: we can robustly detect when a chart boundary is non-monotone (see Section 3.2.1), and identify the *turning points*, see Figure 5, where the boundary orientation changes. Moreover, as shown in this figure, such boundaries typically can be resolved by locally adjusting the segmentation in the problematic areas.

Similarly, one cannot *a priori* locally optimize for, or even evaluate, the number of chart boundaries before a labeling is computed. Once a labeling exists, however, counting them is trivial. In practice charts with less than four boundaries either have non-monotone boundaries and are seamlessly resolved once these boundaries are corrected (Figure 5) or are small enough to be simply discarded. Thus, they require minimal special processing in our framework.

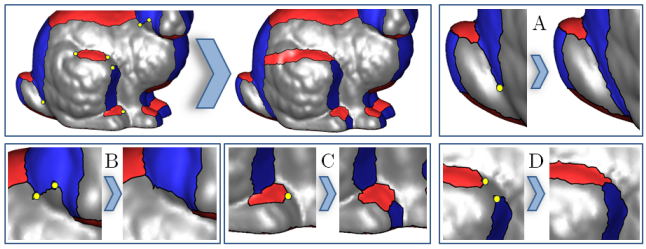


Figure 5: *Zooming-in on a set of non-monotone boundaries in an unconstrained segmentation of the bunny (Figure 4) and their monotone solutions: A. Extending the non-monotone boundary to reach another boundary; B. straightening a boundary; C. introducing a new chart with different label; D. Extending two non-monotone boundaries to reach one another. Note that the list is not exhaustive as multiple turning point configurations can exist within one chart or even along a single boundary curve.*

Motivated by these observations, we use multi-label graph-cut optimization as a building block in our algorithm, employing it to compute locally optimal segmentations within a global discrete constrained optimization framework. The goal of this constrained optimization framework is to enforce monotonicity and validity while minimally increasing the fidelity and compactness costs compared to an unconstrained solution. We observe that in order to enforce monotonicity with minimal cost increases, we must support a variety of topological strategies for resolving each detected turning point (Figure 5): e.g. straightening the offending boundary portion, extending the boundary until the turning point touches another chart and the boundary gets split in two, or even adding a new chart with a different label from those sharing the non-monotone boundary. The optimal choice depends on the local geometry, and may involve complex interplay between nearby non-monotone boundaries (e.g. Figure 5, D). Instead of a set of rigid heuristics, such as the one used by the post-process in [Gregson et al. 2011] we opt for a solution guided by the input data, and in particular the fidelity term which encodes the local surface geometry. In particular, we observe that the labeling algorithm would automatically resolve non-monotone boundaries if we **locally** bias the fidelity cost around the turning points in an appropriate direction (Figure 6). We therefore search for the minimal local bias, in terms of area of impact and magnitude, that achieves this goal for each turning point.

The high-level framework we adopt can be summarized as follows. We start by computing an unconstrained locally optimal labeling (Section 3.1). We then use this labeling as a starting point for a local search that resolves any non-monotone boundaries that the labeling contains, while minimally deviating from the input solution. This search explores possible local changes in the fidelity terms around the turning points and repeatedly reruns the labeling following each such change and re-evaluates the result prior to deciding on the next set of changes (Section 3.2). To facilitate convergence and avoid introducing new turning points, after each labeling step we evaluate the validity and monotonicity of the produced charts and prevent the subsequent labeling steps from changing the shape of any valid charts with all monotone boundaries. The process terminates once all charts are deemed valid and monotone. This process can be seen as a special case of the classical hill-climbing [Hoos and Sttzle 2004] discrete optimization framework.

The hill climbing approach produces a locally near-optimal result which satisfies global constraints, but is not designed to explicitly consider parameterization distortion, and in particular bound-

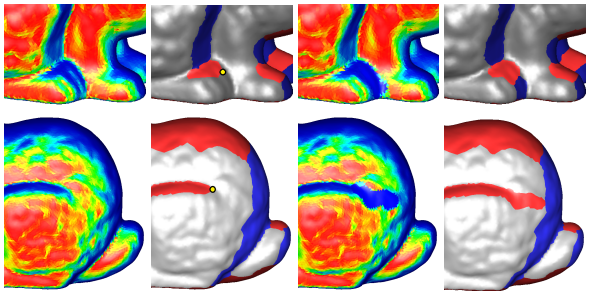


Figure 6: *Selective biasing, sometimes minuscule (top) and sometimes more significant (bottom), of the fidelity term around turning points resolves non-monotonicity. (top) Left to right: original fidelity costs for positive X (gray) assignment and a turning point on the initial segmentation of bunny's paw. Slightly increasing this cost near the turning point resolves the boundary. (bottom) Same process for back thigh requires larger increase to be propagated across for the turning point to be resolved.*

ary alignment with the corresponding oriented axes. To improve the segmentation we use the assigned labels to detect preferred boundary orientations, and then fine-tune chart boundaries while keeping the chart topology intact (Section 3.2.2).

3.1 Graph-Cut Based Labeling

The graph cut algorithm described here provides the starting point, or initial guess, for the local search labeling framework. We express the fidelity versus compactness trade-off as an energy minimization problem, in which each triangle t must be assigned one of six discrete labels. Each label $s \in \{-X, +X, -Y, +Y, -Z, +Z\}$ represents a signed normal direction $\vec{s}$ along the corresponding axis. We search for a segmentation S which minimizes the energy $E(S)$:

$$E(S) = \sum_{t \in T} F_t(s_t) + c \cdot \sum_{pq \in E} C_{pq}(s_p, s_q) \qquad (1)$$

subject to local validity constraints.

The unary fidelity term $F_t(s_t)$ describes the cost of assigning the label s_t to a triangle t, while the binary compactness term $C_{pq}(s_p, s_q)$ is the cost of assigning the label s_p to a triangle p and the label s_q to a triangle q. The constant c serves as a user control for the overall compactness of the PolyCube; the higher the value of c, the more compact the resulting PolyCube becomes. We set $c = 3$ unless otherwise specified (Section 5).

We now describe the fidelity and compactness terms in detail. The cost of assigning a triangle t to a given label s is measured as a function of the angle between the triangle normal $\vec{n}_t$ and the label direction $\vec{s}$:

$$F_t(s) = 1 - e^{-\frac{1}{2}\left(\frac{\vec{n_t} \cdot \vec{s} - 1}{\sigma}\right)^2} \qquad (2)$$

Setting $\sigma = 0.2$, using the three-sigma rule, yields a labeling cost that goes from 0, when the triangle normal and label direction are perfectly aligned, to close to 1, when the angle between the normal and the label direction is approximately 65 degrees. Note that a normal that is equidistant to each of the X, Y and Z axes will be at a 55-degree angle to each of them; our choice of maximum penalty is designed to weakly differentiate between labeling costs when these angles are close to 55. Example fidelity terms, and a labeling generated using the fidelity term alone, are shown in Figure 7.

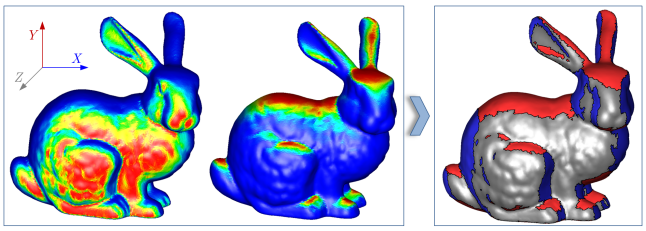


Figure 7: *The graph cut method weighs fidelity against compactness. (left) The fidelity term expresses the cost of assigning a triangle to one of the six principal axis directions (here, shown for the positive Z (gray) and positive Y (red), cost range is from red (zero) to one (blue)). (right) Labeling using fidelity alone produces noisy results which may be better in some areas of the model but much worse in others compared to the result in Figure 4a, obtained by balancing fidelity against compactness.*

The compactness term is designed to minimize boundary length. Hence we set the cost $C_{pq}(s_p, s_q)$ for two triangles sharing a common edge to 0 if the two triangles share the same label. For adjacent triangles assigned two different labels, we set the cost to a function of the dihedral angle between the two:

$$C_{pq}(s_p, s_q) = e^{-\frac{1}{2}\left(\frac{\vec{n}_p \cdot \vec{n}_q - 1}{\sigma}\right)^2} \qquad (3)$$

We use this function, instead of a constant cost, to weakly guide boundaries to align with geometric features. Using $\sigma = 0.25$, the cost is 1 for coplanar triangles going down to e^{-8} for 90 degree angles.

To compute the labeling that minimizes our energy function (Equation 1), we use a graph-cut based multi-label optimization framework (http://vision.csd.uwo.ca/code/gco-v3.0.zip) that implements a combination of the methods in [Boykov et al. 2001; Kolmogorov and Zabih 2004; Boykov and Kolmogorov 2001].

Validity Two of the validity constraints described in Section 1 and in [Eppstein and Mumford 2010] can be directly accounted for by this local optimization process. We explicitly prevent two triangles p and q sharing an edge or a vertex from being assigned to opposite direction labels $-L$ and $+L$ by using the pairwise constraint mechanism provided by the optimizer. To eliminate chart corners with valence above three, following the generation of a labeling, we first detect such corners and then rerun the labeling with additional constraints that split each such corner into two. To generate the split we choose the most prevalent chart label, resolving ties by preferring the label with largest area in the corner's one-ring, and force all triangles in the one-ring to have that label (setting the fidelity cost to zero for this label, and to infinity for all others). In practice, we never needed to run this step more than once.

3.2 Iterative Local Search via Hill Climbing

As previously noted, an initial segmentation generated by the above method possesses most of the properties we would like a PolyCube segmentation to have, and is close, in the space of all possible labelings, to a similar valid solution with monotone boundaries. The question we face is how to obtain such a segmentation in a principled way. Analyzing the results, we observe that we can achieve the desired output using our original labeling algorithm after perturbing the per-triangle, per-label, fidelity costs $F_t(s)$ (Equation 2) in the vicinity of the turning points in the initial segmentation (Figure 6).

Our challenge, therefore, is to compute the suitable local perturbations of these costs that result in a valid all-monotone segmentation. Thus instead of resolving turning points by directly editing the segmentation, we resolve them indirectly by tweaking the per-triangle fidelity costs, searching for a set of small local changes that leads to a valid, all-monotone segmentation.

This is a classical search problem, with a search space that is exponentially large with respect to the number of input triangles. To make this problem tractable, we embed our labeling algorithm within an iterative local search, or 'hill climbing', framework, which guides the perturbations toward promising local solutions. At each step of this framework we first detect all currently valid charts with all-monotone edges and 'freeze' them in subsequent hill-climbing steps. Specifically, we remove the frozen chart's triangles from any subsequent labeling operations, and add constraints to prevent any triangle that shares an edge or vertex with the frozen chart from being assigned either the same, or the opposite, label. The first constraint prevents the chart from continuing to grow and potentially introducing new non-monotone boundaries, whereas the second constraint prevents the formation of invalid chart connectivity. We then proceed to incrementally change the per-triangle fidelity cost in the vicinity of the remaining turning points and reapply the labeling algorithm after each such edit. Our algorithm terminates once all chart boundaries are classified as monotone. In the unlikely event that we terminate and still have a chart with less than four boundaries, it is merged with one of its neighbors.

When exploring possible perturbations, we do not know *a priori* what change will best resolve a given turning point. For every turning point we have the option of increasing, or decreasing, the fidelity cost with respect to any of the six labels. Brute-force searching for the best direction to bias each turning point is exponential with respect to the number of turning points to be resolved. In order to reduce the size of our search space, we create six search branches $\{X_{less}, X_{more}, Y_{less}, Y_{more}, Z_{less}, Z_{more}\}$, each of which expresses a preference towards, or away from, a given axis. We seed each of these branches with our initial labeling, then introduce an additive bias $\delta = 0.01$ to the fidelity term around each active turning point within a radius of 5% of the bounding box diagonal length. When selecting δ we weigh speed against accuracy - larger values speed up computation but can reduce result quality. Searching each branch in a fixed direction reduces the number of elements in the search space from $O(6^{|t|})$ to $O(|t|)$, or from exponential to linear time. While we have six label options, we introduce one branch per-axis instead of one per direction (plus or minus) as in practice only one of these directions is relevant for any given turning point. Also note that increasing the fidelity cost for the $\pm$ X labels is the same as simultaneously decreasing it for the $\pm$Y and $\pm$Z labels.

Since different turning points may be resolved by different branches, we need a communication strategy between them. We introduce two such strategies, a *fast* strategy that works for most inputs we experimented on, and a much slower but more robust *restartable* one. In the fast strategy, after updating the fidelity costs and applying relabeling independently within each branch, we detect all newly-valid charts with all-monotone edges and 'freeze' them in subsequent labeling steps across *all* branches. The rest of the processing remains independent per branch. In the restartable strategy, in addition to detecting newly valid charts, we also detect when any existing chart is split into two, indicating that a turning point had been eliminated by splitting a non-monotone boundary into two. When either event occurs, we restart the hill-climbing process using the current labeling, frozen charts, and perturbed fidelity terms as the initial input for all branches. The restartable strategy allows for finer-level perturbation control, necessary for resolving charts which require one perturbation direction along one boundary and an opposite direction along another (Figure 9). In practice such charts are quite rare; the kiss was the only example which required the restartable strategy.

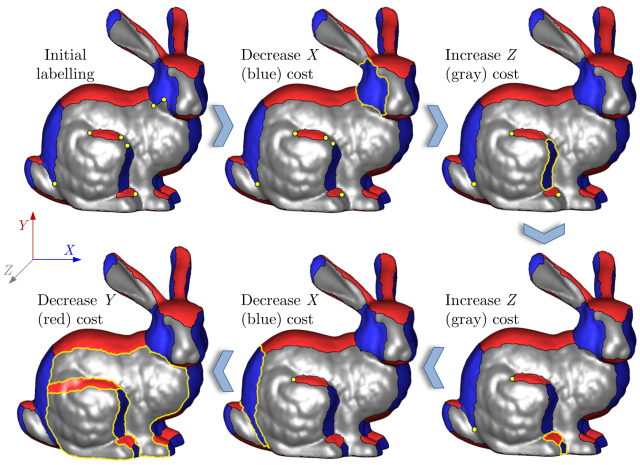


Figure 8: *Hill-climbing visualization: we show the steps at which valid, monotone charts are detected and frozen. Depending on the input complexity it takes the method between a few dozen to a few hundred iterations to converge. Note that as more charts are frozen the labeling problem size shrinks speeding computation. Newly frozen charts highlighted with yellow border.*

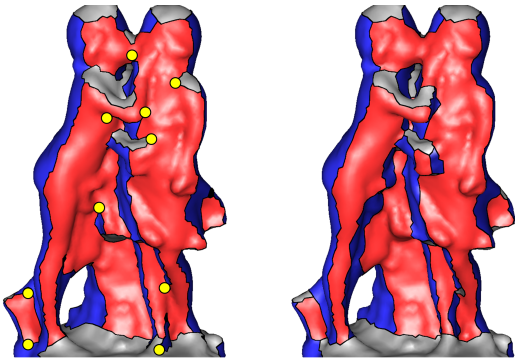


Figure 9: *Left: The highly complex red chart needs opposite bias directions in different regions for a good solution - a task the 'fast' climbing method is unsuited for. Right: Restarting the search after each topological change provides the desired flexibility.*

3.2.1 Computing Turning Points

To detect non-monotone chart boundaries we need to compute the signed orientation of edges along the boundary, using a consistent notion of right and left charts. A change in sign indicates a non-monotone boundary. Assigning orientation at an individual edge level is sensitive to minor changes in local mesh connectivity. To obtain more robust results, we extract turning points using the same graph-cut framework as above, this time with two labels, positive and negative. The unary term is a Gaussian falloff function of the dot product of the edge and axis directions ($\sigma = 0.9$) and the binary term for consecutive edges is zero when assigned the same level and a Gaussian of the dot product of the edge directions ($\sigma = 1.2$) otherwise. Following the labeling, boundaries with more than one segment are classified as non-monotone and the segment joints are marked as turning points.

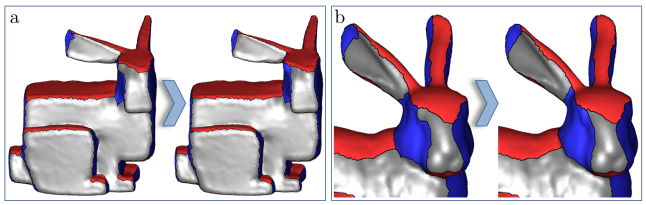


Figure 10: *We use PolyCube deformation (left) to reveal boundary alignment directions and use those to improve the segmentation boundaries. This process improves chart boundaries on the bunny's face (right).*

3.2.2 Boundary Optimization

While the hill-climbing framework aims for short and monotone boundaries, it makes no explicit effort to align boundaries with their corresponding axial directions. The reason being that while we can control for labels assigned to pairs of adjacent triangles during the segmentation process, we do not *a priori* know the *orientation* of each boundary, i.e. the order of its two corners along the axis it maps to. Naively introducing an unordered axial bias into the original formulation would only lead to an increase in non-monotonicity, as it will encourage longer, axially-aligned, but erratic boundaries at the expense of straighter, but misaligned ones. Given a segmentation computed in the hill climbing step, we can estimate the desired orientation of the boundaries and use these orientations to improve the segmentation boundaries.

Our challenge, as always, is to balance the boundary shape against fidelity. However we can no longer use the original unbiased fidelity term as using it will aim to revert the gains of the entire hill-climbing process. While it is theoretically possible to keep track of the individual changes to the fidelity costs introduced during the climbing step, combining those in a coherent way across the multiple branches can be difficult.

We note that computing fidelity (Equation 2) on an approximate PolyCube suggested by the current labeling provides a good proxy for this biased cost. To generate this approximate PolyCube, we use the iterative deformation framework of Gregson et al. [2011] with some changes detailed below. We then use the computed fidelity, and an updated compactness term which reflects boundary orientation, in a relabeling framework that improves the alignment of chart boundaries with the assigned orientations while still preserving validity and monotonicity (Figure 10).

PolyCube Deformation: Given an input mesh and a PolyCube segmentation that assigns an orientation to each triangle normal, we seek to deform the input by rotating the normals toward these orientations, obtaining an approximate PolyCube. We use soft rather than hard rotational constraints, as the current segmentation is clearly not final, and therefore we aim to balance normal rotation against input distortion.

Similar to Gregson et al. [2011], to avoid self-intersections and better preserve the input structure, we use a volumetric mesh of the input model for the deformation. Since any interior vertex in a chart has a one-ring consisting purely of triangles with the same labeling, it has a known assigned orientation. We compute the minimum rotation for each vertex that aligns the vertex normal with its target orientation. We then propagate these rotations to vertices along the chart boundaries, and through the volumetric mesh interior. We use these rotations to compute the vertex positions in the deformed mesh, attempting to orient each edge in its new preferred direction while maintaining its length. Given original vertex coordinates v_i and v_j with rotation matrices ∇_i and ∇_j, we find new coordinates u_i and u_j by minimizing:

$$\sum_{i,j}((u_j - u_i) - (\frac{\nabla_i + \nabla_j}{2}) \cdot (v_j - v_i))^2 \qquad (4)$$

over all mesh edges (i, j). Since the vertex positioning step only weakly satisfies the target rotations, we repeat the process two more times to obtain a sufficiently clear PolyCube. We will use this framework later, with additional refinements, to compute final PolyCube geometry (see Section 4).

Chart Boundary Optimization We use the deformed models and the orientation information to improve boundary alignment. Since we aim to preserve the segmentation topology intact, we use a process where we iteratively apply the relabeling step to small portions of the segmentation, while preserving the topology. Specifically, we repeatedly relabel pairs of charts that share boundaries and triplets of charts that share corners. To prevent topological changes, when applying the method to pairs of charts, we fix the labels on the triangles along the non-shared boundaries of these charts, and set the cost of assigning any label but the two participating ones to infinity. For similar reasons, for each corner we consider all triangles within a radius of $\frac{1}{3}$ of the smallest incident chart boundary's length. Any label outside of this radius is fixed. We similarly restrict ourselves to considering only the three chart labels incident to the corner in question. While these constraints drastically reduce the likelihood of topology changes, they do not fully prevent them. If such a change does occur, we roll-back the relabeling result, leaving the boundary as-is.

The labeling framework uses the same fidelity term as in Section 3.1, but with the fidelity computed on the deformed model. The compactness term is similarly computed on the deformed model, but is now modified to take into account how well a given boundary edge is aligned with its target orientation:

$$C_{pq}(s_p, s_q) = 1 - e^{-\frac{1}{2}\left(\frac{\vec{e}_{pq} \cdot \vec{d} - 1}{\sigma}\right)^2} \qquad (5)$$

Here the edge direction $\vec{e}_{pq}$ is given as per Section 3.2.1. Fixing a 'start' and 'end' vertex for each boundary that are consistent with the axis orientation and our notion of right and left charts, the boundary direction $\vec{d}$ is given by the outgoing edge vector from the starting vertex of the boundary in question, with respect to the current chart.

4 PolyCube Positioning and Parametrization

Once we have a PolyCube segmentation, we must still generate the corresponding base-complex and parameterization. We extract the PolyCube geometry from the segmentation using the deformation framework of Section 3.2.2, augmenting it with hard constraints to ensure an exact PolyCube output. We first repeat the iterative process in Section 3.2.2, but add planarity constraints to Equation 4 minimizing, for every surface edge, the difference between its endpoint coordinate values along the relevant axis. We use gradual deformation with soft constraints so as to minimize distortion. Once the process converges, we compute the final solution by adding hard planarity constraints forcing vertices in each chart to have the same coordinate value along the corresponding axis.

The end result of this deformation is that each corner vertex on our PolyCube is now in its correct position however, the positions computed for the rest of the vertices are not guaranteed to be on the PolyCube defined by these corners. To compute a low-distortion parameterization from the input mesh to the PolyCube

requires parameterizing each chart into a fixed, possibly, concave planar polygon, a well known open problem in mesh processing [Xu et al. 2011]. To obtain a low distortion mapping, sidestepping this challenge, we first compute a bijective but possibly poor quality map from the input model to the PolyCube, and then improve the mapping by operating in the opposite direction, i.e. from the PolyCube to the input model.

To generate the initial map, we first map each chart boundary to its corresponding PolyCube edge using arc-length parameterization. We then use the method of [Xu et al. 2011] with mean-value coordinates to position the interior chart vertices. If we require a volumetric parameterization, we re-use the deformation framework, keeping surface vertex positions fixed and specifying surface rotations using a coordinate frame given by the new normal and one of the edges.

For applications such as seamless texturing or meshing, the corners of the PolyCube need to be placed at vertices of a fixed size grid. To perform such quantization, we first place corner vertices in the quantized locations. We then relocate the PolyCube surface vertices using mean-value coordinates [Hormann and Floater 2006] with respect to the corners of its corresponding PolyCube face. Finaly the interior vertices are similarly relocated using the surface mesh as a cage for 3D mean-value coordinates [Ju et al. 2005].

To compute a low distortion PolyCube to input map, we first remesh the PolyCube using existing software [Alice Project-team] and use the mapping which we have just computed to project the PolyCube mesh to the input model. We then iteratively slide the projected vertices along the input surface in order to minimize the mapping distortion between the two meshes, measured using mean-value coordinates. We note that once we have the PolyCube and the initial map, there are multiple methods for improving the parametrization. While our technique is effective, we include it for completeness only.

5 Results

We tested our method on over a dozen diverse inputs shown throughout the paper, including both natural and engineered shapes, and were able to generate valid, all-monotone boundary segmentations and produce suitable PolyCube base complexes on all these inputs. Figures 2, 4, and 11 demonstrate the power of our discrete optimization framework, highlighting both the prevalence and the variety of non-monotone boundaries in unconstrained solutions, and their appropriate resolution by our hill-climbing process. The statistics for the models are summarized in Tables 1 and 2. Unless otherwise specified, all the results were produced using the default parameters described in the text. For the bimba and lion (Figure 13) we used a compactness factor c of 4, and for the armadillo (Figure 11,d) a factor of 4.5; for the rest, the factor was set to 3. For all the models, except the 'kiss' (Figures 9 and 11, e) we used the fast climbing mode described in Section 3.2. The runtime using this method varies between one minute for largely axis-aligned models like the bumpy torus (Figure 11, c) which contain few non-monotone boundaries in the initial labeling, to ten minutes for models such as the bunny which have many off-axis features. The kiss model, which was generated with the restartable hill-climbing strategy (Section 3.2), took 50 min to compute. Times were measured on a MacBook Air (1.7Ghz Intel Core i5, 4GB RAM). The runtimes are comparable with those of recent PolyCube segmentation [Gregson et al. 2011] and hex meshing [Lévy and Liu 2010] methods.

One important advantage of our approach, compared to methods such as [Lin et al. 2008; Wan et al. 2011; Gregson et al. 2011], is the flexible control of the trade-off between fidelity and compactness. Figure 12 shows the results of varying the compactness factor (Equation 1) on the PolyCubes computed for the bunny and girl models. As expected, increasing this factor gradually reduces the number of PolyCube faces at the expense of increased stretch/area distortion (Table 2). Providing this control to the user is important for many applications where distortion is weighed against singularity counts, such as hex meshing or atlas generation.

Model	size # tri.	# t.p.	# corners	# charts	ang./area dist.	stretch
bunny (Tarini'04)			67	34	1.069/1.034	0.913
bunny (Lin08)			34	19	1.12/1.15	0.79
bunny (He'09)			363	206	1.007/1.068	0.871
bunny (Gregson'11)			88	46	1.055/1.077	0.820
bunny (ours)	56k	10	64	34	1.033/1.033	0.908
lion (He'09)			285	151	1.028/1.100	0.804
lion (ours)	57k	17	74	39	1.073/1.059	0.831
squirrel (He'09)			40	24	1.001/1.084	0.854
squirrel (ours)	53k	3	16	10	1.035/1.042	0.890
bimba (Gregson'11)			115	61	1.100/1.106	0.712
bimba (ours)	46k	6	30	17	1.061/1.053	0.843
girl (Gregson'11)			88	48	1.064/1.112	0.747
girl (ours)	72k	16	60	34	1.043/1.062	0.850
fertility (Gregson'11)			96	43	1.074/1.100	0.772
fertility (ours)	37k	6	94	42	1.092/1.066	0.808
kiss (Gregson'11)			156	74	1.087/1.090	0.770
kiss (ours)	29k	19	120	56	1.047/1.049	0.871
rocker arm (Gregson'11)			70	38	1.082/1.066	0.819
rocker arm (ours)	27k	6	62	34	1.066/1.051	0.857
carter (Gregson'11)			153	82	1.040/1.051	0.868
carter (ours)	69k	11	132	71	1.073/1.029	0.870
bumpy torus (Gregson'11)			208	104	1.093/1.069	0.793
bumpy torus (ours)	40k	4	172	86	1.041/1.053	0.865
armadillo (Tarini'04)			80	42	1.318/1.224	0.577
armadillo (ours)	56k	26	140	72	1.105/1.130	0.714
Kitten (ours)	37k	7	32	16	1.099/1.053	0.820

Table 1: *PolyCube statistics, including a comparison to earlier methods. Left to right: triangle count, number of turning points in the initial segmentation (our method), number of singularities, number of charts, angular and area distortion [Tarini et al. 2004], stretch [Praun and Hoppe 2003]. For all metrics the optimal value is one. Our method consistently introduces a smaller number of singularities while producing parameterization with better stretch and area distortion.*

Comparisons: We compared the PolyCubes generated by our method to those created via existing alternatives. Our bunny PolyCube computed with the compactness factor set to 6 (Figure 12) is quite similar to the output of Lin et al., and has comparable distortion (Tables 1 and 2). However, in contrast to PolyCut, neither [Lin et al. 2008] nor [Wan et al. 2011] support other PolyCube resolutions.

When compared to recent techniques [He et al. 2009; Gregson et al. 2011], our method is able to generate PolyCubes with significantly fewer singularities and charts, while achieving comparable or better stretch and area distortion (Figures 3, 13, Table 1). For Gregson et al., we measured distortion on their output quad meshes - directly measuring distortion on their PolyCubes produces infinite values

Model	compactness	#corners	#charts	ang./area dist.	Stretch
girl	3	60	34	1.044/1.062	0.850
girl	6	32	19	1.074/1.176	0.691
girl	10	24	14	1.068/1.305	0.583
girl	17	8	6	1.170/1.131	0.652
bunny	3	64	34	1.033/1.033	0.908
bunny	6	34	19	1.068/1.104	0.750
bunny	10	16	10	1.062/1.184	0.683
bunny	17	8	6	1.103/1.197	0.657

Table 2: *Impact of adjusting the compactness factor (Figure 12).*

model model	Gregson [2011] S. Jac. min/average	PolyCut S. Jac. min/avg	PolyCut min S. Jac. improv factor
fertility	.196/.911	.259/.872	30%
bunny	.138/.930	.274/.938	100%
rockerarm	.226/.899	.370/.890	64%
girl/BU	.235/.925	.401/ .926	71%
carter	.177/.823	.250/.894	41%

Table 3: *Hex-meshing quality comparison to [Gregson et al. 2011]. Left to right: minimal and average scaled Jacobian for the two methods, improvement factor for minimal Jacobian. Our method dramatically increases the worst element quality, a critical metric for analysis accuracy.*

due to collapsed or flipped triangles. The reduction in singularity numbers is particularly significant in models with many off-axis features, such as the bimba (Figure 13, right) where our method reduces the number of singularities by a factor of five (from 115 to 30) compared to [Gregson et al. 2011] while simultaneously reducing distortion, or the lion (Figure 13, left) where we reduce the singularity count by a factor of four (from 285 to 74) compared to [He et al. 2009] while improving stretch and area distortion (our result has a marginally higher angular distortion). Overall, when compared to these methods we observe, on average, a reduction of around 30% in the number of singularities. In contrast to Gregson et al.[2011], our method resolves all non-monotone boundaries, while theirs may leave those in place, e.g. see the forehead of the girl statuette in Figure 13. We speculate that the major factor in the observed improvement is the use of a principled local search to eliminate turning points, as opposed to the local heuristic used by Gregson et al., which attempts to fix turning points individually, choosing from three templated solutions.

We had also compared our method against the manually generated PolyCubes of Tarini et al. [2004] (Table 1). On the bunny model our method produces a PolyCube with nearly identical complexity and comparable distortion statistics. On the armadillo model our method produces a much finer PolyCube, one which as expected leads to less distortion. Producing such a high detail PolyCube manually would be a challenging task.

Lastly, to showcase the importance of using better PolyCubes from an application perspective, we used our PolyCubes as input to the hex meshing method of [Gregson et al. 2011] (Figure 14). The improvement on the bunny highlights the importance of correctly resolving non-monotone boundaries. While the heuristic approach of Gregson produces an unnatural vertical chart on the bunny's side (Figure 3) leading to visible mesh artifacts, our method resolves the thighs in a more natural manner. For the fertility model the improvement is a result of better alignment between boundaries and their corresponding axis directions. The element quality statistics are listed in Table 3. As pointed out by Gregson et al., the critical number to look at is the minimal Jacobian, which is significantly better for all our results.

6 Conclusions

We have presented PolyCut, a novel method for generating PolyCube base-complexes via constrained multi-label optimization. Our method generates high quality results, with low singularity counts and small distortion, across a wide range of models. We offer a user-controllable balance between compactness and parametric distortion via the compactness factor. We significantly outperform previous work in terms of PolyCube compactness, without compromising mapping distortion. As our method is fully automatic and robust across a wide range of inputs, applications requiring PolyCubes can now be developed without a need for time-consuming manual or semi-automatic schemes.

Limitations Like in previous PolyCube computation methods, our outputs depend on the initial orientation of the input model and it would be worthwhile to explore computing optimal orientations automatically, e.g. via PCA analysis of surface normals. Following [Eppstein and Mumford 2010] we require every corner in the segmentation to have valence three, a sufficient but not necessary requirement for PolyCube existence, which sometimes leads to formation of redundant charts. Unfortunately, to our knowledge, no weaker sufficient requirement exists. Lastly,the local search method we use is an approximation method, and as such is not guaranteed to converge to a valid solution or find the best possible PolyCube segmentation for a given mesh (as represented by an exhaustive search).

Future Work Currently, it is up to the user to select a compactness factor that fits their needs via trial and error. Directly relating this factor to mapping distortion is a challenging open problem. Future work also involves the integration of other global search frameworks into our PolyCut algorithm. Better PolyCubes may be generated via stochastic methods or by adding random restarts to our hill climbing. We also feel that our hybrid framework approach can be used to extend multi-label optimization to work on other problems in geometric mesh processing and computer graphics.

Acknowledgements We would like to thank AIM@SHAPE (`http://shapes.aim-at-shape.net`) for the use of their models. Financial support was provided by NSERC, MITACS NCE, GRAND NCE, and UBC 4YF.

References

ALICE PROJECT-TEAM. Graphite. `http://alice.loria.fr/software/graphite/`.

BOMMES, D., LÉVY, B., PIETRONI, N., PUPPO, E., SILVA, C., TARINI, M., AND ZORIN, D. 2013. Quad-mesh generation and processing: A survey. *Computer Graphics Forum*.

BOYKOV, Y., AND KOLMOGOROV, V. 2001. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 359–374.

BOYKOV, Y., VEKSLER, O., AND ZABIH, R. 2001. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23*, 2001.

CHANG, C.-C., AND LIN, C.-Y. 2010. Texture tiling on 3d models using automatic polycube-maps and wang tiles. *J. Inf. Sci. Eng. 26*, 1, 291–305.

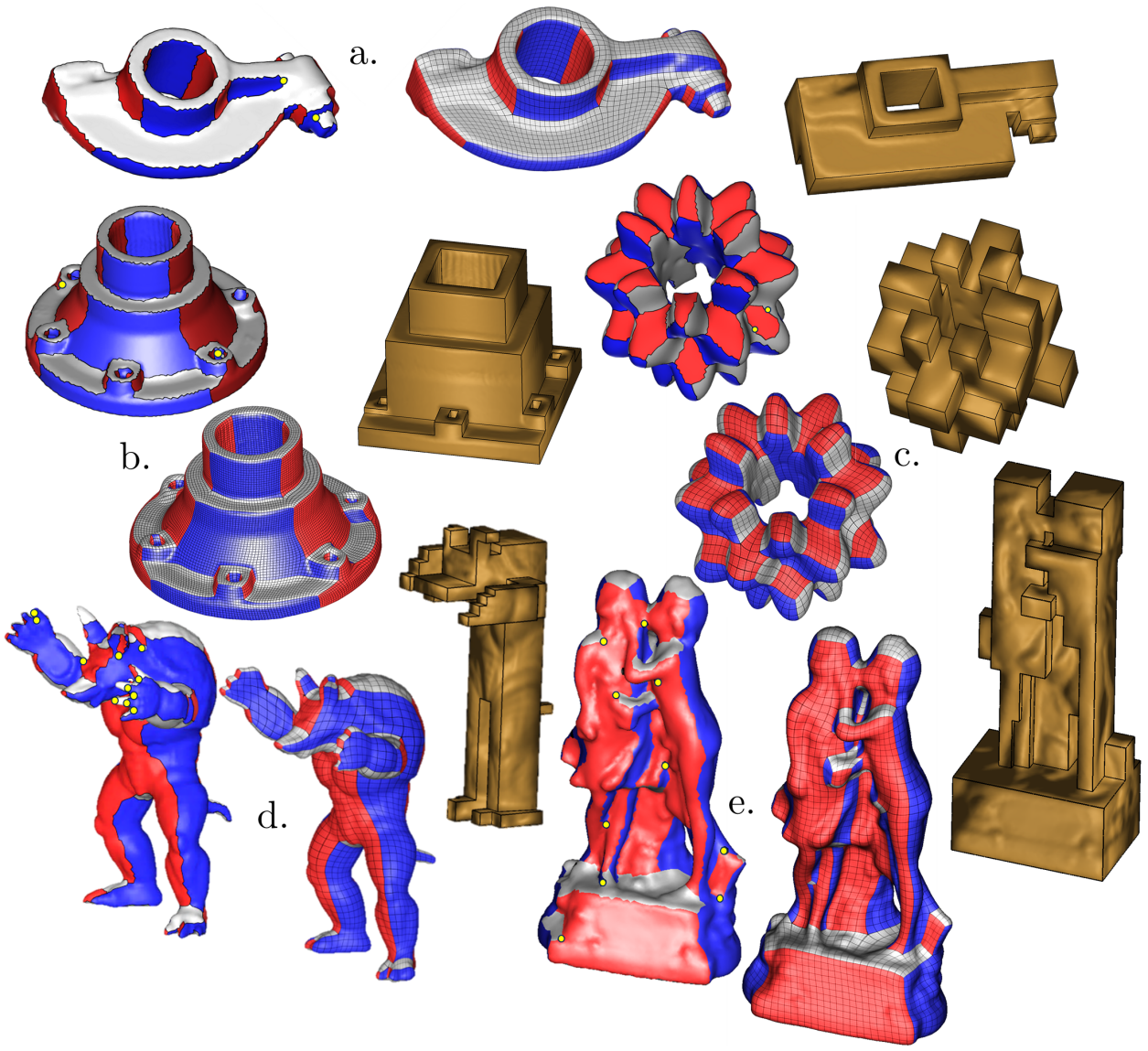


Figure 11: *PolyCuts of a variety of natural and CAD shapes. We show for each model the raw graph-cut segmentation with turning points on non-monotone boundaries highlighted in yellow, the final segmentation and parameterization and the PolyCubes themselves. Bunny and Armadillo models provided by Stanford Model Repository. CAD models provided courtesy of INRIA by the AIM@SHAPE Shape Repository. Bumpy Torus provided courtesy of Max-Planck Institut für Informatik by the AIM@SHAPE Shape Repository.*

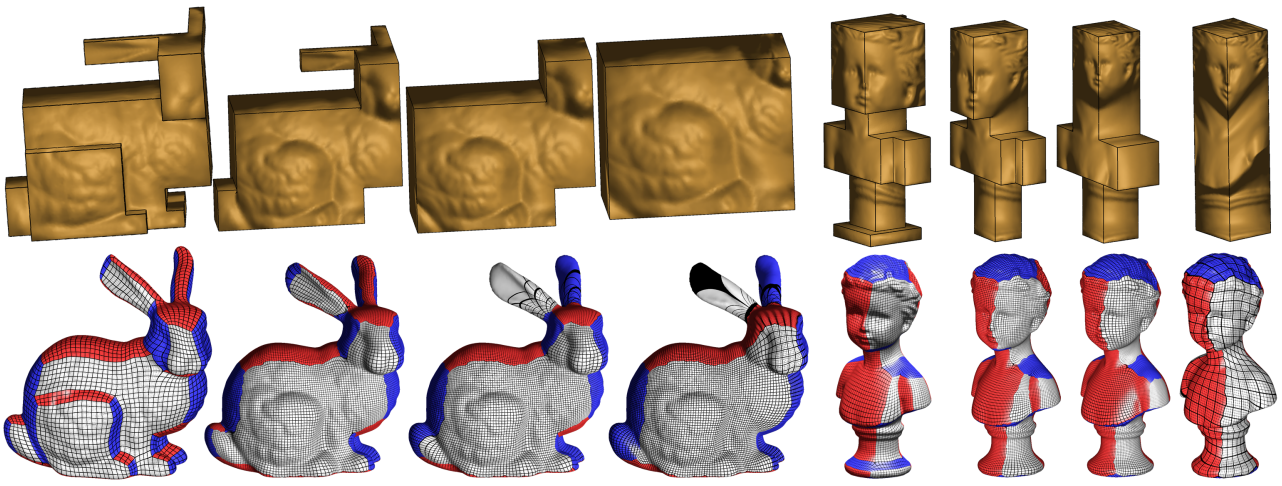


Figure 12: *Sequence of PolyCubes and corresponding parameterizations for the bunny and girl models generated by changing the compactness factor. (Bunny model courtesy of the Stanford Model Repository; girl provided courtesy of INRIA by the AIM@SHAPE Shape Repository.)*

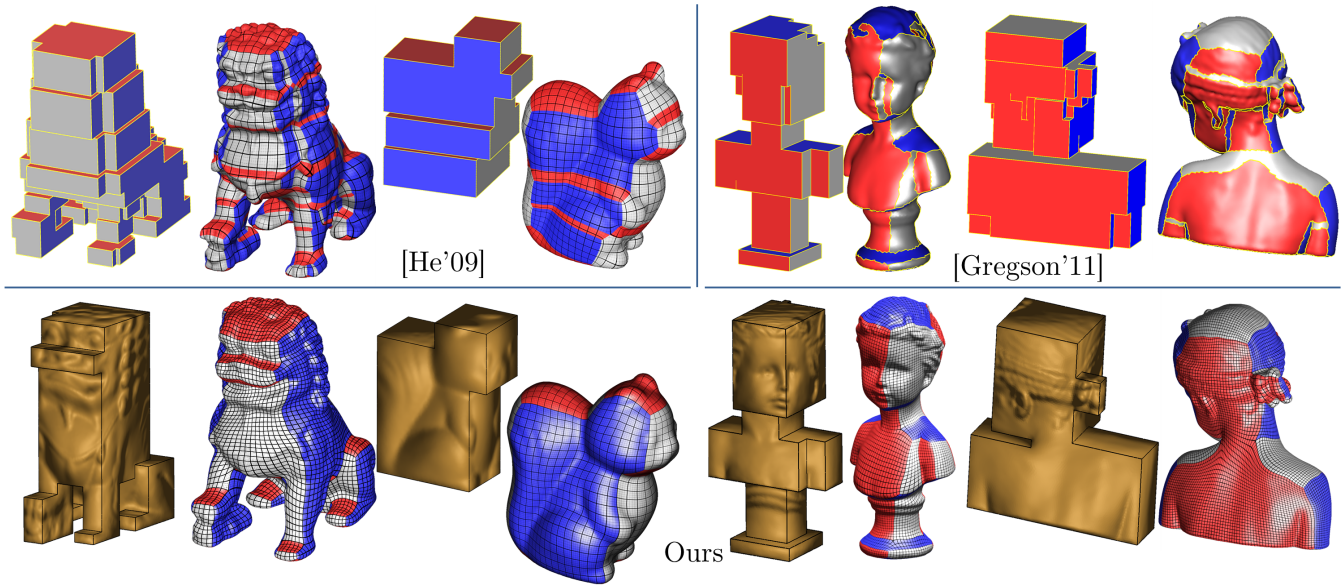


Figure 13: *Comparisons with [He et al. 2009] (left) and [Gregson et al. 2011] (right). Our results shown on the bottom. Note the reduction in the chart/singularity counts. The segmentations of Gregson et al. contains non-monotone boundaries (see girl's forehead). Squirrel provided courtesy of the Max-Planck Institut für Informatik by the AIM@SHAPE Shape Repository. All other models provided courtesy of INRIA by the AIM@SHAPE Shape Repository.*

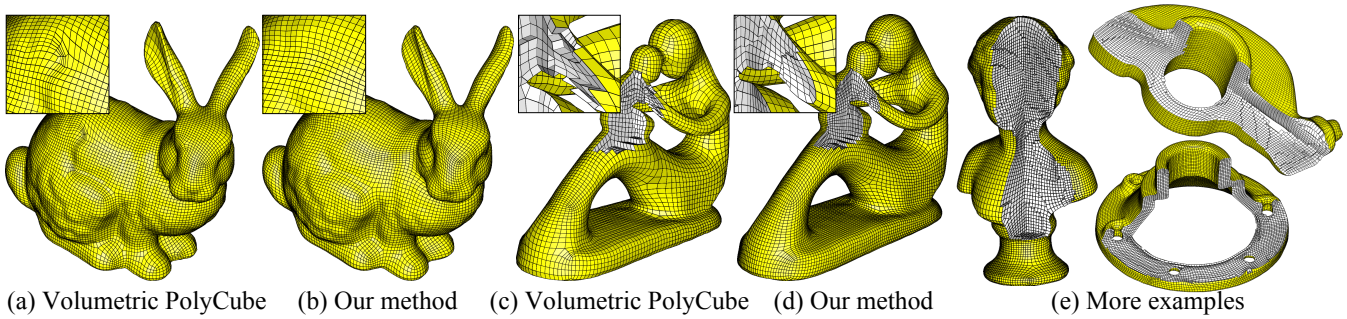


Figure 14: *Hex meshes computed using our method, including a comparison to [Gregson et al. 2011], highlighting the advantage of using better PolyCubes. Quality statistics shown in Table 3. (Bunny model courtesy of the Stanford Model Repository. Fertility model provided courtesy of Frank ter Haar by the AIM@SHAPE Shape Repository. All other models provided courtesy of INRIA by the AIM@SHAPE Shape Repository.)*

EPPSTEIN, D., AND MUMFORD, E. 2010. Steinitz theorems for orthogonal polyhedra. In *Proc. Symposium on Computational Geometry*, 429–438.

GREGSON, J., SHEFFER, A., AND ZHANG, E. 2011. All-hex mesh generation via volumetric polycube deformation. *Computer Graphics Forum (Proc. SGP) 30*, 5.

GU, X., GORTLER, S. J., AND HOPPE, H. 2002. Geometry images. *ACM Trans. Graph. 21*, 3 (July), 355–361.

HAN, S., XIA, J., AND HE, Y. 2010. Hexahedral shell mesh construction via volumetric polycube map. In *Proc. Symposium on Solid and Physical Modeling*, 127–136.

HE, Y., WANG, H., FU, C.-W., AND QIN, H. 2009. A divide-and-conquer approach for automatic polycube map construction. *Computers and Graphics 33*, 3, 369–380.

HOOS, H., AND STTZLE, T. 2004. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

HORMANN, K., AND FLOATER, M. S. 2006. Mean value coordinates for arbitrary planar polygons. *ACM Transactions on Graphics 25*, 4 (Oct.), 1424–1441.

JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. SIGGRAPH, 561–566.

KOLMOGOROV, V., AND ZABIH, R. 2004. What energy functions can be minimized via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 65–81.

LÉVY, B., AND LIU, Y. 2010. Lp centroidal voronoi tesselation and its applications. *ACM Transactions on Graphics (Proc. SIGGRAPH)*.

LI, B., AND QIN, H. 2012. Component-aware tensor-product trivariate splines of arbitrary topology. *Computers & Graphics*.

LI, B., LI, X., WANG, K., AND QIN, H. 2010. Generalized polycube trivariate splines. In *Proc. Shape Modeling International*, 261 –265.

LI, X., XU, H., WAN, S., YIN, Z., AND YU, W. 2010. Feature-aligned harmonic volumetric mapping using mfs. *Computers & Graphics 34*, 3, 242–251.

LI, B., LI, X., WANG, K., AND QIN, H. 2012. Surface mesh to volumetric spline conversion with generalized polycubes. *IEEE Transactions on Visualization and Computer Graphics 99*.

LIN, J., JIN, X., FAN, Z., AND WANG, C. C. L. 2008. Automatic polycube-maps. In *Proc. Advances in geometric modeling and processing*, 3–16.

PRAUN, E., AND HOPPE, H. 2003. Spherical parametrization and remeshing. *ACM Trans. Graph. 22*, 3 (July), 340–349.

SHAMIR, A. 2008. A survey on mesh segmentation techniques. *Comput. Graph. Forum 27*, 1539–1556.

SHEFFER, A., PRAUN, E., AND ROSE, K. 2006. Mesh parameterization methods and their applications. *Foundations and Trends in Computer Graphics and Vision 2*, 2, 105–171.

TARINI, M., HORMANN, K., CIGNONI, P., AND MONTANI, C. 2004. PolyCube-Maps. *ACM Transactions on Graphics 23*, 3 (Aug.), 853–860. Proc. of ACM SIGGRAPH 2004.

WAN, S., YIN, Z., ZHANG, K., ZHANG, H., AND LI, X. 2011. A topology-preserving optimization algorithm for polycube mapping. *Computers & Graphics 35*, 3, 639–649.

WANG, H., HE, Y., LI, X., GU, X., AND QIN, H. 2007. Polycube splines. In *Proc. 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 *Proc. Symposium on Solid and physical modeling*, 397–404.

XIA, J., HE, Y., YIN, X., HAN, S., AND GU, X. 2010. Direct-product volumetric parameterization of handlebodies via harmonic fields. In *Proc. Shape Modeling International*, IEEE, 3–12.

XIA, J., GARCIA, I., HE, Y., XIN, S.-Q., AND PATOW, G. 2011. Editable polycube map for gpu-based subdivision surfaces. In *Proc. Symposium on Interactive 3D Graphics and Games*, 151–158.

XU, Y., CHEN, R., GOTSMAN, C., AND LIU, L. 2011. Embedding a triangular graph within a given boundary. *Computer Aided Geometric Design 28*, 6, 349–356.

YAO, C., AND LEE, T. 2008. Adaptive geometry image. *IEEE Transactions on Visualization and Computer Graphics 14*, 4, 948–960.