Learning 3D Mesh Segmentation and Labeling

Evangelos Kalogerakis   Aaron Hertzmann   Karan Singh

University of Toronto

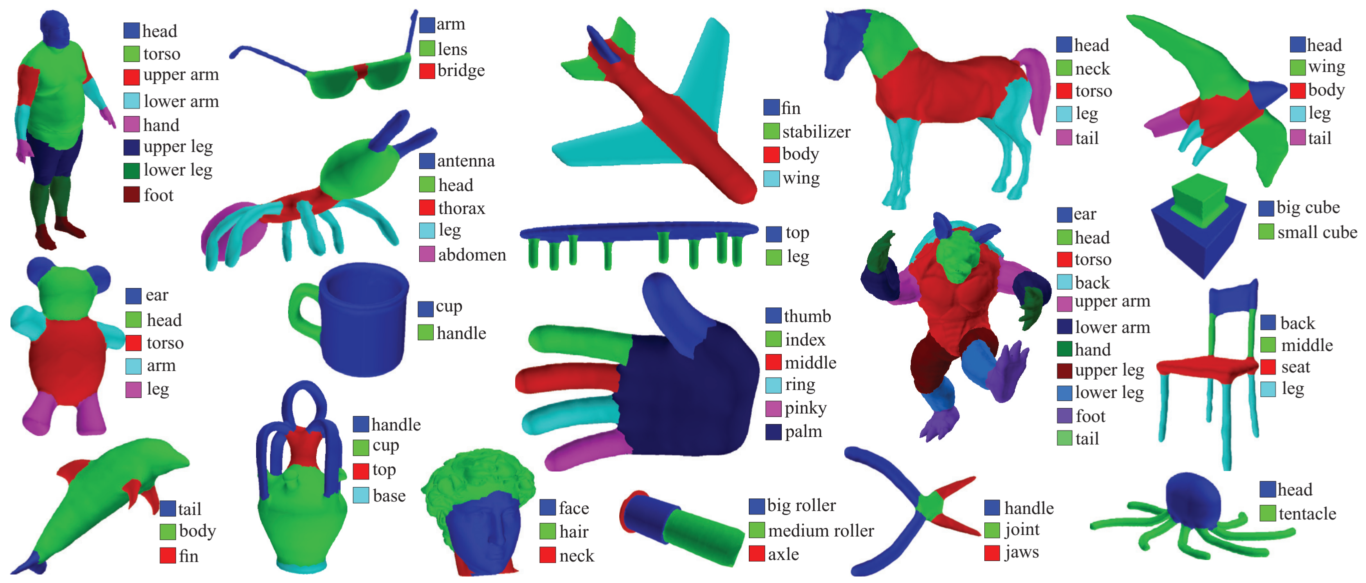

Figure 1: *Labeling and segmentation results from applying our algorithm to one mesh each from every category in the Princeton Segmentation Benchmark [Chen et al. 2009]. For each result, the algorithm was trained on the other meshes in the same class, e.g., the human was labeled after training on the other meshes in the human class.*

Abstract

This paper presents a data-driven approach to simultaneous segmentation and labeling of parts in 3D meshes. An objective function is formulated as a Conditional Random Field model, with terms assessing the consistency of faces with labels, and terms between labels of neighboring faces. The objective function is learned from a collection of labeled training meshes. The algorithm uses hundreds of geometric and contextual label features and learns different types of segmentations for different tasks, without requiring manual parameter tuning. Our algorithm achieves a significant improvement in results over the state-of-the-art when evaluated on the Princeton Segmentation Benchmark, often producing segmentations and labelings comparable to those produced by humans.

1 Introduction

Segmentation and labeling of 3D shapes into meaningful parts is fundamental to shape understanding and processing. Numerous tasks in geometric modeling, manufacturing, animation and texturing of 3D meshes rely on their segmentation into parts. Many of these problems further require labeled segmentations, where the parts are also recognized as instances of known part types. For most of these applications, the segmentation and labeling of the input shape is manually specified. For example, to synthesize texture for a humanoid mesh, one must identify which parts should have "arm" texture, which should have "leg" texture, and so on. Even tasks such as 3D shape matching or retrieval, which do not directly require labeled-segmentations, could benefit from knowledge of constituent parts and labels. However, there has been very little research in part labeling for 3D meshes, and 3D object segmentation likewise remains an open research problem [Chen et al. 2009].

This paper introduces a data-driven approach to simultaneous segmentation and labeling of parts in 3D meshes. Labeling of mesh parts is expressed as a problem of optimizing a Conditional Random Field (CRF) [Lafferty et al. 2001]. This segments a mesh into parts, with each part having a corresponding label. The CRF objective function includes unary terms that assess the consistency of faces with labels, and pairwise terms between labels of adjacent faces. The objective function is learned from a collection of labeled training meshes. The basic terms of the CRF are learned using JointBoost classifiers [Torralba et al. 2007], which automatically select from among hundreds of possible geometric features to choose those that are relevant for a particular segmentation task. Holdout validation is used to learn additional CRF parameters. We evaluate our methods on the Princeton Segmentation Benchmark, with manually-added labels. Our method yields 94% labeling accuracy, and is the first labeling method applicable to such a broad range of meshes. In segmentation, our method yields 9.5% Rand In-

http://www.dgp.toronto.edu/~kalo/papers/LabelMeshes/

dex error, significantly better than the current state-of-the-art, with results similar to human-provided segmentations for most classes. No manual parameter tuning is required. The main limitation of our approach is that it requires a consistently-labeled training set; however, we find that, for many cases, just a few training meshes suffice to obtain high-quality results. Different segmentation tasks can be specified by providing examples of the new task, without requiring any manual parameter adjustments. Once learned, the algorithm can be applied to databases of the same type of objects to automatically segment and label them.

To date, nearly all existing mesh segmentation methods attempt segmentation without recognition. When the goal of segmentation can be formulated mathematically (e.g., partitioning into developable patches), low-level geometric cues may be sufficient. However, many tasks require some understanding of the functions or relationships of parts, which are not readily available from low-level geometric cues. It is unknown whether human-level 3D mesh segmentation is possible without the benefit of higher-level cues. It is worth noting that, in computer vision, after decades of research on performing image segmentation alone, most work has turned to the joint segmentation and recognition of images. Furthermore, current models are learned from training data, allowing them to employ much more sophisticated models than are possible with manually-tuned models. These methods produce state-of-the-art results on several benchmark tests. Hence, it is worth asking: is part recognition useful for 3D mesh segmentation? Furthermore, can segmentation algorithms benefit from models learned from human-labeled meshes? Our work provides positive evidence for both questions.

2 Related work

Mesh segmentation has been a very active area of research in computer graphics. Most effort has focused on finding simple geometric criteria for segmentation of a single input mesh [Mangan and Whitaker 1999; Shlafman et al. 2002; Katz and Tal 2003; Liu and Zhang 2004; Katz et al. 2005; Simari et al. 2006; Attene et al. 2006b; Lin et al. 2007; Golovinskiy and Funkhouser 2008; Li et al. 2008; Lai et al. 2008; Lavoué and Wolf 2008; Huang et al. 2009]; see [Attene et al. 2006a; Shamir 2008; Chen et al. 2009] for surveys. Such approaches employ simple, interpretable geometric algorithms, but are limited to a single generic rule (e.g., concavity, skeleton topology, fitting shape primitives) or a single feature (e.g., shape diameter, curvature tensor, geodesic distances) to partition an input mesh. Our method employs many of the geometric features proposed by these methods. For many problems, different types of surfaces and different surface parts may require different features for segmentation. Because our model is learned, it can employ many different geometric features to partition the input mesh. Our algorithm learns problem-specific parameters from training examples, rather than requiring manually-tuned parameters. Furthermore, our method jointly segments and labels meshes. Simari *et al.* [2009] perform segmentation and labeling jointly. However, this method requires manual definition and tuning of objective functions for each type of part, and is sensitive to local minima.

A few approaches make use of part matching for segmentation, and can transfer part labels based on the matches. Kraevoy *et al.* [2007] and Shapira *et al.* [In Press] perform an initial segmentation, and then match segments and transfer labels based on this segmentation. These methods require the initial segmentation to be sufficiently reliable. Pekelny and Gotsman [2008] track and label rigid components in sequences of 3D range data through the Iterated Closest Point registration algorithm, given an initial user segmentation. Similarly, Golovinskiy and Funkhouser [2009] simultaneously partition collections of 3D models by matching points between meshes based on rigid mesh alignment. A user may provide example segmentations to be included in the matching. These methods are limited to cases where an accurate rigid correspondence exists.

Joint image segmentation and recognition has recently been an active topic in computer vision research. Early works in this area include [Duygulu et al. 2002; He et al. 2004; Konishi and Yuille 2000; Kumar and Hebert 2003; Tu et al. 2005; Schnitman et al. 2006]. Our method is most directly inspired by TextonBoost [Shotton et al. 2009], which performs joint image segmentation and recognition, using a model learned from a training database. As in Textonboost, we also make use of JointBoost and Conditional Random Fields. We add new components to the model, including 3D geometric feature vectors, 3D contextual features, cascades of classifiers, and a learned pairwise classifier term, all of which we find to be essential to obtaining good results.

Our work is also related to segmentation and recognition of 3D range data [Anguelov et al. 2005; Lim and Suter 2007; Munoz et al. 2008]. These methods employ small sets of features, such as local point density or height from ground, which are specialized to discriminate a few object categories in outdoor scenes, or to separate foreground from background. Golovinskiy et al. [2009] segment urban range data using a graph cut method, and then apply a learned classifier, based on geometric and contextual shape cues. Range data methods aim to identify large-scale structures from point clouds, such as separating cars from roads, whereas we aim to distinguish smaller parts in 3D meshes. Hence, unlike these methods, we employ a large variety of shape-based mesh features along with appropriate contextual features, and also use sophisticated classifiers for the unary and pairwise terms.

3 CRF model for segmentation and labeling

We now describe our algorithm for segmenting and recognizing parts of a mesh; the procedure for learning this model is described in Section 4. Our goal is to label each mesh face i with a label $l \in \mathcal{C}$, where $\mathcal{C}$ is a predefined set of possible labels, such as "arm," "leg," or "torso." Each face has a vector of *unary features* $\mathbf{x}_i$, which includes descriptors of local surface geometry and context, such as curvatures, shape diameter, and shape context. These features provide cues for face labeling. In addition, for each adjacent pair of faces, we define a vector of *pairwise features* $\mathbf{y}_{ij}$, such as dihedral angles, which provide cues to whether adjacent faces should have the same label. Then, computing all mesh labels involves minimizing the following objective function:

$$E(\mathbf{c};\theta) = \sum_i a_i E_1(c_i;\mathbf{x}_i,\theta_1) + \sum_{i,j} \ell_{ij} E_2(c_i,c_j;\mathbf{y}_{ij},\theta_2) \quad (1)$$

where the unary term E_1 measures consistency between the features $\mathbf{x}_i$ of mesh face i and its label c_i, the pairwise term E_2 measures consistency between adjacent face labels c_i and c_j, given pairwise features $\mathbf{y}_{ij}$. The model parameters are $\theta = \{\theta_1, \theta_2\}$. The terms are weighted by the area a_i of face i, and the length of the edge ℓ_{ij} between faces i and j. In order to make energies comparable across meshes, the areas a_i are normalized by the median face area in the mesh, and the edge lengths ℓ_{ij} are normalized by the median edge length. Details of the energy terms and feature vectors are given later in this section.

This type of model is referred to as a Conditional Random Field (CRF) [Lafferty et al. 2001]. In a CRF, the conditional probability of a labeling given the mesh is defined as:

$$P(\mathbf{c}|\mathbf{x},\mathbf{y},\theta) = \exp(-E(\mathbf{c};\theta))/Z(\mathbf{x},\mathbf{y},\theta) \quad (2)$$

where Z is a normalization factor. This is in contrast to a Markov Random Field (MRF) model [Geman and Geman 1984], which defines a joint probability over the mesh and the labels, from which

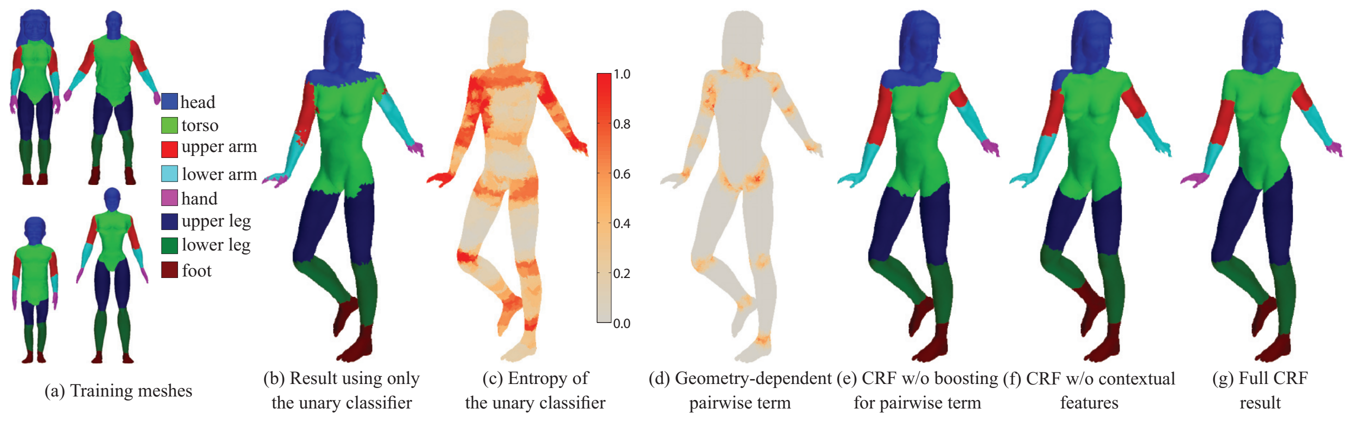
head
torso
upper arm
lower arm
hand
upper leg
lower leg
foot
1.0
0.8
0.6
0.4
0.2
0.0
(a) Training meshes
(b) Result using only the unary classifier
(c) Entropy of the unary classifier
(d) Geometry-dependent pairwise term
(e) CRF w/o boosting for pairwise term
(f) CRF w/o contextual features
(g) Full CRF result

Figure 2: *Components of our algorithm.* ***(a)*** *The entire training set for this example consists of four meshes. The bottom-right mesh is used as the validation set.* ***(b)*** *Labeling result using only the unary classifier.* ***(c)*** *Visualization of classifier uncertainty, computed as the entropy of the probabilities output by the unary classifier. Red values indicate greater uncertainty. The classifier is uncertain mainly near object boundaries, and where corresponding parts in the training meshes have inconsistent boundaries.* ***(d)*** *Geometry-dependent pairwise term (exponentiated and normalized). This term prefers boundaries to occur at specific locations.* ***(e)*** *Result of applying a CRF model without JointBoost for the pairwise term.* ***(f)*** *CRF result, but omitting contextual label features.* ***(g)*** *Result of applying our complete CRF model. Note the accuracy of the result, despite the mesh having different pose and body shape from the training meshes.*

the conditional may then be derived. For segmentation and labeling, CRFs have two advantages over MRFs. First, the pairwise term E_2 in a CRF can depend on the input data, which is not true in an MRF. This allows us, for example, to express that segment boundaries are more likely to occur between a pair of faces with a small exterior dihedral angle. Second, CRF learning algorithms optimize for labeling performance, whereas MRF learning algorithms attempt to model both the input features and the labels, and thus may have worse labeling performance. For these reasons, CRFs have become popular in natural-language parsing (e.g., [Lafferty et al. 2001]) and image segmentation (e.g., [He et al. 2004; Shotton et al. 2009]).

The objective $E(\mathbf{c};\theta)$ is optimized using alpha-expansion graph-cuts [Boykov et al. 2001]. The resulting labeling $\mathbf{c}$ implicitly defines a segmentation of the mesh, with segment boundaries lying between each pair of faces with differing labels. Note that this means that our method cannot separate adjacent parts that share the same label. Furthermore, our method is only suitable for learning segmentations that have attached labels. However, we do not require the number of segments to be specified in advance.

3.1 Unary Energy Term

The unary energy term evaluates a classifier. The classifier takes the feature vector $\mathbf{x}$ for a face as input, and returns a probability distribution of labels for that face: $P(c|\mathbf{x},\theta_1)$. Specifically, we use a JointBoost classifier [Shotton et al. 2009; Torralba et al. 2007], summarized in Section 3.3. Then, the unary energy of a label c is equal to its negative log-probability:

$$E_1(c;\mathbf{x},\theta_1) = -\log P(c|\mathbf{x},\theta_1) \tag{3}$$

The unary classifier is the most important component of our system. As illustrated in Figure 2(b), labeling using just this term alone gives good results in part interiors, but not near boundaries. This is accurately reflected by the uncertainty of the classifier (Figure 2(c)). Next, we add a pairwise term to refine these boundaries.

3.2 Pairwise Energy Term

The pairwise energy term penalizes neighboring faces being assigned different labels:

$$E_2(c,c';\mathbf{y},\theta_2) = L(c,c')\,G(\mathbf{y}) \tag{4}$$

This term consists of a label-compatibility term L, weighted by a geometry-dependent term G. The main role of the pairwise term is to improve boundaries between segments and to prevent incompatible segments from being adjacent. The pairwise energy term is always zero when c and c' have the same label. Hence, the pairwise term cannot be used on its own, since it assigns zero energy when all faces have the same label. The geometry-dependent term is visualized in Figure 2(d).

The label-compatibility term $L(c,c')$ measures the consistency between two adjacent labels. This term is represented as a matrix of penalties for each possible pair of labels, which allows different pairs of labels to incur different penalties. For example, head-ear boundary edges may need to be penalized less than head-torso boundary edges (since ears might be much smaller parts and less common in the training examples) while head-foot boundaries might never occur. The costs are non-negative $(0 \leq L(k,l))$ and symmetric $(L(k,l) = L(l,k))$, for labels $k,l \in \mathcal{C}$. Furthermore, we constrain there to be no penalty when there is no discontinuity: $L(k,k) = 0$ for all k.

The geometry-dependent term $G(\mathbf{y})$ measures the likelihood of there being a difference in labels, as a function of the geometry alone. This term has the following form:

$$\begin{aligned} G(\mathbf{y}) = &-\kappa \log P(c \neq c'|\mathbf{y},\xi) \\ &-\lambda \log\left(1 - \min(\omega/\pi, 1) + \epsilon\right) + \mu \end{aligned} \tag{5}$$

The first term is the output of a JointBoost classifier that computes $P(c \neq c'|\mathbf{y},\xi)$, the probability of two adjacent faces having distinct labels, as a function of pairwise geometric features $\mathbf{y}$. This classifier helps detect boundaries better than using only dihedral angles (Figure 2e). The second term penalizes boundaries between faces with high exterior dihedral angle ω, following Shapira et al. [In Press]. The μ term penalizes boundary length and is helpful for preventing jaggy boundaries and for removing small, isolated segments [Golovinskiy and Funkhouser 2008; Shapira et al. In Press]. A small constant ϵ is added to avoid computing $\log 0$.

3.3 JointBoost classifier

Here we briefly summarize the JointBoost classifier; see [Torralba et al. 2007] for more information. JointBoost is a boosting algorithm that has many appealing properties: it performs automatic feature selection and can handle large numbers of input features for multiclass classification, it has a fast sequential learning algorithm (Section 4.1), and it produces output probabilities suitable for combination with other terms in the CRF model. JointBoost is designed to share features among classes, which greatly reduces generalization error for multiclass recognition when classes overlap in feature space. For these reasons, we believe that JointBoost is the best available classifier for this task.

The classifier takes as input a feature vector $\mathbf{z}$, and outputs a probability $P(c = l|\mathbf{z})$ for each possible class label $l \in \mathcal{C}$, where $\mathcal{C}$ is the set of possible labels. In the unary energy term, a classifier computes the likelihood of a part label c given unary features $\mathbf{x}$. In the pairwise energy term, a second JointBoost classifier is used to determine the likelihood that adjacent faces have different classes ($c \neq c'$) given pairwise features $\mathbf{y}$. This is a binary classifier; in this case, JointBoost reduces to an earlier algorithm called GentleBoost [Friedman et al. 2000]. Finally, we use a cascade of JointBoost classifiers to define contextual label features (Section 3.4).

The classifier is composed of *decision stumps*. A decision stump is a very simple classifier that scores each possible class label l, given a feature vector $\mathbf{z}$, based only on thresholding its f-th entry z_f. A JointBoost decision stump can be written as:

$$h(\mathbf{z}, l; \phi) = \begin{cases} a & z_f > \tau \text{ and } l \in \mathcal{C}_S \\ b & z_f \leq \tau \text{ and } l \in \mathcal{C}_S \\ k_l & l \notin \mathcal{C}_S \end{cases} \tag{6}$$

In other words, each decision stump stores a set of classes $\mathcal{C}_S$. If $l \in \mathcal{C}_S$, then the stump compares z_f against a threshold τ, and returns a constant a if $z_f > \tau$, and another constant b otherwise. If $l \notin \mathcal{C}_S$, then the comparison is ignored; instead, a constant k_l is returned instead. There is one k_l for each $l \notin \mathcal{C}_S$. The parameters ϕ of a single decision stump are f, a, b, τ, the set $\mathcal{C}_S$, and k_l for each $l \notin \mathcal{C}_S$.

The probability of a given class l is then computed by summing the decision stumps and then performing a softmax transformation:

$$H(\mathbf{z}, l) = \sum_j h(\mathbf{z}, l; \phi_j) \tag{7}$$

$$P(c = l|\mathbf{z}, \xi) = \frac{\exp(H(\mathbf{z}, l))}{\sum_{\ell \in \mathcal{C}} \exp(H(\mathbf{z}, \ell))} \tag{8}$$

The parameters ξ consist of the parameters $\{\phi_j\}$ of all the individual decision stumps.

3.4 Feature vectors

We do not know in advance which features will be useful for segmentation. Furthermore, it may be that different features are informative for different mesh parts and for different styles of segmentation. As a result, we construct our feature vectors out of as many informative features as possible. Since the JointBoost algorithm performs automatic feature selection, each classifier only uses a subset of the provided features. In our experiments, we have not found a case where adding informative features led to worse results. Hence, one may add other features besides the ones listed here. We find that the precise form of the features is important: careful selection of details, such as binning strategy and normalization, can improve results. Adding features does increase computation time, especially for preprocessing and learning. Hence, we have attempted to design features that are as informative as possible.

Unary features. We use multi-scale surface curvature, singular values extracted from Principal Component Analysis of local shape, shape diameter [Shapira et al. In Press], distances from medial surface points [Liu et al. 2009], average geodesic distances [Hilaga et al. 2001; Zhang et al. 2005], shape contexts [Belongie et al. 2002], and spin images [Johnson and Hebert 1999] to form a basic 375-dimensional feature vector $\tilde{\mathbf{x}}_i$ per face i. Full details of our implementation for these features are given in Appendix A.

Contextual label features. Training a classifier using only the above features is often sufficient for labeling. However, in many cases, better results can be achieved by re-training an additional classifier that uses information about the global distribution of labels around each mesh face. Since the labels are not known in advance, they are approximated by an initial application of the classifier with the above features. We introduce *contextual label features* based on these initial labels.

We first train an initial JointBoost classifier using the initial feature vector $\tilde{\mathbf{x}}$. This classifier can be applied to each training mesh to produce per-face class probabilities $P(\mathbf{c}|\tilde{\mathbf{x}})$. Then, for each face i, we compute a histogram of these probabilities, which captures the global distribution of part labels relative to the face, in a manner inspired by shape contexts [Belongie et al. 2002], and similar to image auto-contexts [Tu 2008] and bags of semantic textons [Shotton et al. 2008]. The histogram bins are determined as a function of geodesic and euclidean distances. These features allow the algorithm to make use of estimates of labels from the global context of each face. Details of the histograms are given in the Appendix.

The values of these histogram bins form a set $\bar{\mathbf{x}}_1$ of contextual label features that are concatenated with $\tilde{\mathbf{x}}$ to produce the full feature vector $\mathbf{x}_1 = [\tilde{\mathbf{x}}^T, \bar{\mathbf{x}}_1^T]^T$. The new feature vector has $375 + 35 \cdot |\mathcal{C}|$ features, where $|\mathcal{C}|$ is the number of labels. Then, we train a new JointBoost classifier from $\bar{\mathbf{x}}_1$ to class probabilities. The new classifier will now take into account the generated contextual features to further discriminate parts, as shown in Figure 2g, compared to the result of Figure 2f, where only the initial JointBoost classifier is used (without the contextual label features).

After training the second classifier, we bin the newly produced class probabilities $P(\mathbf{c}|\mathbf{x}_1)$ to produce new contextual label features $\bar{\mathbf{x}}_2$. These are concatenated with $\tilde{\mathbf{x}}$ to produce a third feature vector $\mathbf{x}_2$. Then, we can train a third classifier based on the feature vector $\mathbf{x}_2$. This process can be iterated to further refine the discrimination of classes, similar to cascade generalization [Gama and Brazdil 2000]. This approach may be iterated N times to produce N feature vectors, until the error on the validation set does not increase. In our experiments, the algorithm usually selects $N = 3$. Computing the feature vectors for a new surface entails repeating the same process as above: $\tilde{\mathbf{x}}$ is computed for each face, then the first JointBoost produces the first set of contextual features $\bar{\mathbf{x}}$, and the process repeats until getting $\mathbf{x}_N$, which is used as the complete feature vector $\mathbf{x}$. We find that using these contextual features produces a significant improvement in performance, about $3 - 10\%$, depending on the mesh category.

Pairwise features. The pairwise feature vector $\mathbf{y}_{ij}$ between faces i and j consists of the dihedral angles between the faces, and differences of the following features between the faces: curvatures and surface third-derivatives, shape diameter differences, and differences of distances from medial surface points. We also use contextual label features, similar to the features above; however, we found in our experiments that these contextual features have little impact on the results (about 0.5% improvement). The complete feature vector $\mathbf{y}$ is 191-dimensional. Details of the pairwise features are given in Appendix B.

4 Learning CRF parameters

We now describe a procedure for learning the parameters of the CRF model, given a set of labeled training meshes. The natural approach to CRF learning is Maximum Likelihood or MAP, e.g., maximizing Equation 2 over all training meshes. Unfortunately, computing the normalization Z is intractable. While contrastive divergence can be used for this optimization [He et al. 2004], this method is computationally expensive, and would not be feasible at the scale of mesh processing.

Instead, we perform the following steps, based on the approach of Shotton et al. [2009]. First, we randomly split the training meshes into an *exemplar set* and a *validation set*, in a proportion of approximately $4:1$. We then learn the JointBoost classifiers for the unary term and the pairwise term from the exemplar set. Finally, the remaining CRF parameters are learned by iteratively optimizing segmentation performance on the validation set. These steps are described below.

4.1 Learning JointBoost classifiers

For completeness, we now summarize the JointBoost learning algorithm, for learning classifiers of the form described in Section 3.3. See [Torralba et al. 2007] for an excellent explanation and derivation of the algorithm.

The input to the algorithm is a collection of M training pairs $(\mathbf{z}_i, c_i)$, where $\mathbf{z}_i$ is a feature vector and c_i is the corresponding class label for that feature. Furthermore, each training pair is assigned a per-class weight $w_{i,c}$. JointBoost minimizes the weighted multiclass exponential loss over the exemplar set:

$$J = \sum_{i=1}^{M} \sum_{l \in \mathcal{C}} w_{i,c} \exp\left(-I(c_i, l)\, H(\mathbf{z}_i, l)\right) \tag{9}$$

where $H(\mathbf{z}, l)$ is defined in Equation 7, $\mathcal{C}$ is the set of possible class labels, and $I(c, c')$ is an indicator function that is 1 when $c = c'$ and -1 otherwise.

For the unary terms, the training pairs are the per-face feature vectors and their labels $(\mathbf{x}_i, c_i)$ for all mesh faces in the exemplar set. For the pairwise terms, the training pairs are the pairwise feature vectors and their binary labels $(\mathbf{y}_{ij}, c_i \neq c_j)$. For the unary term, the $w_{i,c}$ is the area of face i. For the pairwise term, $w_{i,c}$ is used to reweight the boundary edges, since the training data contains many more non-boundary edges. Let N_B and N_{NB} be the number of each type of edge, then $w_{i,c} = \ell N_B$ for non-boundary edges and $w_{i,c} = \ell N_{NB}$ for boundary edges, where ℓ is the corresponding edge length.

The algorithm proceeds iteratively. The algorithm stores a set of weights $\tilde{w}_{i,c}$ that are initialized to the weights $w_{i,c}$. Then, at each iteration, one decision stump (Equation 6) is added to the classifier. The parameters ϕ_j of the stump at iteration j are computed to optimize the following weighted least-squares objective:

$$J_{wse}(\phi_j) = \sum_{l \in \mathcal{C}} \sum_{i=1}^{M} \tilde{w}_{i,l} (I(c_i, l) - h(\mathbf{z}_i, l; \phi_j))^2 \tag{10}$$

where $\mathcal{C}$ are the possible class labels. Following Torralba et al. [2007], the optimal a, b, k_l are computed in closed-form, and $f, \tau, \mathcal{C}_S$ are computed by brute-force. When the number of labels $|\mathcal{C}|$ is greater than 6, the greedy heuristic search is used for $\mathcal{C}_S$. Once the parameters ϕ_j are determined, the weights are updated as:

$$\tilde{w}_{i,c} \leftarrow \tilde{w}_{i,c} \exp(-I(c_i, l)\, h(\mathbf{z}_i, l; \phi_j)) \tag{11}$$

and the algorithm continues with the next decision stump.

We run the algorithm for at most 300 iterations. To avoid overfitting, we also monitor the classifier's performance on the validation set by computing the cost function of Eq. 9 after each iteration, and keep track of which iteration j^* gave the best score. At the end of the process, we return the classifier from step j^* (i.e., discarding decision stumps from after step j^*). We also terminate early if the classifier's performance in the validation set has not improved over the last 50 iterations.

4.2 Learning the remaining parameters

Once the JointBoost classifiers have been learned, we learn the remaining parameters of the pairwise term $(\kappa, \lambda, \mu, L)$ by hold-out validation. Specifically, for any particular setting of these parameters, we can apply the CRF to all of the validation meshes, and evaluate the classification results. We seek the values of these parameters that give the best score on the validation meshes.

We need to define an error function by which to evaluate classification results. A obvious choice would be to measure what percentage of the mesh's surface area is correctly labeled. We refer to this as the Classification Error:

$$E = \left(\sum_i a_i (I(c_i, c_i^*) + 1)/2 \right) \Big/ \left(\sum_i a_i \right) \tag{12}$$

where a_i is the area of face i, c_i is the ground-truth label for face i, $c_i^* = \arg\max P(c|\mathbf{x}_i)$ is the output of the classifier for face i, and $I(c, c')$ defined as in Section 4.1. However, when training against this error, the algorithm tends to mostly refine boundaries between larger parts but skip cuts that generate small parts, producing noticeable errors in the results without incurring much penalty.

Instead, we optimize with respect to the Segment-Weighted Error which weighs each segment equally:

$$E_S = \sum_i \frac{a_i}{A_{c_i}} (I(c_i, c_i^*) + 1)/2 \tag{13}$$

where A_{c_i} is the total area of all faces within the segment that has ground-truth label c_i.

These parameters are optimized in two steps. First, the Segment-Weighted Error is minimized over a coarse grid in parameter space by brute-force search. Second, starting from the minimal point in the grid, optimization continues using MATLAB's implementation of Preconditioned Conjugate Gradient with numerically-estimated gradients.

5 Results

We now describe experimental validation and analysis of our approach.

Data set. We employed data from the Princeton Segmentation Benchmark [Chen et al. 2009] for all of our tests. The dataset provides 19 categories of meshes, segmentations provided by human

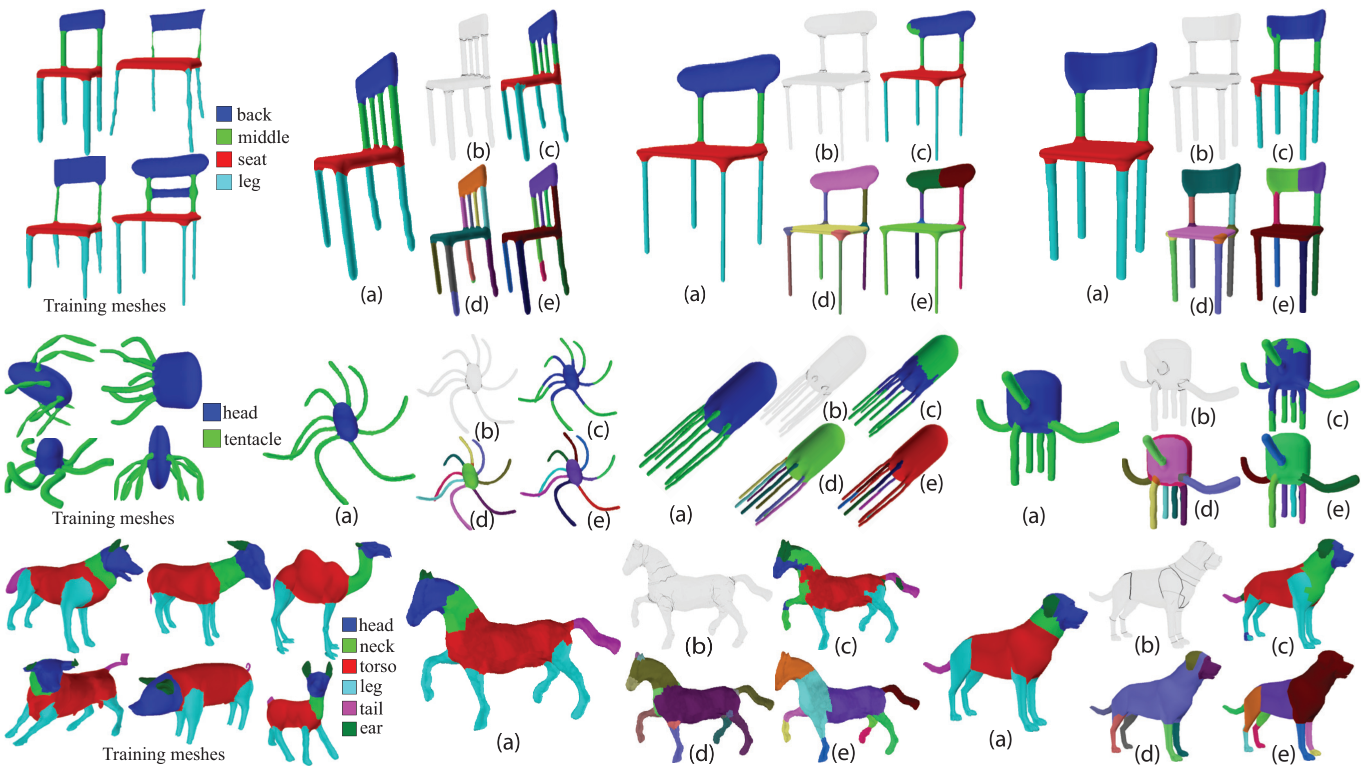

Figure 3: *Comparisons to previous segmentation methods, for chairs, octopuses, and quadrupeds. For each test, the entire training set is shown on the left. In each figure, the methods compared are:* ***(a)*** *our method,* ***(b)*** *average human segmentation from the Princeton Segmentation Benchmark,* ***(c)*** *Consistent Segmentation [Golovinskiy 2009],* ***(d)*** *Shape Diameter [Shapira In Press],* ***(e)*** *Randomized Cuts [Golovinskiy 2008], with number of segments defined as the average number of segments in the category. The Consistent Segmentation method provides labels in addition to segmentation based on the same training set. The other methods only perform segmentation, and do not make use of training data.*

users, source code for computing evaluation scores, and the results of applying many previous segmentation methods.

We performed a few initial steps to process the data. Since segment labels are not provided with the data, we manually assigned a set of labels to each class (Figure 1), according to the average human segmentation for each category. For example, almost all users partition the elements of the chair class (Fig. 3) into legs, seats, back, and middle.

For each mesh, the Princeton benchmark provides multiple segmentations. The dataset contains significant variations in types of segmentations: while many segmentations are consistent with each other, one user might segment a human into just 4 segments, whereas another might use 50 segments. We select one of these segmentations to be labeled and used as the training/test data for that mesh, in order to reduce the size of the dataset and remove outlier segmentations. For most meshes, the exemplar segmentation was selected as the segmentation with the lowest average Rand Index to all other segmentations for that mesh. However, in a few cases, the mesh with the best score had a very atypical segmentation to the rest of the category (e.g., the best segmentation for one octopus mesh had tentacles subdivided into many parts, whereas the tentacles in the rest of the category were not), in which case, we manually merged segments or chose the second-best segmentation.

Labeling results. We now evaluate the quality of the labels produced by our method. Because each category in the database has only 20 meshes, we evaluate our method using leave-one-out cross-validation. For each mesh i in each category, we train a CRF model on the other 19 meshes in that class, and then apply it to mesh i, and compute the Classification Error (Eq. 12) according to the ground-truth data. We report Recognition Rate, which is one minus Classification Error, reported as a percentage. Averaging over all categories, our method obtains approximately 94% accuracy.

In order to determine the effect of training set size, we repeated the experiment with smaller training sets. When testing on mesh i, the CRF is trained on a subset of M of the remaining 19 meshes. These are averaged over 5 randomly-selected subsets. We tested with $M = 3, 6, 12$. Table 1(left) shows scores of our method for different mesh categories and for different values of M. When reducing the training set size, we find that our method still gives excellent results for categories with little geometric variation (such as the Octopus), whereas other categories, such as Bust and Bearing, have very different geometric parts; a subset of 3 meshes often lacks some of the labels used elsewhere in the category.

The Segment-Weighted Error (Eq. 13) scores of our algorithm are: 89% (leave-one-out error), 85% ($M = 12$ training examples), 81% ($M = 6$ training examples) and drops to 75% ($M = 3$ training examples). The scores using this metric are lower than those of Classification Error, because tiny missing segments cause disproportionally large penalties.

To our knowledge, the only previous method that can label meshes by example is that of Golovinskiy and Funkhouser [2009]. This method assumes that rigid alignment can be performed between a mesh and the training data, which does not hold for most of the benchmark data; comparisons are shown in Figure 3. To our knowledge, our method is the first to be able to accurately label such a

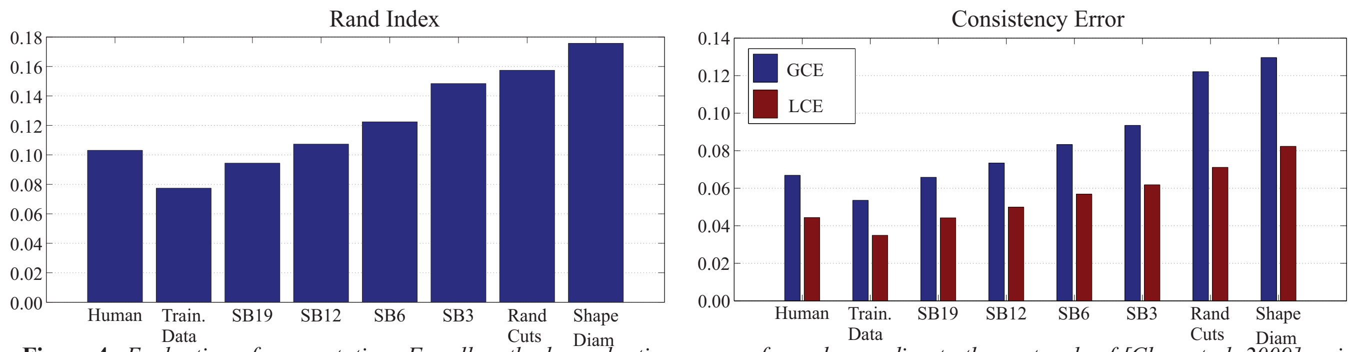

Figure 4: *Evaluation of segmentation. For all methods, evaluations are performed according to the protocols of [Chen et al. 2009], using all human segmentations in the Princeton Segmentation Benchmark. 'SB19' represents leave-one-out-error of our technique averaged over all the categories of the benchmark. 'SB12', 'SB6', 'SB3' represents the average error using training sets of size 12,7,6, and 3 (see text for details). SB19 performs almost* 50% *better than the best existing methods. Performance drops with less training data, but, even with only* 3 *examples, our method still out-performs previous methods by a small margin.*

	Labeling: Recognition Rate				**Segmentation: Rand Index**					
	SB19	*SB12*	*SB6*	*SB3*	*Bench.*	*Train.*	*SB19*	*SB12*	*SB6*	*SB3*
Human	93.6	93.2	89.4	83.2	13.5	11.2	11.9	12.9	14.3	14.7
Cup	99.6	99.6	99.1	96.3	13.6	9.8	9.9	9.9	10.0	10.0
Glasses	97.4	97.2	96.1	94.4	10.1	8.4	13.6	14.1	14.1	14.2
Airplane	96.3	96.1	95.5	91.2	9.2	7.4	7.9	8.2	8.0	10.2
Ant	98.8	98.8	98.7	97.4	3.0	1.7	1.9	2.2	2.3	2.6
Chair	98.5	98.4	97.8	97.1	8.9	5.2	5.4	5.6	6.1	6.6
Octopus	98.4	98.4	98.6	98.3	2.4	1.8	1.8	1.8	2.2	2.2
Table	99.4	99.3	99.1	99.0	9.3	5.9	6.2	6.6	6.4	11.1
Teddy	98.1	98.1	93.3	93.1	4.9	3.1	3.1	3.2	5.3	5.6
Hand	90.5	88.7	82.4	74.9	9.1	9.1	10.4	11.2	13.9	15.8
Plier	97.0	96.2	94.3	92.2	7.1	5.1	5.4	9.0	10.0	10.5
Fish	96.7	95.6	95.6	94.1	15.5	11.8	12.9	13.2	14.2	13.5
Bird	92.5	87.9	84.2	76.3	6.2	4.4	10.4	14.8	14.8	18.6
Armadillo	91.9	90.1	84.0	83.7	8.3	6.3	8.0	8.4	8.4	8.6
Bust	67.2	62.1	53.9	52.2	22.0	18.8	21.4	22.2	33.4	39.3
Mech	94.6	90.5	88.9	82.4	13.1	8.5	10.0	11.8	12.7	24.0
Bearing	95.2	86.6	84.8	61.3	10.4	6.8	9.7	17.6	21.7	32.7
Vase	87.2	85.8	77.0	74.3	14.4	10.5	16.0	17.1	19.9	25.3
FourLeg	88.7	86.2	85.0	82.0	14.9	11.6	13.3	13.9	14.7	16.3
Average	**93.8**	**92.0**	**89.4**	**85.4**	**10.3**	**7.7**	**9.4**	**10.7**	**12.2**	**14.8**

Table 1: ***Left:*** *Recognition rate scores for our method across all categories in the benchmark, and for various training set sizes* $M = 3, 6, 12, 19$. ***Right:*** *Rand Index scores for human segmentations, training segmentations, and our method. Recognition rate is measured against our labeling dataset (see text for details), whereas the Rand Index is measured against all human segmentations in the Princeton benchmark.*

broad class of meshes.

Segmentation results. In contrast to labeling, we test our segmentation algorithm using all original human segmentations for all meshes, according to the protocol from Chen et al. [2009]. Results are shown in Figure 4, and comparisons to previous methods are shown in Figure 3. Our method gives a significant improvement over the previous state-of-the-art, according to all measures proposed by Chen et al. [2009]. Even when training on just three meshes, our method obtains better scores than other methods in nearly all cases. Table 1(right) provides Rand Index scores for each category and for different choices of training set size as above.

There are a few details to note about these experiments. First, Rand Cuts requires as input the number of segments for the mesh. For this, we used the average number of segments for each category. Second, the Human Score is worse than the score for our training

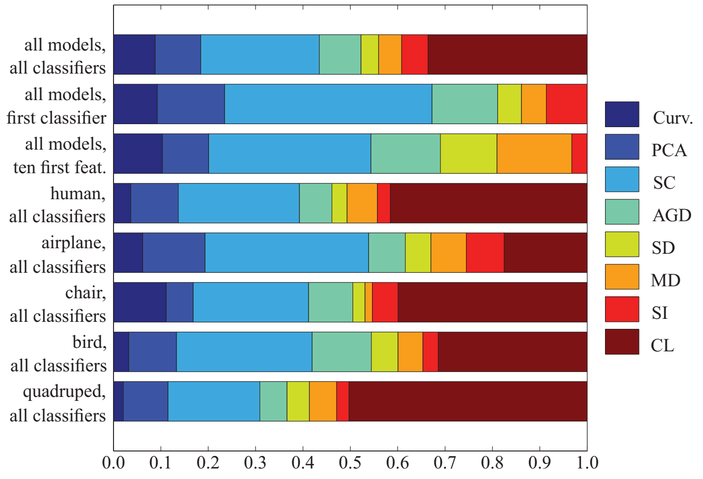

Figure 5: *Percentages of features used by JointBoost for different cases. See text for details. Legend: Curv.=curvature, PCA=PCA singular values, SC=shape contexts, AGD=average geodesic distances, SD=shape dimeter, MD=distance from medial surface, SI = Spin Images, CL = contextual label features.*

data (computed as in [Chen et al. 2009]) because we reduce the training set as described above. Third, when disconnected parts on the same mesh have the same label (e.g., the two hands on a human), they are scored as separate segments.

Feature selection. Figure 5 visualizes which features were selected by JointBoost in the unary term, for various subsets of the data. For example, the top row shows, for each type of feature, the percentage of this feature that was used, across the entire benchmark dataset. The most features came from Shape Contexts [Belongie et al. 2002] and the Contextual Label. The third row shows the ten features that were selected by the first ten rounds of JointBoost (i.e., the features used by the first ten decision stumps). The remaining rows show features used for individual categories. These results indicate that the Shape Context features were the most important among the basic features. However, each type of feature was used multiple times. This is a common theme among boosting algorithms: adding more features that provide independent sources of information typically improves results.

Generalizing to different categories. Fig. 6 shows results in which we train on one category and test on another. The algorithm

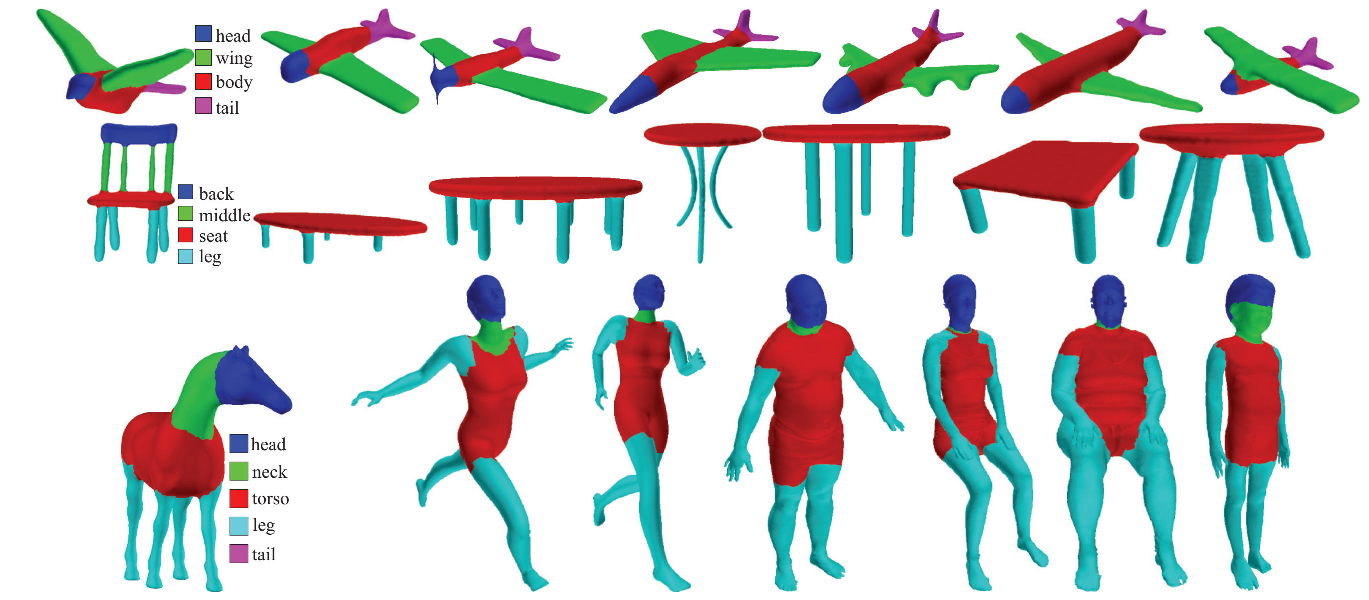

Figure 6: *Experiments where training and test categories are different. Sample training data shown on the left (complete training sets not shown).* ***Top row:*** *Training on birds, applying to planes.* ***Middle row:*** *Labeling tables as chairs.* ***Bottom row:*** *Labeling humans as quadrupeds. A failure case here is in the lower-right, where much of the child's face is confused for a neck. The seated humans illustrate a limitation of our method, i.e., that connected segments with the same label are not separated; here, a left arm is not separated from a leg when they connect.*

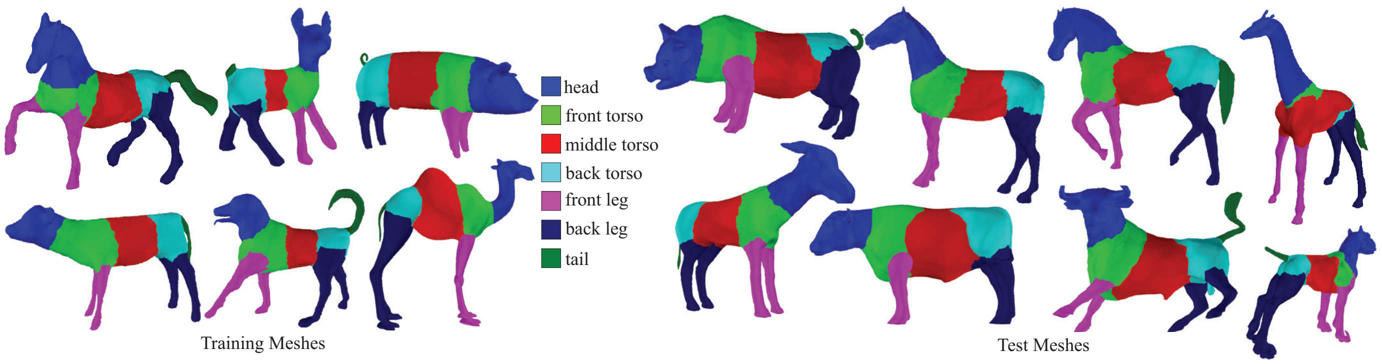

Figure 7: *Using an alternative segmentation style. Our main quantitative experiments used a segmentation style from the Princeton Benchmark in which each animal has a single torso (e.g., see Fig. 3). Here we train on examples from the Benchmark in which the torso is split into three segments. The six training meshes are shown on the left. Changing the style does not require any manual parameter tuning. Good results are obtained for several test meshes, except the giraffe, where the torso is not labeled accurately.*

yields reasonable results when labeling airplanes like birds, tables like chairs, and people like quadrupeds.

Different styles of segmentations. Our algorithm can be used to learn different styles of segmentation for different tasks. We demonstrate this capability with a set of animal segmentations from the benchmark data that separate the torso into three segments (Fig. 7), unlike the dataset used for quantitative evaluation. Our algorithm correctly applies these labels to several test meshes, except the giraffe.

Merging categories. Figure 8 shows an example in which a CRF was learned on a training set consisting of both humans and teddy bears, and applied to new humans and teddy bears. The algorithm successfully learns a model given the non-homogeneous data.

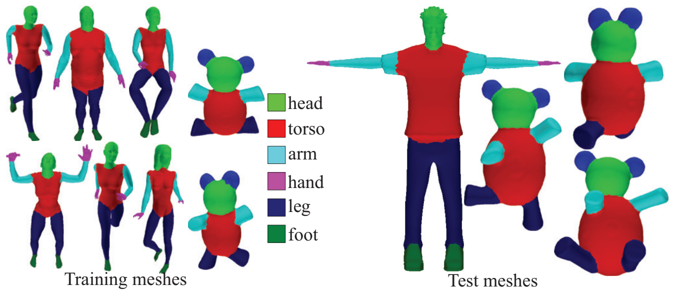

Figure 8: *Merging categories. A CRF was learned from the training meshes on the left, which include both humans and teddy bears. Results on a test human and bears shown on right.*

6 Applications

We now briefly describe a few procedures that illustrate how our approach could be used to automate workflows that would otherwise involve laborious manual effort. For each application, we implemented an automatic pipeline that takes a mesh of an object category as input, computes segmentation and labeling, and then processes the extracted parts. Such procedures could automate processing of large databases of objects of the same category.

Functional prototyping. Functional prototyping entails creating a real and working 3D object from a mesh, such as created by a designer. Our eyeglass pipeline (Figure 9(a-d)) takes a mesh as input and computes segmentation and labeling. The frontal silhouettes of the *lens* parts are offset, extruded, and subtracted from the object to create a frame. A frontal plane passing through the combined centroid of the two arm-lens segment boundaries is used to cut the mesh, separating the arms from the frame. Hinges and pins are created at the cut, resulting in a wearable pair of glasses. We have also implemented a procedure that, using a modeling tool like Teddy [Igarashi et al. 2007], converts a single sketched stroke into an articulated 3D mannequin, with joint-types based on extracted part labels (Figure 9i-k).

Rigging and texturing. Given an automatically computed segmentation and labeling, a skeleton may be created by placing joints at centroids of part boundaries. We further create texture for armadillo meshes (Figure 9(e-h)), using textures and accessories assigned to different labels, such as leathery skin for the feet, fur for the torso, and a hook in place of a missing hand.

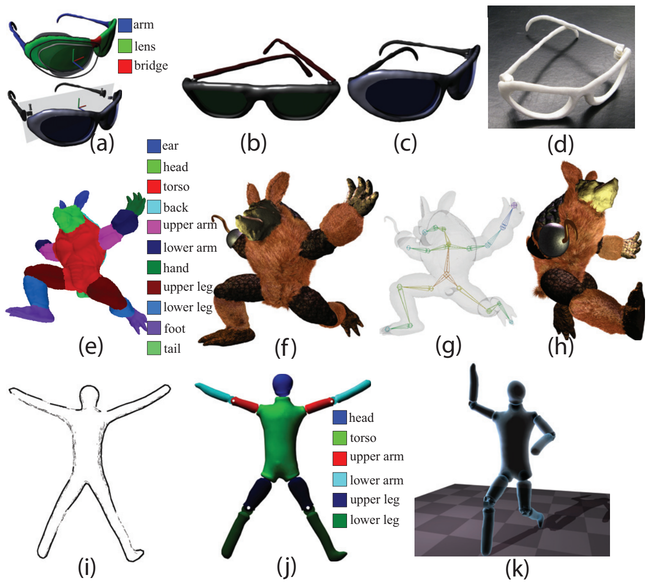

Figure 9: ***Top row:*** *Automatic procedure for converting Glasses meshes into manufacturable 3D objects with lenses and hinges.* ***(a)*** *The mesh is broken at the segment boundaries between the frame and arms with our labeling technique, and corresponding hinges are placed. Lenses are procedurally offset and subtracted from the frame.* ***(b-c)*** *Two example glasses created with this procedure.* ***(d)*** *A functional prototype, with working hinges, printed on a 3D printer.* ***Middle row:*** *Automatic shader assignment and rigging based on segment labels.* ***(e)*** *Labeled armadillo.* ***(f)*** *Procedural shaders assigned based on part labels, e.g., fur for the torso.* ***(g)*** *An animation skeleton is fitted to Armadillo automatically by placing the joints at the centroids of corresponding segment boundaries.* ***(h)*** *Posed armadillo.* ***Bottom row:*** *Automatic conversion of a 3D model drawn with the Teddy sketching package into an articulated mannequin.* ***(i)*** *Sketched 3D model.* ***(j)*** *Labeled model with mechanical joints placed at segment boundaries.* ***(k)*** *Articulated model.*

7 Discussion

We have described the first learning algorithm for both labeling and segmentation of 3D meshes. The model is learned from a training set without requiring any manual parameter tuning, and it obtains state-of-the-art results on all categories in the Princeton Segmentation Benchmark. Our method is the first to demonstrate effective labeling on a broad class of meshes. As our method represents an early attempt in this area, there are several limitations to our method (Figure 10), and many exciting directions for future work.

While considerable effort has rightly been put into devising geometric criteria for shape classification, it remains an open question as to whether simple geometric criteria are sufficient for segmenting the way humans do. Our work suggests that learning models from data—using carefully-chosen geometric features—can significantly improve results. While this method is not easily interpretable in terms of geometric intuitions, this kind of approach may nonetheless be of great practical value.

A major limitation of our approach is the need for labeled training data. The dataset must have consistent labels, although some variation can be tolerated. For example, in Figure 3, the pig does not have a neck segment, unlike the other meshes in the training data.

Generalization performance typically drops with fewer training meshes. Classes with larger variability across the data require larger training sets for good results. For example, the Ant and Octopus classes give good results with very few training examples, whereas the Bust and Vase categories give very poor results with small training sets (Table 1). For all classes, increasing the training set size improves performance.

Our method cannot learn "generic" segmentations, that is, segmentation without class-specific labels. The method cannot model segmentations where connected parts share labels (Figure 10(a-b)). We also assume that the target mesh is consistent with the training data; e.g., there are no outlier segments. However, we believe that elements of our approach could be useful for these or related problems. For example, our pairwise term could be used with a different unary term, such as one based on interactive labeling or mesh alignment.

Adding additional informative geometric features should improve results. At present, our algorithm cannot distinguish left/right/up/down (e.g., left arm vs. right arm); features informative of orientation [Fu et al. 2008] may help. Symmetry-based features and constraints could also be useful. Because many of our features depend on geodesic distances, they may not be very accurate when a test mesh exhibits significantly different topology than the training. Developing new part-aware and topology-insensitive shape descriptor features may help our method.

Our choice of features assumes that each shape is described by a watertight 3D mesh with a single connected component. Applying our technique for point clouds or polygon soups would require several modifications in our feature set. This should allow our method to be applied to data such as found in 3D scanning and architectural applications.

The size of our training set is limited by training time, which is several hours for our largest datasets e.g., training with 6 training meshes of about 20K-30K faces and six labels takes about 8 hours on a single Xeon E5355 2.66GHz processor. Once the model

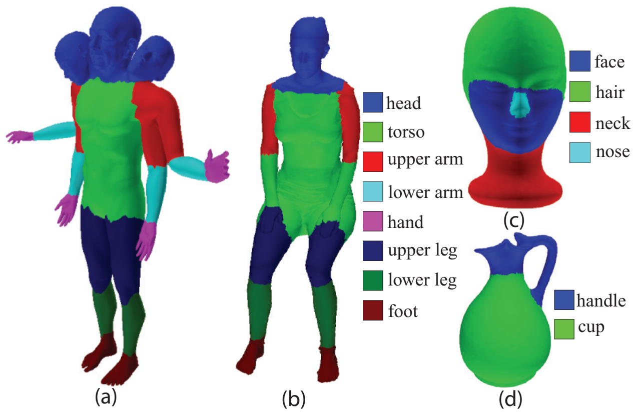

Figure 10: *Examples of limitations of our algorithm:* ***(a)*** *Shiva statue (not included in the benchmark), classified with a CRF learned from the Human category. The algorithm correctly labels the multiple heads and arms, but cannot separate connected segments with the same label.* ***(b)*** *Example of a test human mesh that has significantly different topology than the other training meshes of the Human category; its arms are connected to the legs, causing the algorithm to mislabel the lower arms, hands and upper torso.* ***(c)*** *Our lowest scores in the benchmark were on the Bust category; even when all the other busts are used as training meshes, our algorithm can still have significant errors.* ***(d)*** *Example of a vase, classified with a model learned from* 3 *other training meshes from the Vase category; the performance drops significantly in some categories with large variability, when few training meshes are used.*

is learned, applying it to new meshes is fast, usually only a few minutes per mesh. Our implementation is currently far from optimal, and faster training could allow learning a single model from all meshes in the Princeton Segmentation data.

Acknowledgements

We thank Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser for providing their segmentation benchmark and code, Szymon Rusinkiewicz for trimesh2, and Olga Veksler for the graph cut code. We thank Daniela Giorgi and AIM@SHAPE for providing the meshes from the Watertight Track of SHREC 2007 that are included in the benchmark. We also thank David Fleet and John Hancock for computing resources, and Olia Vesselova for proofreading. This project was funded by NSERC, MITACS, CFI, and the Ontario MRI. This work was done while AH was on a sabbatical visit at Pixar Animation Studios.

References

ANGUELOV, D., TASKAR, B., CHATALBASHEV, V., KOLLER, D., GUPTA, D., HEITZ, G., AND NG, A. 2005. Discriminative Learning of Markov Random Fields for Segmentation of 3D Scan Data. In *CVPR*.

ATTENE, M., KATZ, S., MORTARA, M., PATANE, G., SPAGNUOLO, M., AND TAL, A. 2006. Mesh Segmentation - A Comparative Study. In *Proc. SMI*.

ATTENE, M., FALCIDIENO, B., AND SPAGNUOLO, M. 2006. Hierarchical Mesh Segmentation Based on Fitting Primitives. *Vis. Comput. 22*, 3.

BELONGIE, S., MALIK, J., AND PUZICHA, J. 2002. Shape Matching and Object Recognition Using Shape Contexts. *IEEE Trans. Pattern Anal. Mach. Intell. 24*, 4.

BOYKOV, Y., VEKSLER, O., AND ZABIH, R. 2001. Fast Approximate Energy Minimization via Graph Cuts. *IEEE Trans. Pattern Anal. Mach. Intell. 23*, 11.

CHEN, X., GOLOVINSKIY, A., AND FUNKHOUSER, T. 2009. A Benchmark for 3D Mesh Segmentation. *ACM Trans. Graphics 28*, 3.

DUYGULU, P., BARNARD, K., DE FREITAS, N., AND FORSYTH, D. 2002. Object Recognition as Machine Translation: Learning a Lexicon for a Fixed Image Vocabulary. In *Proc. ECCV*.

FRIEDMAN, J., HASTIE, T., AND TIBSHIRANI, R. 2000. Additive Logistic Regression: a Statistical View of Boosting. *The Annals of Statistics 38*, 2.

FU, H., COHEN-OR, D., DROR, G., AND SHEFFER, A. 2008. Upright Orientation of Man-made Objects. *ACM Trans. Graph. 27*, 3.

GAL, R., AND COHEN-OR, D. 2006. Salient Geometric Features for Partial Shape Matching and Similarity. *ACM Trans. Graph. 25*, 1.

GAMA, J., AND BRAZDIL, P. 2000. Cascade Generalization. *Mach. Learn. 41*, 3.

GEMAN, S., AND GEMAN, D. 1984. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Trans. PAMI 6*, 6, 721–741.

GOLOVINSKIY, A., AND FUNKHOUSER, T. 2008. Randomized Cuts for 3D Mesh Analysis. *ACM Trans. on Graph. 27*, 5.

GOLOVINSKIY, A., AND FUNKHOUSER, T. 2009. Consistent Segmentation of 3D Models. *Proc. SMI 33*, 3.

GOLOVINSKIY, A., KIM, V. G., AND FUNKHOUSER, T. 2009. Shape-based Recognition of 3D Point Clouds in Urban Environments. In *Proc. ICCV*.

HE, X., ZEMEL, R., AND CARREIRA-PERPIÑÁN, M. A. 2004. Multiscale Conditional Random Fields for Image Labeling. In *Proc. CVPR*, vol. 2.

HILAGA, M., SHINAGAWA, Y., KOHMURA, T., AND KUNII, T. L. 2001. Topology Matching for Fully Automatic Similarity Estimation of 3d Shapes. In *SIGGRAPH*.

HUANG, Q., WICKE, M., ADAMS, B., AND GUIBAS, L. 2009. Shape Decomposition Using Modal Analysis. *J. Computer Graphics Forum 28*.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 2007. Teddy: A Sketching Interface for 3d Freeform Design. In *SIGGRAPH*.

JOHNSON, A., AND HEBERT, M. 1999. Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes. *IEEE Trans. PAMI 21*, 5, 433–449.

KATZ, S., AND TAL, A. 2003. Hierarchical Mesh Decomposition Using Fuzzy Clustering and Cuts. *ACM Trans. Graphics*.

KATZ, S., LEIFMAN, G., AND TAL, A. 2005. Mesh segmentation using feature point and core extraction. *Visual Computer 21*, 8.

KONISHI, S., AND YUILLE, A. 2000. Statistical Cues for Domain Specific Image Segmentation With Performance Analysis. *Proc. CVPR*.

KRAEVOY, V., JULIUS, D., AND SHEFFER, A. 2007. Model Composition From Interchangeable Components. In *Proc. PG*.

KUMAR, S., AND HEBERT, M. 2003. Discriminative Random Fields: A Discriminative Framework for Contextual Interaction in Classification. In *Proc. ICCV*.

LAFFERTY, J. D., MCCALLUM, A., AND PEREIRA, F. C. N. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *ICML*.

LAI, Y.-K., HU, S.-M., MARTIN, R. R., AND ROSIN, P. L. 2008. Fast Mesh Segmentation Using Random Walks. In *ACM symposium on Solid and Physical Modeling*.

LAVOUÉ, G., AND WOLF, C. 2008. Markov Random Fields for Improving 3D Mesh Analysis and segmentation. In *Eurographics workshop on 3D object retrieval*.

LI, X., GU, X., AND QIN, H. 2008. Surface matching using consistent pants decomposition. In *ACM Symposium on Solid and Physical Modeling*.

LIM, E., AND SUTER, D. 2007. Conditional Random Field for 3D Point Clouds With Adaptive Data Reduction. In *Cyberworlds*.

LIN, H.-Y. S., LIAO, H.-Y. M., AND LIN, J.-C. 2007. Visual Salience-Guided Mesh Decomposition. *IEEE Transactions on Multimedia 9*, 1.

LIU, R., AND ZHANG, H. 2004. Segmentation of 3D Meshes Through Spectral Clustering. In *Proc. PG*.

LIU, R. F., ZHANG, H., SHAMIR, A., AND COHEN-OR, D. 2009. A Part-Aware Surface Metric for Shape Analysis. *Computer Graphics Forum, (Eurographics 2009) 28*, 2.

MANGAN, A. P., AND WHITAKER, R. T. 1999. Partitioning 3D Surface Meshes Using Watershed Segmentation. *IEEE Trans. on Vis. and Comp. Graph. 5*, 4.

MUNOZ, D., VANDAPEL, N., AND HEBERT, M. 2008. Directional Associative Markov Network for 3-D Point Cloud Classification. In *Proc. 3DPVT*.

PEKELNY, Y., AND GOTSMAN, C. 2008. Articulated Object Reconstruction and Markerless Motion Capture from Depth Video. *J. Computer Graphics Forum 27*, 399–408.

SCHNITMAN, Y., CASPI, Y., COHEN-OR, D., AND LISCHINSKI, D. 2006. Inducing Semantic Segmentation From an Example. In *Proc. ACCV*.

SHAMIR, A. 2008. A Survey on Mesh Segmentation Techniques. *Computer Graphics Forum 26*, 6.

SHAPIRA, L., SHALOM, S., SHAMIR, A., ZHANG, R. H., AND COHEN-OR, D. In Press. Contextual Part Analogies in 3D Objects. *International Journal of Computer Vision*.

SHLAFMAN, S., TAL, A., AND KATZ, S. 2002. Metamorphosis of Polyhedral Surfaces Using Decomposition. In *Eurographics*.

SHOTTON, J., JOHNSON, M., AND CIPOLLA, R. 2008. Semantic Texton Forests for Image Categorization and Segmentation. In *Proc. CVPR*.

SHOTTON, J., WINN, J., ROTHER, C., AND CRIMINISI, A. 2009. TextonBoost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context. *Int. J. Comput. Vision 81*, 1.

SIMARI, P., KALOGERAKIS, E., AND SINGH, K. 2006. Folding Meshes: Hierarchical Mesh Segmentation Based on Planar Symmetry. In *SGP*.

SIMARI, P., NOWROUZEZAHRAI, D., KALOGERAKIS, E., AND SINGH, K. 2009. Multi-objective shape segmentation and labeling. *Computer Graphics Forum 28*, 5.

TORRALBA, A., MURPHY, K. P., AND FREEMAN, W. T. 2007. Sharing Visual Features for Multiclass and Multiview Object Detection. *IEEE Trans. Pattern Anal. Mach. Intell. 29*, 5.

TU, Z., CHEN, X., YUILLE, A., AND ZHU, S.-C. 2005. Image Parsing: Unifying Segmentation, Detection, and Recognition. *International Journal of Computer Vision 63*, 2.

TU, Z. 2008. Auto-context and its Application to High-level Vision Tasks. In *Proc. CVPR*.

ZHANG, E., MISCHAIKOW, K., AND TURK, G. 2005. Feature-based Surface Parameterization and Texture Mapping. *ACM Trans. Graph. 24*, 1.

A Unary Features

For each face i in a mesh, we compute a $375 + 35|\mathcal{C}|$-dimensional feature vector $\mathbf{x}_i$ to be used in the Unary Energy Term (Equation 3). Before computing any features, we normalize the scale of the mesh according to the 30th percentile of geodesic distances between all pairs of vertices. The features are as follows:

a) Curvature features: Curvatures have been used for partial matching (e.g., [Gal and Cohen-Or 2006]). Around each face, we fit cubic patches of various geodesic radii ($1\%, 2\%, 5\%, 10\%$ relative to the median of all-pairs geodesic distances). The patches are fitted using the face centers and normals and every sample is weighted with its face area. Let k_1 and k_2 be the principal curvatures of a patch. We include the following features: k_1, $|k_1|$, k_2, $|k_2|$, k_1k_2, $|k_1k_2|$, $(k_1+k_2)/2$, $|(k_1+k_2)/2|$, $k_1 - k_2$, yielding 36 features total.

b) PCA features: We compute the singular values s_1, s_2, s_3 of the covariance of local face centers (weighted by face area), for various geodesic radii ($5\%, 10\%, 20\%, 30\%, 50\%$), and add the following features for each patch: $s_1/(s_1+s_2+s_3)$, $s_2/(s_1+s_2+s_3)$, $s_3/(s_1+s_2+s_3)$, $(s_1+s_2)/(s_1+s_2+s_3)$, $(s_1+s_3)/(s_1+s_2+s_3)$, $(s_2+s_3)/(s_1+s_2+s_3)$, s_1/s_2, s_1/s_3, s_2/s_3, $s_1/s_2+s_1/s_3$, $s_1/s_2+s_2/s_3$, $s_1/s_3+s_2/s_3$, yielding 75 features total.

c) Shape diameter: The Shape Diameter Function (SDF) [Shapira et al. In Press] is computed using cones of angles 30, 60, 90, 120. For each cone, we get the weighted average, median, and squared mean of the samples. We include these shape diameters and their logarithmized versions with different normalizing parameters $\alpha = 1$, $\alpha = 2$, $\alpha = 4$, $\alpha = 8$. This yields 60 features representing different moments and approximations of the local shape diameter.

d) Distance from medial surface: For each of the cones above, we compute the diameter of the maximal inscribed sphere touching each face center and the corresponding medial surface point is roughly its center [Liu et al. 2009]. Then we send rays from this point uniformly sampled on a Gaussian sphere, gather the intersection points and measure the ray lengths. As with the shape diameter features, we use the weighted average, median and squared mean of the samples, we normalize and logarithmize them with the same above normalizing parameters. This yields 60 features.

e) Average Geodesic Distance: The Average Geodesic Distance (AGD) function has been used for shape matching [Hilaga et al. 2001; Zhang et al. 2005]. The function measures how “isolated” each face is from the rest of the surface e.g., limbs have usually higher AGD than other parts in humanoid models. The AGD for each face is computed by averaging the geodesic distance from its face center to all the other face centers. In our case, we also consider the squared mean and the $10th, 20th, ..., 90th$ percentile. Then, we normalize each of these 11 statistical measures by subtracting its mimimum over all faces.

f) Shape contexts: Shape contexts have been used for 2D shape matching [Belongie et al. 2002]. For each face, we measure the distribution of all the other faces (weighted by their area) in logarithmic geodesic distance bins and uniform angle bins, where angles are measured relative to the normal of each face. We use 5 geodesic distance bins and 6 angle bins, yielding 30 features total.

g) Spin images: Spin images [Johnson and Hebert 1999] are created with a fixed 8×8 bin resolution, yielding 64 features.

h) Orientation features: We also include the x, y, z coordinates of each face center in the case that the training dataset is oriented.

i) Contextual label features: The above features provide a feature vector $\tilde{\mathbf{x}}$, which are used to learn contextual features, as described in Section 3.4. The output of a JointBoost classifier provides per-face probabilities $P(c|\tilde{\mathbf{x}})$. The contextual features are histograms of these probabilities around each face:

$$p_i^l = \sum_{j:\, d_b \leq \mathrm{dist}(i,j) < d_{b+1}} a_j \cdot P(c_j = l) \tag{14}$$

where the bin b contains all faces j with distance range $[d_b, d_{b+1}]$ from face i. The a_j is the area of face j, normalized by the sum of face areas in the mesh. The distances between faces are measured from shortest parts (thus, approximating geodesic distances), as well as the Principal Component Axes and dominant symmetry axes of the mesh (measured in absolute values, since the principal axes are uniquely defined up to their sign). We use $B = 5$ ranges of distances $[d_b, d_{b+1})$ where d_b are chosen in the logarithmic space of $[0, \max_i(\max_j(\mathrm{dist}(i, j)))]$, yielding $35|\mathcal{C}|$ contextual features.

B Pairwise Features

For each pair of adjacent faces i and j, the following 191-dimensional feature vector $\mathbf{y}_{ij}$ is computed, for use in the Pairwise Energy Term (Section 3.2). We chose features that are potentially indicative of boundaries between parts.

a) Dihedral angles: Let ω_{ij} be the exterior dihedral angle between faces i and j. The basic feature is given as $\min(\omega_{ij}/\pi, 1)$. We also compute the average of the dihedral angles around each edge at geodesic radii of 0.5%, 1%, 2%, 4% of the median of all-pairs geodesic distances in the mesh. We then exponentiate each of the above features with each exponent in the range 1 to 10. This yields 50 dihedral angle features in total.

b) Curvature and third-order surface derivatives: We first compute the curvature and the derivative-of-curvature tensor per mesh vertex at geodesic radii of 0.5%, 1%, 2%, 4% of the median of all-pairs geodesic distances. For each scale, we include the principal curvatures and the curvature derivatives along the principal directions (in order to assign curvature to each edge, we average the corresponding curvature values of its vertices). This yields 16 features.

b) Shape diameter differences: For each pair of adjacent faces, we include the absolute values of the differences between their corresponding 60 shape diameter features (as described above).

d) Distance from medial surface differences: Similarly, we include the absolute difference of the 60 distance-from-medial-surface features between adjacent faces (as described above).

e) Contextual label features: We also use pairwise contextual features, as described in Section 3.4. The above features form an initial feature vector $\tilde{\mathbf{y}}_{ij}$. We learn a JointBoost classifier $p(c_i \neq c_j|\tilde{\mathbf{y}}_{ij})$, and then bin them, as with the unary contextual features. Here, we bin them based only on geodesic distances in logarithmic space up to 5% of the median of all-pairs geodesic distances in the mesh. This yields 5 pairwise contextual features in total.