Volume xx (200y), Number z, pp. 1–12

Learning boundary edges for 3D-mesh segmentation

Halim Benhabiles[1], Guillaume Lavoué[2], Jean-Philippe Vandeborre[1,3] and Mohamed Daoudi[1,3]

[1]LIFL (UMR Lille1/CNRS 8022), University of Lille 1, France
[2]Université de Lyon, CNRS, INSA-Lyon, LIRIS, UMR5205, F-69621, France
[3]Institut TELECOM; TELECOM Lille 1, France

Abstract

This paper presents a 3D-mesh segmentation algorithm based on a learning approach. A large database of manually segmented 3D-meshes is used to learn a boundary edge function. The function is learned using a classifier which automatically selects from a pool of geometric features the most relevant ones to detect candidate boundary edges. We propose a processing pipeline that produces smooth closed boundaries using this edge function. This pipeline successively selects a set of candidate boundary contours, closes them and optimizes them using a snake movement. Our algorithm was evaluated quantitatively using two different segmentation benchmarks and was shown to outperform most recent algorithms from the state-of-the-art.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Curve, surface, solid, and object representations—Boundary representations

1. Introduction

3D-mesh segmentation is a key ingredient for many applications such as indexing, compression, deformation, etc.

The recent creation of ground-truth databases for the segmentation of 3D-meshes [BVLD09, CGF09], has given to the computer graphics community the opportunity to quantitatively analyze and *learn* mesh segmentation. A very recent work based on learning has been proposed by Kalogerakis *et al.* [KHS10] and has demonstrated its efficiency through the improvement of the results over the state-of-the-art of mesh segmentation.

In this paper, we present a new fully automatic 3D-mesh segmentation algorithm based on boundary edge *learning*. Our algorithm is carried out using two main steps: an off-line step in which an objective boundary edge function is learned from a set of segmented training meshes, and an on-line step in which the learned function is used to segment any input 3D-mesh. The boundary function is learned using the AdaBoost classifier [FS97], which automatically selects from a set of geometric features the most relevant ones to detect candidate boundary edges. In the on-line step, the learned edge function is used successively to select a set of candidate boundary contours, to close them and to optimize them using a snake movement to produce the final segmentation. The best results are obtained when the learning is performed on objects from the same category as the object to segment. However, the results remain excellent even when we generalize the learning on different categories. Hence, we do not need to know the category of the input model to segment it.

Our algorithm provides very similar segmentations to those created by humans (see figure 1), and has been evaluated using the existing benchmarking systems [BVLD09, CGF09]. Our results outperform the best segmentation algorithms from the state-of-the-art in term of similarity to human segmentations. For instance, we achieved 8.8% Rand Index error on the Princeton segmentation benchmark [CGF09], while the last best result achieved on this database is 14.9% Rand Index error obtained by [GF08] (algorithm without learning), and 9.5% Rand Index error obtained by [KHS10] (algorithm with learning). This latter algorithm needs a consistent labeling of the training data, which may requires some manual interactions. However, it provides semantic labeling which allows for instance to retrieve a part of interest across a data-base of labeled 3D-models without additional processing.

Our main contributions are the following:

- The edge function determination based on multiple feature calculations and Adaboost learning.

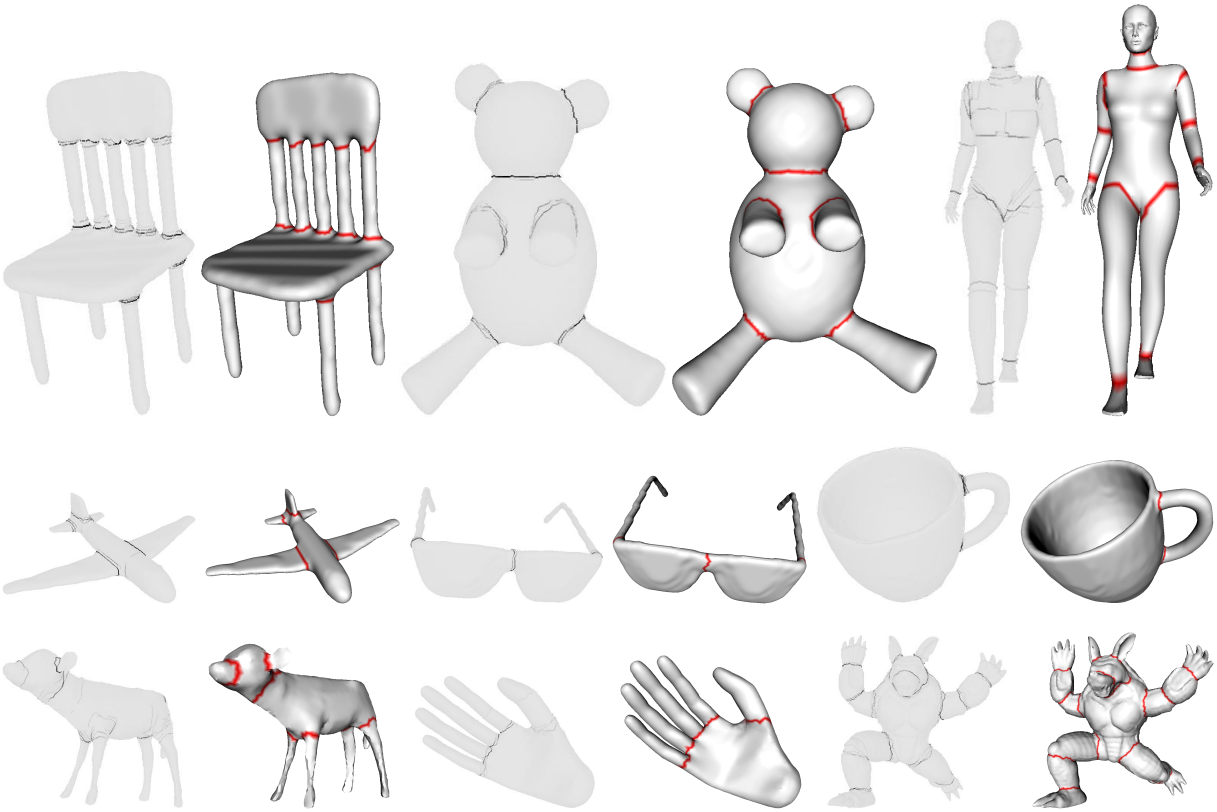


Figure 1: *For each pair of model: on the left, manual boundaries from the Princeton segmentation benchmark [CGF09] (the darkness degree of contours indicates that people have selected the same edges in their cuts); on the right, automatic boundaries from our algorithm.*

- The processing pipeline which produces smooth closed boundaries using the edge function. This pipeline includes the following stages: thinning, closing contours, and a snake movement. More details are provided on each stage in section 3.

2. Related work

During the last decade, a significant attention has been paid, by the computer graphics community, to 3D-mesh segmentation [Sha08, AKM*06]. Although some supervised methods exist [LHMR08, JLCW06], most existing techniques are fully automatic. They either concentrate on the definition of contours and then extract regions limited by these contours [GF08, LLS*05], or use clustering of similar mesh elements to define regions [SNKS09, LW08, SSCO08]. In both cases, they focus on analyzing either low level geometric information, or topological information of the input mesh. For instance, the use of geometric criterion includes curvature [LDB05], geodesic distances [KT03], dihedral angles [ZTS02], shape diameter function [SSCO08], planarity and normal directions [AFS06], etc. The use of topological criterion includes mainly Reeb-graphs [TVD07] and spectral analysis [ZvKD07]. Such criteria suffer either from sensitivity to local surface features and to pose changes or from the deterioration of the topology when connecting or disconnecting parts of the mesh. As raised by Kalogerakis *et al.* [KHS10], the main drawback of this kind of algorithms is the fact that they are limited to a single generic rule (e.g. skeleton topology) or a single feature (e.g. curvature tensor). Indeed, such algorithm cannot be suited to segment an input 3D-mesh which requires a combination of these criteria. Our algorithm does not suffer from this drawback since it is based on a learning approach which determines an edge function using multiple criteria.

According to our knowledge, only one work has been proposed that involves learning for 3D-mesh segmentation [KHS10]. It allows to simultaneously segment and label the input mesh, and is expressed as an optimization problem. The problem consists in optimizing a Conditional Random Field (CRF) of which an objective function is learned from a collection of labeled training meshes. We differ from this latter work in that instead of determining the suited label

of each mesh facet and then implicitly defining a segmentation resulting from this labeling, we explicitly determine the boundary edges that allow then to obtain smooth closed contours that define the segmentation. Moreover, even complex boundaries can be captured (see section 3), while in the previous work, the method rather aims to find compact regions.

Before these recent works for 3D-mesh segmentation, several advanced works have already been introduced for 2D-image segmentation based on learning approaches. Like for the 3D case, these algorithms use a model learned from a database of segmented 2D-images. These 2D-image segmentation algorithms based on learning can be grouped into two categories.

The first category covers algorithms that learn an optimal affinity function between each pixel of the input image and a set of prior defined labels [SWRC09, HZCp04]. A ground-truth (segmented and labeled images) is employed to train the classifier that allows to affect the proper label to each pixel.

The second category covers algorithms that use an objective function to classify edges [MFM04, KH04]. Each edge is classified as a boundary or a non-boundary using a classifier trained on the ground truth (segmented images), resulting in an edge image estimating human designated boundaries.

Our algorithm is inspired by the second category since it classifies edges as boundary or not, while the previous 3D-work [KHS10] is inspired by the first category.

3. Our segmentation approach

In this section, we describe our approach. We provide details on the two main steps of our algorithm: the off-line step in which the objective boundary function is learned using a set of segmented models, and the on-line step in which the learned function is used to segment the input mesh.

3.1. Off-line (learning) step

We formulate the problem of learning the boundary edges as a classification problem. The classification model is learned on a corpus of 3D-meshes accompanied by their manual segmentations using the AdaBoost classifier. The classifier takes as input a training data set and generates a function. The training data set is composed of a set of feature vectors F_E computed for each edge of the ground-truth corpus meshes. A feature vector F_E of a given edge contains a set of geometric criteria and is associated with its proper class label L so that $L = +1$ if the edge is a boundary (according to the manual segmentations of the mesh containing this edge) and $L = -1$ if the edge is not a boundary. Once the learning is done, the classifier produces a function (the boundary edge function). This function takes as input a feature vector from any given edge and outputs a signed scalar value whose sign will provide the estimated classification of the edge (positive for boundary and negative for non-boundary).

Now we briefly summarize, the set of geometric criteria that we use to characterize edges (and which compose the feature vector), and the AdaBoost classifier.

3.1.1. Feature vector

We compute a 33 dimensional feature vector for each edge. It includes a set of geometric criteria which are as follows:

- Dihedral Angle. The angle between two adjacent facets.
- Curvature. We compute different curvatures using the VTK library (`http://www.vtk.org/`). Let K_1, K_2 be the principal curvatures, we include: K_1, K_2, $K_1 \times K_2$ (Gaussian curvature), $(K_1 + K_2)/2$ (Mean curvature), $2/\pi * \arctan[(K_1 + K_2)/(K_1 - K_2)]$ (Shape index), and $\sqrt{(K_1^2 + K_2^2)/2}$ (Curvedness).
- Global curvature. We compute the mean curvature integrated over a large geodesic radius (10% of the bounding box lentgh) as in [LW08].
- Shape diameter. The shape diameter function [SSCO08] is computed using the default parameters: a cone with an opening angle of 120°, 30 rays, and a normalizing parameter $\alpha = 4$.

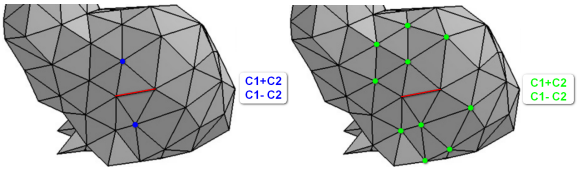
Note that we do not propose new geometric criteria, but we only employ existing ones used in previous segmentation algorithms. As stated in the introduction, the idea is to combine these criteria, then automatically select relevant ones with the appropriate weights using the classifier.

Except the dihedral angle which is computed for each edge, the other criteria (8 criteria) are computed for each vertex of the mesh. As illustrated in figure 2, to derive the criteria for each edge (the red one for instance), we consider its two opposite vertices (blue points in figure 2(a)). Then, considering that C_1 and C_2 are the values of a certain criterion computed respectively on these two vertices, we derive two feature values for the edge: $C_1 + C_2$ and $C_1 - C_2$. The idea is that, according to the nature of the criterion, in certain cases the sum can be relevant while in others the difference can carry a better information.

In order to bring a certain robustness to noise or sampling and to integrate a kind of multi-resolution behavior we also consider, in a second step, the 1-level neighborhood from each side of the edge (see green points in figure 2(b)). In that case C_1 and C_2 are respectively the means of the criterion from vertices at each side of the edge. This yields 32 features (16 in each case) to which we add the angle feature.

3.1.2. Adaboost classifier

AdaBoost is a machine-learning method that builds a strong classifier by combining weaker ones [FS97]. The algorithm takes as input a set of training samples $(x_1, y_1), .., (x_N, y_N)$; each x_i $(i = 1, ..., N)$ represents a feature vector (a vector



(a) (b)

Figure 2: *Example of edge criterion computation with one vertex on each side (a), and with a set of vertices (b).*

F_E in our case), and each y_i represents the class label of the sample x_i (y_i belongs to the domain $Y = \{-1,+1\}$ in our case). A large number of weak classifiers, $h_i : X \longrightarrow Y$, are generated, each one being a decision stump associated with a single dimension of the feature vector. Then, along a finite number of iterations $(t = 1,2,...,T)$, the algorithm iteratively selects the weak classifier which minimizes the current classification error. At the end a strong classifier H is produced as the combination of the weak ones weighted with α_t: $H(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$.

As stated at the beginning of subsection 3.1, the generated function $H(x)$ is now able to produce a signed scalar for each edge of an input 3D-mesh, whose sign gives its class label (positive sign for boundary edges and negative sign otherwise).

Note that we tested some other existing classifiers from the literature including non-parametric and parametric models such as Density estimation, HME (Hierarchical Mixtures of Experts), and SVM (Support Vector Machine). The performance was always nearly the same. We favor the AdaBoost since it yields a slight improvement over the other classifiers, and has the best running time.

3.2. On-line (segmentation) step

Figure 3 shows examples of edge classification results of some 3D-meshes. On the top, the result of the binary decision: boundary ($H(x) > 0$), non-boundary ($H(x) < 0$) is displayed (for visualization quality reason, we colored incident vertices of edges instead of coloring the edges themselves); while in the bottom, the edge function scalar values $H(x)$ are displayed using a color map. One can notice that the boundary edges from the binary decision (in red) are, neither smooth, nor closed. This result is expected since our classification model is learned on different objects (even if they belong to the same category), and uses multiple ground-truths per model which are not necessarily the same (boundaries are defined in a subjective way by humans, see figure 1). Hence it is not possible to directly consider this classifier output as a segmentation result.

To overcome this problem we propose a processing pipeline that transforms these non-connected fuzzy regions

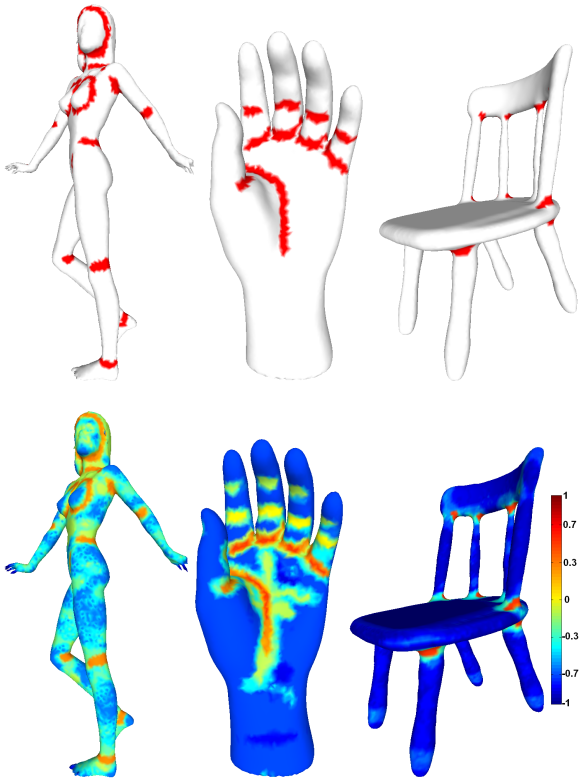
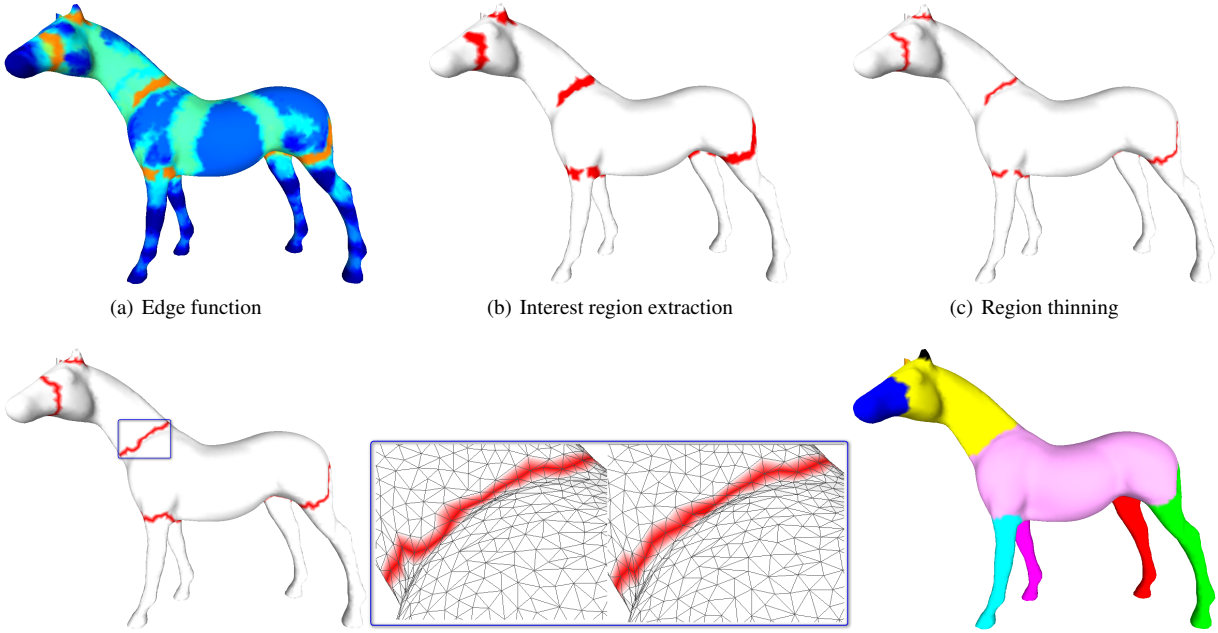


Figure 3: *Edge classification results for some 3D-meshes; (top: boundary edges after binary decision in red color; bottom: edge function scalar field).*

into thin, closed and smooth contours, by using the edge function. This processing pipeline comprises four stages. In the first stage of the process, given an input 3D-mesh, the edge function is computed (see figure 4(a)), and all edges having positive function values are selected. Theses edges constitute a set of interest regions (see figure 4(b)). Then, for each interest region (connected set of edges), a thinning algorithm [HG01] is applied. This latter algorithm gives as output a set of open linear contours (see figure 4(c)). Next, each open contour is completed using an improved version of the algorithm proposed by [LLS*05] based on the edge function (see figure 4(d)). At this step we have created a set of closed contours which represent a first version of the segmentation boundaries. However, these boundaries are often not smooth nor precise since in the thinning stage we do not consider any geometric information. To overcome this drawback, we apply an improved version of the snake movement algorithm proposed by [JK04] based also on the learned edge function. The snake movement allows to improve the quality of the boundaries in term of smoothness and precision without changing the mesh connectivity (see figure 4(e)). This set of improved boundaries defines the final segmentation (see figure 4(f)). These steps are detailed in the following



(a) Edge function (b) Interest region extraction (c) Region thinning

(d) Contour completion (e) Snake movement (f) Final segmentation

Figure 4: *Overview of the processing pipeline.*

subsections. Material illustrating the different steps of the algorithm is available at the third author's home page.

Note that in all our experiments (more than 350 models), we never encountered any topological problem (e.g., broken regions representing the same boundary) like in the work from Lee *et al.* [LLS*05]. The main reason is probably that our regions of interest come from a learning processed on different models associated with different manual segmentations; hence it introduces a kind of fuzziness which smooth/filter the results. Thus, we create one closed boundary for each connected region, even if two regions are close to each other. We do not process any contour filtering or contour merging.

3.2.1. Region thinning

In this stage, we transform a set of interest regions into a set of thin contours; this set of contours will be further used as the initial set of boundaries. Each interest region is represented by a set of connected edges. The algorithm from Hubeli *and* Gross [HG01] allows to thin a given interest region to a piecewise linear contour by deleting the edges from the border of the patch toward the inside. Initially, the algorithm inserts all border edges into a list. A border edge is an edge of which at least one of the four edges of its two opposite triangles does not belong to the interest region. Then, each border edge is deleted from the interest region if it does not disconnect this latter one. The deleting operation produces new border edges which are added to the list. The algorithm is performed until there is no removable edge. This leads to produce a connected skeleton of the interest region. Moreover, the algorithm allows to obtain a directly closed contour if the interest region forms a loop. Note that this algorithm does not contain any parameter setting.

However, it is possible to obtain a branching skeleton. Figure 5 illustrates an example in which a model has an interest region that leads to the creation of a branching skeleton after undergoing a thinning algorithm. This skeleton is composed of external and internal branches. An external branch is limited by one endpoint and one junction point while an internal branch is limited by two junction points. We consider that only two external branches are correct regarding the real boundary and we consider others like noise; to select these two relevant branches we compute a weight for each external branch by summing the learned function values of their edges, and we keep the two branches that have the highest weights together with the internal branches that connect them (in the case where they do not share the same junction point). We precise here that according to our experiments, such branching skeletons appear mostly when the corresponding interest region is very large, which almost never happens.

3.2.2. Contour Completion

In this stage, each open contour is completed to form a closed boundary around a specific part of the input mesh.

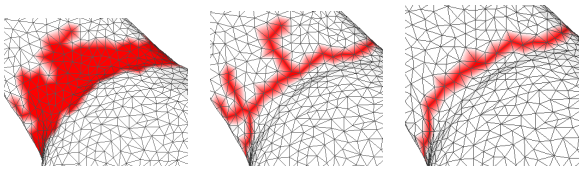


Figure 5: *From left to right: the interest region, the branching skeleton after thinning, the open boundary after removing the noisy branches for the horse model.*

To this end, we propose a modified version of the completion algorithm from Lee *et al.* [LLS*05]. The principle is to find the weighted shortest path between the two endpoints of the contour.

Let ζ be an open contour composed of a set of mesh vertices v. To close the open contour, we search the shortest path between the two end points of ζ by selecting among candidate edges using the following edge cost function:

$$cost(e) = \eta_d(e)^{w_d}.\eta_n(e)^{w_n}.\eta_e(e)^{w_e}$$

where $\eta_d(e)$ and $\eta_n(e)$ are defined as the average of values of the two incident vertices of the edge e.

η_d is the distance function. It measures the distance from ζ to a given vertex v as:

$$\eta_d(v) = \sum_{v_i \in \zeta} \frac{1}{d(v, v_i)}$$

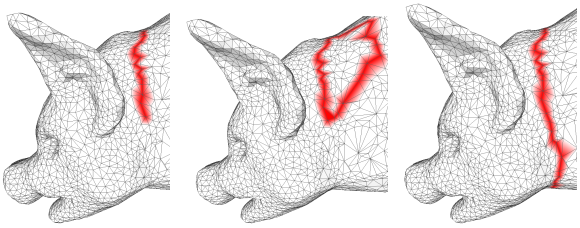
where $d(v, v_i)$ is the Euclidean distance. The function is high in the neighborhood of the contour ζ and decreases otherwise.

η_n is the normal function. It helps to go over the other side of the mesh, and is defined for a given vertex v as:

$$\eta_n(v) = \begin{cases} 1, & \text{if } n_\zeta.n_v \geq \cos(\alpha) \\ \frac{n_\zeta.n_v+1}{\cos(\alpha)+1}, & \text{else} \end{cases}$$

n_ζ is the average normal vector of ζ, n_v is the normal vector of vertex v, and α is the angle between the normals of the two endpoints of ζ.

η_e is the feature function; in the original work from Lee *et al.* [LLS*05] the feature function includes minimum curvature and centricity. The centricity of a vertex is defined as the average geodesic distance from the given vertex to all other vertices of the mesh. As stated by the authors, the original algorithm sometimes failed to correctly close the open contours. In our modified version, we replace the feature function by our learned edge function; it guides the path towards the regions learned as boundaries according to the results of the classifier. The results are significantly improved (see an example in figure 6).



(a) (b) (c)

Figure 6: *Example of completing a contour on a 3D-mesh (a) using: the original version of the algorithm from Lee* et al. *[LLS*05] based on their feature function (b), and the improved version based on our learned edge function (c).*

Note that the three function values are normalized in the range $[0, 1]$. We set the weights w_d,w_n to 1 and w_e to 0.4.

3.2.3. Snake movement

The snake movement is used to optimize the set of closed contours resulting from the previous stage. Each contour is used as the initial position of the snake. We propose a modified version of the snake movement algorithm from Jung *et al.* [JK04]. The algorithm is based on an iterative process in which the snake evolves (each vertex of the snake is moved to one of its neighbor vertices on the mesh) by minimizing an energy functional composed of internal and external parts until it is adjusted (see figure 4(e)). In the original work from Jung *et al.* [JK04], the internal energy controls the length and the smoothness of the snake (i.e. the closed contour), and is defined for a given vertex v_i as:

$$E_{int}(v_i) = \alpha \|v_i - v_{i-1}\| + \beta \|v_{i+1} - 2v_i + v_{i-1}\|$$

where α and β are tuning parameters that affect respectively the smoothness of the snake in term of distance and curvature. We set the α to 0.2 and the β to 0.8. The external energy controls the fitting of the snake to the desired feature, and is defined in [JK04] by the maximum principal curvature. In our modified version, we replace the maximum curvature by the learned edge function again. The external energy of a given vertex is then computed by averaging the edge function values of its incident edges, and normalizing them in the range $[0, 1]$ after reversing the sign since we aim to minimize the energy. The modification of the external energy is justified by the fact that the original algorithm aims to find features related to ridges and valleys based only on a single geometric criterion (maximum curvature). Hence, when replacing the maximum curvature by the edge function, the quality of boundaries is clearly improved (see experiment 4.4), since this latter function is based on multiple geometric criteria which are learned to detect boundaries.

4. Experiments

Our segmentation method was evaluated and compared quantitatively to the most recent algorithms from the state-of-the-art. To this end, we used two recent benchmarks dedicated to 3D-mesh segmentation evaluation, namely the Princeton benchmark [CGF09] (http://segeval.cs.princeton.edu/), and the LIFL/LIRIS benchmark [BVLD09] (http://www-rech.telecom-lille1.eu/3dsegbenchmark/).

Note that we used the same control parameter values in all our experiments, except when we explicitly modify the threshold of the H function for the hierarchical segmentation experiment (see section 4.5). The different parameters are set as follows:

- H Function threshold: $H(x) > 0$.
- Thinning and branch-filtering: no parameter.
- Contour completion: w_d,w_n to 1 and w_e to 0.4 of $cost(e)$.
- Snake: α to 0.2 and β to 0.8 of E_{int}.

4.1. Segmentation results on the Princeton benchmark

The Princeton segmentation benchmark provides 19 categories of 3D-meshes, each one containing 20 3D-models accompanied with multiple ground-truth segmentations (manual segmentations). Our segmentation method was trained and tested on this benchmark, using different learning strategies, namely categorical learning and global learning.

In the first type of learning (categorical), we train and test our algorithm class by class. Similarly to [KHS10], we evaluate our method using leave-one-out cross-validation. For each mesh i in each class, we learn the edge function on the 19 other meshes from that class using all ground-truths, and then we use that function to segment the mesh i. In order to analyze the effect of the training set size on the quality of the results, we repeat the same experiment, using less meshes in the training set: we learned the edge function on 6 meshes randomly selected for each class.

In the second type of learning (global), we learn the edge function in a generic way using a subset of the whole database (6 models randomly selected from each category), then we test on the remaining models (14*19 models). In this generic (or global) learning scenario, we do not need to know the category of the model to segment.

To evaluate the quality of the segmentation produced by our algorithm, we follow the protocol defined in the Princeton segmentation benchmark. Figure 7 shows the Rand Index error averaged over all models of the corpus for our algorithm, using the different learning strategies, and for the most recent algorithms from the state-of-the-art. The first point to make is that, when using a categorical learning with a training set size of 19 models, our algorithm provides very high quality results; indeed, our algorithm yields the smallest Rand Index error (8.8%) among all the other algorithms. One can notice also on the figure that when reducing the training set size to 6 models, and keeping a categorical learning, our algorithm still provides very good results with a slight drop of performance (9.7% Rand Index Error). The second point to make is that, our algorithm performs better than the algorithm from Kalogerakis *et al.* [KHS10] which is also based on learning (with the same training set sizes: 19 and 6 models).

However, categorical learning involves the fact that before segmenting a model, you have to know its category hence it is not really fair to compare these results with generic algorithms [GF08, SSCO08, LHMR08, AFS06] which do not need such knowledge of the data. When using Global learning, our algorithm does not need this prior knowledge and thus can be compared with generic algorithms; the fact is that it significantly outperforms them: its Rand Index Error is 10.4% while the value of the second best [GF08] is 14%.

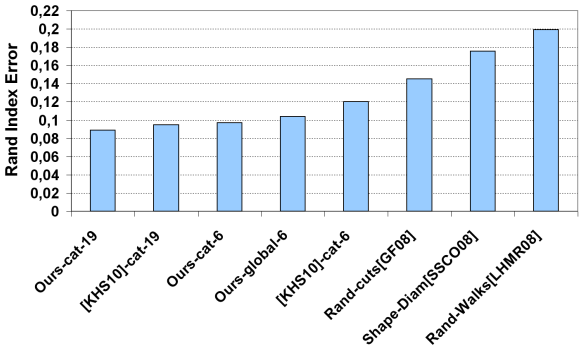


Figure 7: *Rand Index Error averaged over all the Princeton corpus models and sorted in increasing order for different algorithms. Reference-type-size represent the Index Error of algorithms based on learning with: learning type (categorical or global), size of the used training set (19 and 6 models).*

It is interesting to study which criteria are selected by the classifier in the learning step. Figure 8 illustrates the percentage of each criterion selected by the AdaBoost classifier for both types of learning (categorical for several categories such as *bust*, *human*, etc. and global). We can notice that, whatever the learning strategy (categorical or global), all the criteria are selected by the classifier, hence they all contribute to the results. However, the distribution of the criteria percentages differs from a category to another. For instance, in the *fourleg* category, the most used criteria are the shape diameter and minimum curvature, while in the *table* category, the most used are the dihedral angle and maximum curvature. One can notice also a more isotropic distribution between the different criteria in the global learning with comparison to the categorical learning. This is due to the variety of 3D-meshes included in the training. All categories together may have different shapes and topologies,

and then they do not share necessarily the same important features. In this case one or two criteria are clearly not sufficient to obtain a correct segmentation.

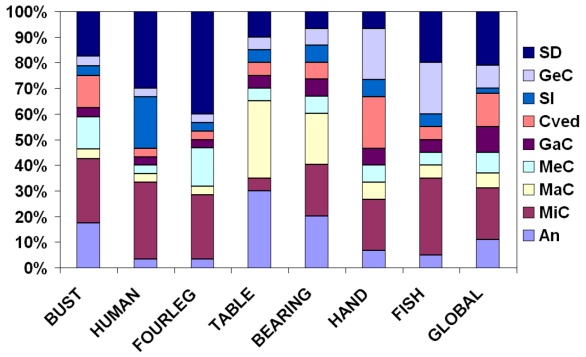


Figure 8: *Percentage of criteria selected by AdaBoost: for a categorical learning of size 19, and for a global learning of size 6. Legend: An (Angle), MiC (Minimum Curvature), MaC (Maximum Curvature), MeC (Mean Curvature), GaC (Gaussian Curvature), Cved (Curvedness), SI (Shape Index), GeC (Geodesic Curvature), SD (Shape Diameter).*

Figure 9 shows a visual comparison between our segmentation results (categorical learning - 19 training models) and those from recent algorithms from the state-of-the-art on some 3D-meshes from the Princeton benchmark; the average of manual segmentations (ground-truths) is also included. The quality of our algorithm is confirmed; indeed, our segmentations appear better than those of the other methods in term of similarity to the ground-truths, and particularly regarding boundary precision.

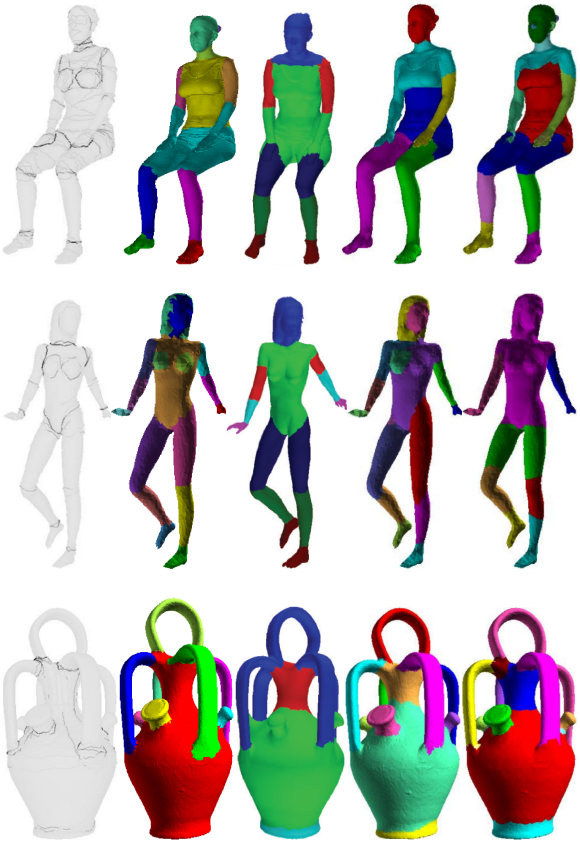


Figure 9: *From left to right segmentations obtained by: average of ground-truths of Princeton benchmark, our algorithm trained on the Princeton benchmark, [KHS10], [GF08], [SSCO08].*

4.2. Genericity of the learning across databases

In a second experiment, we still have trained our edge function on the Princeton benchmark but we have launched the segmentation on a different benchmark, the LIFL/LIRIS benchmark which contains a different set of 3D-models. Besides, the 3D-models are associated with vertex-based manual segmentations, while those of the Princeton benchmark are associated with facet-based segmentations. The LIFL/LIRIS benchmark contains 28 3D-models grouped in five classes. Each 3D-model is associated with 4 ground-truths. Similarly to the previous experiment, to evaluate the segmentation produced by our algorithm, we have followed the protocol defined in the benchmark. The edge function used to segment the models of this benchmark was learned on the Princeton segmentation benchmark using the global learning, with a training set size of 6 models. Figure 10 shows the NPRI (Normalized Probabilistic Rand index) averaged over all the corpus models for different algorithms including ours. Contrary to the Rand Index Error, the NPRI [BVLD10] gives an indication about the similarity degree between the automatic segmentation and the manual segmentations. It is in the range [-1,1]. A value of 1 indicates a complete similarity, whereas a value of -1 indicates a maximum deviation between the segmentations being compared. The figure clearly shows the significant improvement of the results obtained by our method with comparison to the others. More precisely, our method reaches 60% of similarity rate, when the best result reached by the other methods on the same corpus is 50%. We have to precise here that these good results confirm the robustness and the genericity of our learning since we have trained our edge function on a different database containing different models.

Figure 11 shows segmentations obtained by our algorithm for some 3D-meshes selected from INRIA (`http://www-roc.inria.fr/gamma/`), TOSCA (`http://tosca.cs.technion.ac.il/`), and Stanford (`http://graphics.stanford.edu/data/`) databases. The edge function used to segment the models was learned on the Princeton segmentation benchmark (global learning, 6 models). Again, our al-

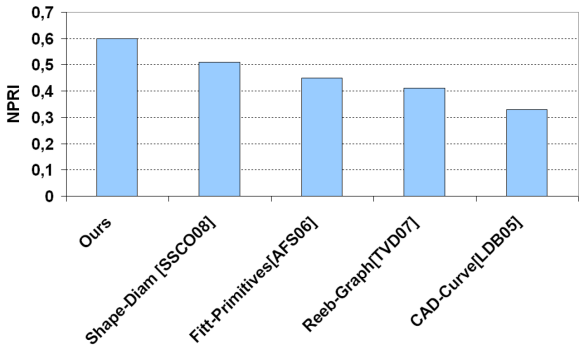


Figure 10: *Scores of NPRI (Normalized Probabilistic Rand Index) averaged over all the LIFL/LIRIS corpus models and sorted in decreasing order for different algorithms. Although in this experiment, our method is based on a global learning, performed on a different database, it outperforms the others.*

gorithm correctly segments these meshes, and finds a set of meaningful parts.

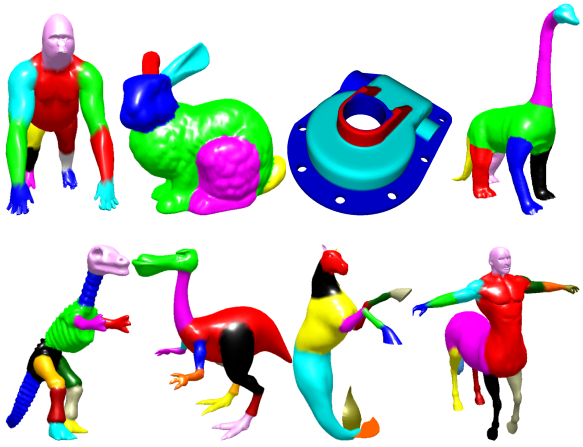


Figure 11: *Segmentations results, obtained by our algorithm trained on the Princeton benchmark, for a variety of meshes from different databases.*

4.3. Algorithm efficiency regarding the category of models

In this third experiment, we want to study the behavior of our algorithm regarding the different categories of models in both benchmarks. For this reason we use the NPRI which is a more discriminative metric than a simple Rand Index (see [BVLD10] for more details). The NPRI is computed for each model, then averaged by category. Figure 12 illustrates the results obtained by our algorithm for the Princeton corpus models, and for the LIFL/LIRIS corpus models. Note that for the first corpus, we use a categorical learning of size 19, while for the second corpus, we use a global learning of size 6, both trained on the Princeton benchmark. Globally, we can notice that for both corpuses, our algorithm gives quite good results for each category since the scores are much higher than zero. An interesting point is that the scores of common categories among the two corpuses are consistent, with a slight drop of performance for the LIFL/LIRIS corpus which is due to the difference of learning strategy (categorical vs global on a different database). The figure illustrates also that the *bust* category seems to be the most difficult one to segment (it is associated with the smallest NPRI values for both benchmarks). The fact is that human face images are well-known in subjective experiments as a high-level factor attracting human attention, hence some features not relevant from a geometrical point of view can be considered highly relevant for human observers and thus can influence the manual ground-truth segmentations.

4.4. Study of the performance of our improved snake movement

In this experiment, we show how the use of the learned boundary edge function improves the original snake movement algorithm from Jung *et al.* [JK04]. To this end, we compute the similarity between boundaries extracted by our algorithm using both versions of the snake movement, and the manual boundaries on the LIFL/LIRIS corpus models. The most appropriate metric to compute this kind of similarity is the CDI (Cut Discrepancy Index) [BVLD10], since it allows to compute the mapping degree between the extracted boundaries of two segmentations of the same model. The metric is in the range $]-\infty;1]$, and a value of 1 indicates a perfect matching between boundaries of the two segmentations. Figure 13 illustrates the scores of CDI averaged for each category of models and over all models of the LIFL/LIRIS corpus for these cases. The results show clearly that the new snake movement always improves the quality of the boundaries, whatever the category of models.

4.5. User interaction and coarse to fine segmentation

One of the strong point of our algorithm is that it is fully automatic; in particular the number of boundaries (and thus the number of segments) is automatically determined within our processing pipeline; it corresponds to the number of connected interest regions from the edge classification step (see figure 4(b)). However if needed, it is quite easy to introduce human interactions in our process. This can be done by:

- Tuning the classification threshold applied on the edge function. This threshold is set to 0 in our method; however it is still possible to decrease (resp. increase) this threshold to obtain more (resp. less) segments. Such a coarse to fine segmentation is illustrated in figure 14. We have to notice however that for a too low threshold (see the right

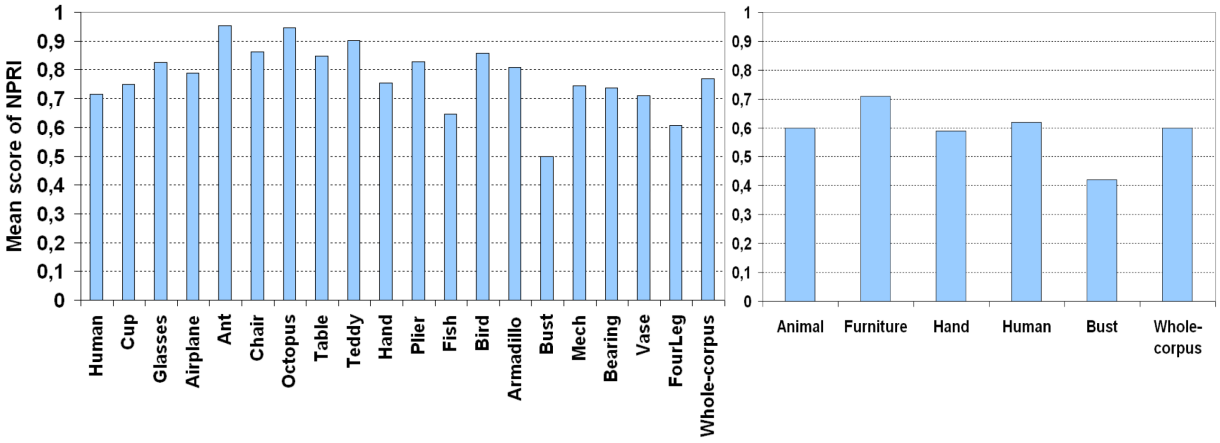


Figure 12: *From left to right, scores of NPRI averaged for each category and for all models from the Princeton benchmark [CGF09], and from the LIFL/LIRIS benchmark [BVLD09].*

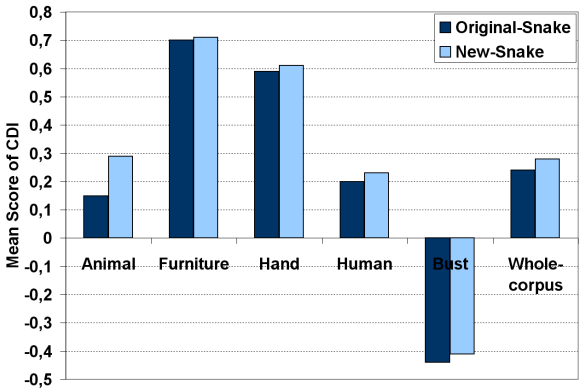


Figure 13: *Scores of CDI averaged for each category and over all models of the LIFL/LIRIS corpus with the original [JK04] and our improved version of the snake movement.*

(a) $H(x) > 0.3$ (b) $H(x) > 0$ (c) $H(x) > -0.2$

Figure 14: *Example of coarse to fine segmentation obtained by tunning the classification threshold applied on $H(x)$.*

hand in figure 14) some regions of interest may become very large and thus may be abnormally merged leading to merging some segments (like the bottom of the fingers). However this kind of problems never happened in all our experiments, with a threshold set to 0.

- Using a paintbrush, to directly select on the mesh a set of edges representing a new interest region, similarly to [LLS*05]. The segmentation process is then completed by performing the remaining steps.

4.6. Algorithm robustness regarding geometric transformations

We assessed the robustness of our method against two kinds of transformations, namely pose-variation and noise.

- Pose-variation: figure 15 shows the segmentations obtained by our algorithm for the *armadillo* and *human* models with different poses. These models are available in the Princeton segmentation benchmark. The edge function used to segment the models is based on a global learning, with a training set size of 6×19 models (6 models selected from each category). Globally, the segmentation of the models is quite stable, which underlines the pose robustness of our algorithm.
- Noise: we applied on the vertices of the test models two random displacements in the direction of $x-$, $y-$, and $z-$axis (3% and 5% of the bounding box length). Figure 16 shows the boundaries extracted by our algorithm for the noisy versions of the *ant* model. On the top, the boundaries are generated using an edge function based on

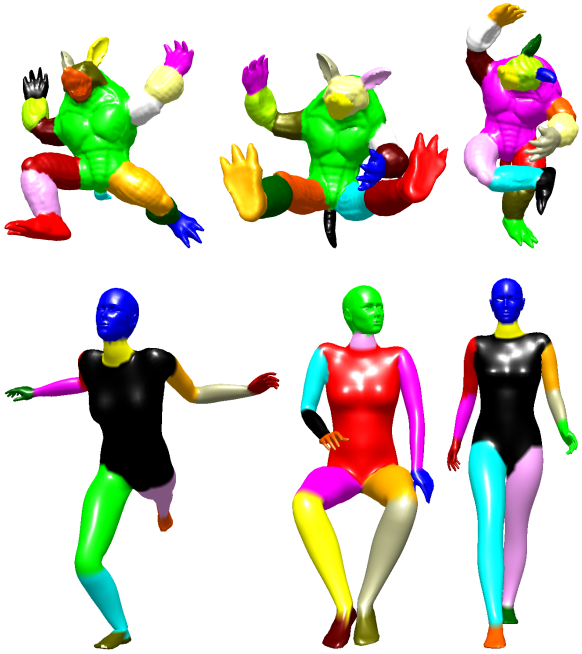
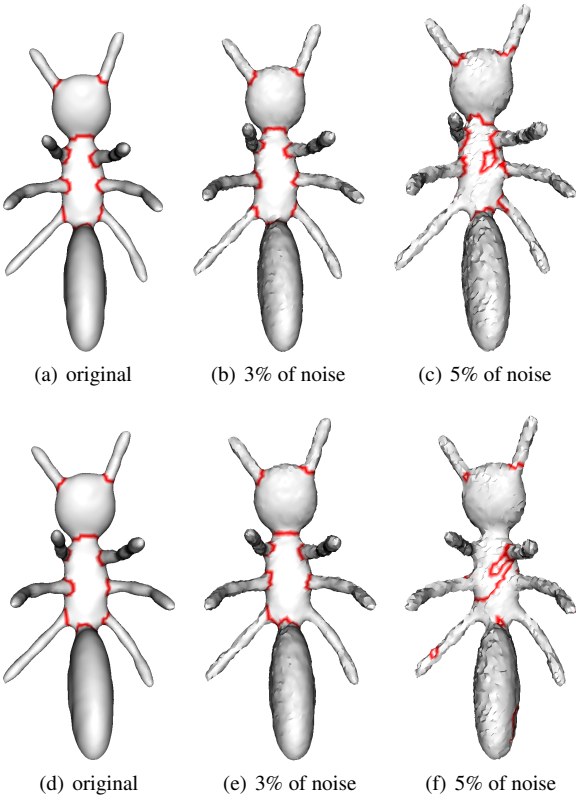


Figure 15: *Algorithm robustness against pose-variation.*



(a) original (b) 3% of noise (c) 5% of noise

(d) original (e) 3% of noise (f) 5% of noise

Figure 16: *Algorithm robustness against noise; (top: boundary extraction based on categorical learning, bottom: boundary extraction based on global learning).*

a categorical learning, with a training set size of 19 models (all the training models belong to the *ant* category); while in the bottom, the edge function is based on a global learning, with a training set size of 6×19 (6 models selected from each category). When applying a noise of 3%, the results remain very good for the categorical learning and correct for the global learning. However when applying a strong noise (5%) the quality of the boundaries is seriously degraded particularly when using the global learning. We have conducted tests on 19 models (one model randomly selected from each category of the Princeton segmentation benchmark) for 3% and 5% of noise, the quality of segmentation decreases respectively of 1% and 12% when using the categorical learning, and of 10% and 50% when using the global learning. The quality is measured by computing the NPRI for each test model. This moderate robustness is due to the fact that the learning step is performed on clean data; hence the learned function fails to extract the right boundaries in the noisy models since the edges composing them do not share the same geometric properties as in the clean ones. A solution could be to artificially add noise on the segmented training data, before the feature extraction and learning.

4.7. Running time

All the previous experiments were carried out on a 2.99 GHz Intel(R) Core(TM) 2 Duo CPU with 3.25 Gb memory. Globally, the running time for the learning step is less than 10 minutes for most of the categories. The process is longer, around 40 minutes, for some categories of which the size of models is important such as the *armadillo* category. The online step (segmentation) runs at an interactive time (around one minute), except for the *armadillo* category for which the running time is more important (around 9 minutes). This is due to the size of mesh and the number of extracted boundaries which are both important (24k vertices and 16 boundaries on average). More precisely, the average running time in seconds for the thinning, contour completion, and snake movement steps is respectively 0.58, 3.81, and 45.95.

5. Conclusion

In this paper we have presented a framework for segmentation based on a learning approach. A boundary edge function is learned from a set of ground-truths using AdaBoost classifier, and then is used to segment any input 3D-mesh through different processing steps. Our framework is the first to address the problem of learning boundary edges for 3D-mesh segmentation. We have shown the possibility to make a generic (or global) learning that can lead to segment a model

without having a prior information about its category. Our algorithm outperforms the most recent algorithms from the state-of-the-art, and produces segmentations which are very similar to those created by humans.

Although these results are accurate, some limitations remain and require a thorough analysis and future work. First, the number of geometric criteria used for classification could be more important. In particular, adding some rich features from 3D shape retrieval research fields would certainly improve the results and the sensitivity regarding to noise. The *bust* models are difficult to segment even with a training set of size 19, and our algorithm rather produces a coarse segmentation for this kind of models. This is probably due to the high semantic aspect carried by a face. This semantic aspect influences the manual segmentations, and is difficult to capture using simple local geometric criteria. Perhaps the solution is to include more global features such as topological ones; another solution would be to learn a whole structure of the segmentation per category (a prior graphical model from the manual segmentations), instead of a simple binary classification model.

ACKNOWLEDGMENTS

This work is supported by the ANR (Agence Nationale de la Recherche, France) through MADRAS project (ANR-07-MDCO-015).

References

[AFS06] ATTENE M., FALCIDIENO B., SPAGNUOLO M.: Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer 22*, 3 (2006), 181–193. 2, 7

[AKM*06] ATTENE M., KATZ S., MORTARA M., PATANÉ G., SPAGNUOLO M., TAL A.: Mesh segmentation, a comparative study. *IEEE International Conference on Shape Modeling and Applications (SMI)* (2006). 2

[BVLD09] BENHABILES H., VANDEBORRE J.-P., LAVOUÉ G., DAOUDI M.: A framework for the objective evaluation of segmentation algorithms using a ground-truth of human segmented 3D-models. In *IEEE International Conference on Shape Modeling and Applications (SMI)* (2009), pp. 36–43. 1, 7, 10

[BVLD10] BENHABILES H., VANDEBORRE J.-P., LAVOUÉ G., DAOUDI M.: A comparative study of existing metrics for 3D-mesh segmentation evaluation. *The Visual Computer 26*, 12 (2010), 1451–1466. 8, 9

[CGF09] CHEN X., GOLOVINSKIY A., FUNKHOUSER T.: A benchmark for 3d mesh segmentation. *ACM Transactions on Graphics (SIGGRAPH) 28*, 3 (2009). 1, 2, 6, 10

[FS97] FREUND Y., SCHAPIRE R. E.: A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences 55*, 1 (1997), 119–139. 1, 4

[GF08] GOLOVINSKIY A., FUNKHOUSER T.: Randomized cuts for 3d mesh analysis. *ACM Transactions on Graphics 27*, 5 (2008), 1–12. 1, 2, 7, 8

[HG01] HUBELI A., GROSS M.: Multiresolution feature extraction for unstructured meshes. In *Conference on Visualization (VIS'01)* (2001), pp. 287–294. 4, 5

[HZCp04] HE X., ZEMEL R. S., CARREIRA-PERPIÑÁN M. Á.: Multiscale conditional random fields for image labeling. In *IEEE Computer Vision and Pattern Recognition (CVPR)* (2004), pp. 695–702. 3

[JK04] JUNG M., KIM H.: Snaking across 3d meshes. In *12th Pacific Conference on Computer Graphics and Applications (PG'04)* (2004), IEEE Computer Society, pp. 87–93. 4, 6, 9

[JLCW06] JI Z., LIU L., CHEN Z., WANG G.: Easy mesh cutting. *Computer Graphics Forum 25*, 3 (2006), 283–291. 2

[KH04] KAUFHOLD J., HOOGS A.: Learning to segment images using region-based perceptual features. *IEEE Computer Vision and Pattern Recognition (CVPR)* (2004). 3

[KHS10] KALOGERAKIS E., HERTZMANN A., SINGH K.: Learning 3D Mesh Segmentation and Labeling. *ACM Transactions on Graphics (SIGGRAPH) 29*, 3 (2010). 1, 2, 3, 7, 8

[KT03] KATZ S., TAL A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics (SIGGRAPH) 22*, 3 (2003), 954–961. 2

[LDB05] LAVOUÉ G., DUPONT F., BASKURT A.: A new cad mesh segmentation method based on curvature tensor analysis. *Computer Aided Design 37*, 10 (2005), 975–987. 2

[LHMR08] LAI Y.-K., HU S.-M., MARTIN R. R., ROSIN P. L.: Fast mesh segmentation using random walks. In *ACM Symposium on Solid and Physical Modeling* (2008), pp. 183–191. 2, 7

[LLS*05] LEE Y., LEE S., SHAMIR A., COHEN-OR D., SEIDEL H.-P.: Mesh scissoring with minima rule and part salience. *Computer Aided Geometric Design 22*, 5 (2005), 444–465. 2, 4, 6, 10

[LW08] LAVOUÉ G., WOLF C.: Markov random fields for improving 3d mesh analysis and segmentation. In *Eurographics Workshop on 3D Object Retrieval* (2008). 2, 3

[MFM04] MARTIN D. R., FOWLKES C. C., MALIK J.: Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 26*, 5 (2004), 530–549. 3

[Sha08] SHAMIR A.: A survey on mesh segmentation techniques. *Computer Graphics Forum 27*, 6 (2008), 1539–1556. 2

[SNKS09] SIMARI P., NOWROUZEZAHRAI D., KALOGERAKIS E., SINGH K.: Multi-objective shape segmentation and labeling. *Computer Graphics Forum 28*, 5 (2009), 1415–1425. 2

[SSCO08] SHAPIRA L., SHAMIR A., COHEN-OR D.: Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer 24*, 4 (2008), 249–259. 2, 3, 7, 8

[SWRC09] SHOTTON J., WINN J., ROTHER C., CRIMINISI A.: Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *Intl. Journal of Computer Vision 81*, 1 (2009). 3

[TVD07] TIERNY J., VANDEBORRE J.-P., DAOUDI M.: Topology driven 3D mesh hierarchical segmentation. In *IEEE International Conference on Shape Modeling and Applications (SMI)* (2007). 2

[ZTS02] ZUCKERBERGER E., TAL A., SHLAFMAN S.: Polyhedral surface decomposition with applications. *Computers and Graphics 26*, 5 (2002), 733–743. 2

[ZvKD07] ZHANG H., VAN KAICK O., DYER R.: Spectral methods for mesh processing and analysis. In *Eurographics State-of-the-art Report* (2007), pp. 1–22. 2