Hierarchical mesh segmentation based on fitting primitives

Marco Attene, Bianca Falcidieno and Michela Spagnuolo
IMATI-GE / CNR

Key-words: clustering, denoising, sharp feature, shape abstraction, reverse engineering

Abstract
In this paper we describe a hierarchical face clustering algorithm for triangle meshes based on fitting primitives belonging to an arbitrary set. The method proposed is completely automatic, and generates a binary tree of clusters, each of which fitted by one of the primitives employed. Initially, each triangle represents a single cluster; at every iteration, all the pairs of adjacent clusters are considered, and the one that can be better approximated by one of the primitives forms a new single cluster. The approximation error is evaluated using the same metric for all the primitives, so that it makes sense to choose which is the *most suitable* primitive to approximate the set of triangles in a cluster.
Based on this approach, we implemented a prototype which uses planes, spheres and cylinders, and have experimented that for meshes made of 100k faces, the whole binary tree of clusters can be built in about 8 seconds on a standard PC.
The framework here described has natural application in reverse engineering processes, but it has been also tested for surface de-nosing, feature recovery and character skinning.

1. Introduction

Modern industry is spending more and more efforts in the exploitation of 3D acquisition devices for diverse applications, including reverse engineering and medical imaging, while producers of precise acquisition tools such as recent laser scanners are focusing on improving the quality and flexibility of their products. As a consequence, we can now deal with huge volumes of rather precise data representing 3D shapes. In many contexts, however, it is important to *understand* the data acquired in order to fully exploit its potential. Unfortunately, in most cases the process of associating high level information to raw 3D data is hard to automate, and a time-consuming manual annotation work is required. In a reverse engineering scenario, for example, the user is typically required to track lines on the surface to subdivide it into simple regions which will be eventually approximated by fitting primitives. Sometimes such features can be detected automatically or semi-automatically [21][26], but in most cases the user is required to manually post-process the results to fill gaps or track features missed by the algorithm [29]. In the case of *free-form* 3D models the approximating primitives are typically NURBs whose parameters are determined through an automatic fitting procedure. When the model is built by a given set of primitives, these primitives are typically fitted to the data. Models belonging to this latter class are also referred to as *regular models* [28] and include CSG generated shapes, assemblies of physical primitive objects, and so on.
While a person looking at a rendered 3D model can easily percieve a decomposition of the surface into *interesting* sub-parts such as primitive shapes, it turns out to be a rather hard task for a computer and forms the base of several recent research works categorized under the term *surface segmentation*. This field of investigation is becoming more and more important as a support to methods for *shape reasoning* and *understanding* [2][4], in which the identification of a high-level structure, or signature, is often required to perform the difficult task of *abstracting* a class of similar shapes.
Broadly speaking, segmentation algorithms attempt to exploit geometric information to infer a decomposition of the surface that corresponds to the one that experts would produce manually.

The concept of *good* segmentation, however, strongly depends on the context in which it will be used. In the broadest scenario of supporting the process of shape understanding, methods based on intrinsic characteristics of the shape, such as curvature and topology, tend to group together parts of the model having a kind of coherence and/or uniformity. In [2] and [4], for example, Morse theory is used to partition a 2-manifold in regions having a prescibed topology; once such a partitioning has been determined, regions are connected together to form an Extended Reeb Graph structure representing the *signature* of the shape. The identification of regions of constant curvature has been addressed in [16], where a multi-scale method to evaluate the surface curvature has been introduced. Being a multi-scale approach, the algorithm produces several segmentations (one for each scale), so that a more complete interpretation of the shape is supported.
In some particular contexts, such as the one addresed in this article, more specific and effective methods may be used which exploit an a-priori knowledge of the shape. We tackle the problem of decomposing a triangulated surface into areas with prescribed characteristics. Specifically, we present a framework to compute a hierarchical segmentation of a given shape into connected regions approximated by primitives belonging to a given set. The set of primitives to be used is arbitrary, and does not influence the validity of the framework. The reminder of the paper is organized as follows: In section 2, previous work on mesh segmentation is classified and discussed. In section 3, the hierarchical face clustering method, which is the basis of our work, is described. Section 4 provides an overview of our framework, while in section 5 the computation of the parameters of some fitting primitives is explained. Besides the mathematical foundations described in section 5, we provide some suggestions for an efficient implementation in section 6. Finally, section 7 reviews some application contexts in which our framework is particularly useful, and we conclude the paper in section 8 by discussing potential improvements of the method, extensions and future research directions.

2. Prior Art on Mesh Segmentation

In the more specific context of retrieving the underlying structure of *regular models*, which is the field of investigation of this paper, we can identify segmentation algorithms within two main classes:

- *Feature-Based Detection* – Methods belonging to this class try to follow the same path of the human expert, and thus attempt to determine surface regions indirectly first by computing a set of feature lines and then by splitting the surface through a network of such features.
- *Direct Region Detection* – Methods of this type group faces and vertices into regions through an estimation of approximating primitives. Some approaches grow the regions starting from seed faces, while some others follow a top-down paradigm and split big regions into smaller ones that can be better approximated by the primitives employed.

Both the algorithm classes have their advantages and drawbacks, but none is sufficiently accurate in general and it is often necessary to refine the automatic process by a human intervention to clean the results.

Feature-Based Segmentation

The main difficulty in this class of algorithms is the automatic detection of feature lines. When the model is a height-field, for example, lines of discontinuity are used to define the so-called *surface primal sketch* [18]. In the more general case of triangulated surfaces, typical methods are based on *fold* detection, that is, regions of the surface having a high principal curvature are used to extract feature lines. If, for example, the maximum of the two principal curvatures exceeds a prescribed threshold, the vertex is probably part of a fold [32]. In [22] morphological operators adapted to triangle meshes are used to detect the presence of folds, while in the different setting described in [30] curvature extrema are computed to identify

perceptually salient surface regions which are then turned into feature lines through a skeletonization procedure. When feature lines are not actually present in the model because they have been *chamfered* by the sampling process, they can be reconstructed starting from the neighboring smooth regions and tagged for further processing [3]. In a different setting, the features may be not completely sharp, and surface primitives may be separated by *blended* edges. In these cases, a feature sensitive metric may be used as described in [19].
For the purpose of segmentation, however, all the above mentioned methods are not generally sufficient as they typically produce gaps in the boundaries of the regions, and such a sparsity causes serious difficulties when determining fairly bounded regions.

Direct Segmentation

Instead of deriving the regions starting from their boundary, direct methods go straight to the identification of the regions as sets of adjacent faces or neighboring points. In this context, several approaches are based on *region growing*; starting from few seed points, either randomly selected or determined starting from some geometrical criteria, each region is constructed by expanding the corresponding initial seed [24][7][23]. When simple primitives such as planes are required to approximate the regions, the method proposed in [7] provides an extremely efficient segmentation using Lloyd's partitioning approach [13].
After a first coarse segmentation based on a feature-based approach, Várady et al. [28] classify each region as *simple* or *multiple*, depending on whether the region can be effectively approximated by one of the primitives employed or not. In the latter case, each multiple region is analyzed and partitioned through dimensionality filtering on the Gaussian sphere.
The main difficulty in region growing approaches is the choice of the seed points and, in particular, of their number. Even if one assumes that the number of regions is known, however, region growing approaches may be easily trapped in local minima, and heuristic techniques are required to escape from such situations [7].
The hierarchical segmentation introduced in this paper does not require any seed element to start from, it is extremely efficient and produces a hierarchy of clusters without requiring any parameter to be set by the user.

3. Hierarchical face clustering

The algorithm proposed in this paper is a variation of the hierarchical face clustering (HFC) method described in [8]. The basic idea in the HFC approach is to merge neighboring triangles into representative *clusters*. Here a cluster is a connected set of triangles, not necessarily simply connected, which can be approximated by a simple primitive. In [8], for example, clusters are approximated by fitting planes computed through principal component analysis [11], and the *cost* of merging a set of triangles into a single representative cluster is the integral L^2 distance of its vertices from the fitting plane. The method produces a hierarchy which can be represented by a binary tree of clusters.
To describe the main points of the HFC it is convenient to fix some notation rules.
Let $M=(V,E,T)$ be a manifold triangle mesh, possibly with boundary. The dual graph $D=(C,A)$ of M is defined as follows: each node of C corresponds to a triangle of T, and there is an arc (dual edge) in A connecting two nodes in C if the corresponding triangles in M share an edge. Now, if one considers each node of such a dual graph to represent a cluster (initially made of a single triangle), merging two triangles into a single representative cluster corresponds to contracting a dual edge into a single node, that is, the two nodes of the arc are identified and the adjacency relations are updated accordingly (see Figure 1). In the HFC approach a priority queue is created in which all the dual edges are sorted based on the *cost* of their contraction. At each step, the dual edge with lowest cost is popped from the queue, it is contracted, and all the edges incident to the new representative node are updated, that is, their cost is re-computed

and their position in the queue is updated according to the new cost. Using the cost described in [8] produces a hierarchy of clusters which can be efficiently approximated by planes.

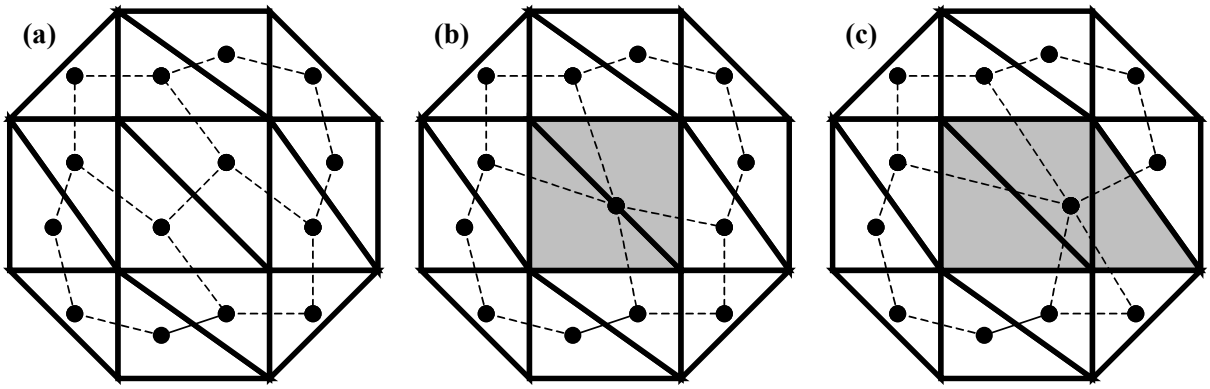

Figure 1: *In (a) a triangle mesh and the corresponding dual graph are depicted. In (b) an arc of the dual graph has been contracted, and the two triangles corresponding to the dual arc's end-points have been marked as belonging to the same single cluster. In (c) another arc has been contracted producing a resulting cluster made of three triangles.*

4. Overview of the segmentation framework

Our segmentation algorithm is based on a variation of the HFC approach. In our framework, the type of primitive to be fitted to the triangles of a cluster is picked from a given finite set. Specifically, for each primitive type the corresponding fitting parameters are computed and the approximation error evaluated. The cost of merging a set of triangles into a single representative cluster is the minimum of the approximation errors computed against the primitives. If, for example, the algorithm is required to fit planes and spheres, for each potential contraction it is necessary to compute 1) the coefficients of the best fitting plane and 2) the coefficients of the best fitting sphere; then, the approximation error is evaluated against both the plane and the sphere, and the minimum one is considered for re-ordering the queue. In Figure 2 two examples of clustering are shown in which planes, spheres and cylinders have been used.

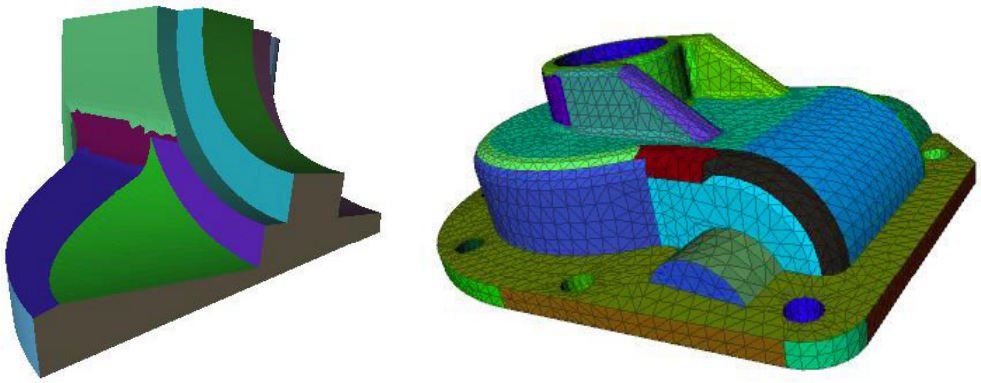

Figure 2: *Example of clustering of two models in regions fitted by planes, spheres and cylinders. The models shown on the left and on the right have been segmented using 21 and 45 clusters respectively.*

Summarizing, the sturcture of the algorithm is the same as in the HFC approach, that is, at each step a dual edge is popped from the priority queue, it is contracted, and the queue is updated consequently. What changes is the *cost* assigned to each dual edge, which is computed by simulating the contraction and by computing the error against each of the fitting primitives employed to approximate the resulting cluster.

The evaluation of the approximation error is a delicate issue, and somehow it depends on which assumptions are made about the triangle mesh being analyzed. If the mesh has been directly obtained through interpolation of a set of scattered points, for example, one can

reasonably assume that the only reliable information is the position of the vertices. In such a case an L^2 error may be simply computed as the sum of the squared distances of the vertices from the fitting primitives [8]. If, on the other hand, the mesh is the result of a simplification process, the triangles of the mesh may incorporate part of the geometric information of the removed vertices. In this latter case, it might be more appropriate to integrate the squared distances over the whole surface [7].
In the framework described in this paper, however, deriving closed-form expressions for integral squared distances might not be easy for certain kinds of primitives, and relying on numerical integration becomes prohibitive in terms of computational costs. Thus, if one wants to take into account the geometric information carried by triangles, the algorithm provides the possibility to *weight* the distances computed at vertices. Specifically, for each vertex v a *restricted Voronoi area* $a(v)$ is computed which corresponds to a third of the total area of the triangles incident at v [15] and belonging to the cluster under consideration. The integral L^2 error is then approximated by the weighted sum of the squared distances of the vertices from the fitting primitive. Notice that such an approximation is nothing but a numerical integration in which the domain is discretized at vertices.

5. Fitting Primitives

In this section we describe how to compute the parameters of a small family of primitives (planes, spheres and cylinders) that we used to implement and test our segmentation framework.

Fitting Planes

To compute the best fitting plane to a set of triangles we make use of a classical method based on Principal Component Analysis [8][7]. Specifically, we compute the (possibly weighted) covariance matrix Cov_v of the vertices of the cluster:

$$Cov_v = \sum_i a(v_i)(v_i - \bar{v})(v_i - \bar{v})^T \ , \ \bar{v} = \frac{\sum_i a(v_i) v_i}{\sum_i a(v_i)}$$

where v_i is a vertex and $a(v_i)$ is the restricted Voronoi area of v_i. If a non-integral error evaluation is required, $a(v_i)$ can be set to 1 for all the vertices.
The best-fitting plane passes through $\bar{v}$, and its normal n is the eigenvector corresponding to the minimum eigenvalue of Cov_v. The L^2 fitting error can be computed through:

$$L^2 = \sum_{i=1}^{n} a(v_i)\big(n(v_i - \bar{v})\big)^2$$

Fitting Spheres

The determination of the parameters (center and radius) of the least-squares best fitting sphere to a set of 3D points can be formalized as follows:
Let P be a set of n points (x_i, y_i, z_i), and let $(x-c_x)^2 + (y-c_y)^2 + (z-c_z)^2 - r^2 = 0$ be the implicit equation of the sphere S of radius r centered at $c=(c_x,c_y,c_z)$. The squared Euclidean distance of a point $p_i=(x_i, y_i, z_i)$ from that sphere is:

$$d^2(p_i, S) = \left(\sqrt{(x_i - c_x)^2 + (y_i - c_y)^2 + (z_i - c_z)^2} - r\right)^2$$

In the least-squares sense, computing the best fitting sphere to the set of points P amounts to determining the center (c_x,c_y,c_z) and the radius r such that the sum of all the squared distances is minimized, that is:

$$\min\left(\sum_{i=1}^{n} d^2(p_i, S)\right)$$

The above problem is non-linear and may be solved using the Gauss-Newton method [25]. In our framework, unfortunately, using such a technique to compute the exact optimum becomes prohibitive in terms of computational cost, so we simply strive to find a good fitting using the concept of *algebraic distance* [20]. To do this, the implicit equation of the sphere may be re-written as:

$$x^2 + y^2 + z^2 + c_x{}^2 + c_y{}^2 + c_z{}^2 - 2c_x x - 2c_y y - 2c_z z - r^2 = 0$$

which, in vector form, is equivalent to:

$$\begin{bmatrix} 2x & 2y & 2z & 1 \end{bmatrix} \begin{bmatrix} c_x \\ c_y \\ c_z \\ r^2 - c_x{}^2 - c_y{}^2 - c_z{}^2 \end{bmatrix} = x^2 + y^2 + z^2$$

where the four unknowns c_x, c_y, c_z and r have been isolated in a single vector. Substituting x, y and z with the coordinates of the points in P we obtain the following over-determined linear system:

$$Aw = b \text{ where } A = \begin{bmatrix} 2x_1 & 2y_1 & 2z_1 & 1 \\ 2x_2 & 2y_2 & 2z_2 & 1 \\ & \vdots & & \\ 2x_n & 2y_n & 2z_n & 1 \end{bmatrix}, \; b = \begin{bmatrix} x_1{}^2 + y_1{}^2 + z_1{}^2 \\ x_2{}^2 + y_2{}^2 + z_2{}^2 \\ \vdots \\ x_n{}^2 + y_n{}^2 + z_n{}^2 \end{bmatrix} \text{ and}$$

$$w = \begin{bmatrix} c_x \\ c_y \\ c_z \\ r^2 - c_x{}^2 - c_y{}^2 - c_z{}^2 \end{bmatrix}$$

which can be solved in the least-squares sense by computing $w = (A^T A)^{-1} A^T b$.

To accommodate this framework to the case of weighted points, we simply multiply each equation of the system by the corresponding weight.

Notice that if all of the points lie exactly on a sphere, then the solution of the above system represents exactly that sphere. In the other cases where the residue is not null, the sphere represented by w is sufficiently close to the one computed through iterative approaches such as the Gauss-Newton method or other sorts of non-linear regression [10]. We have experienced that such a level of precision is enough for the purpose of mesh segmentation, where the user is typically interested in few clusters made of numerous faces. In other application contexts, however, it may be necessary to obtain a more precise fitting; in such cases the approximation described above can be used to initiate a regression.

We have also experimented the direct fitting method described in [20], in which the sphere's equation is re-written as:

$$A(x^2 + y^2 + z^2) + Bx + Cy + Dz + E = 0$$

and the minimizing vector (A,B,C,D,E) is found under the constraint:

$$B^2 + C^2 + D^2 - 4AE = 1$$

Such a method is significantly more precise when the number of points is limited, while it is virtually equivalent to the non-constrained version (with A=1) when such a number grows. On the other hand, this method requires an additional equation representing the constraint and, by using Lagrange multipliers, finding the minimum amounts to solving a 5x5 eigensystem, which is slower than inverting a 4x4 matrix. In many application contexts, including mesh segmentation, the user is typically interested in the approximation of dense models with few primitives, hence each primitive usually fits numerous points; In these cases the results of the

non-constrained method are precise enough and the increase of computing time due to the solution of the eigensystem is not justified.
Having established the parameters c and r of the sphere, the L^2 fitting error can be computed through:

$$L^2 = \sum_{i=1}^{n} a(v_i)\left(\|v_i - c\|_2 - r\right)^2$$

Fitting Cylinders

We found it convenient to represent each approximating cylinder through its radius, a unit vector parallel to its axis, and a point belonging to the axis that we call the *center* of the cylinder.
Given a connected set of triangles (from now on, the *cluster*), the problem can be summarized as finding the parameters of a cylinder that fairly approximates that set. This problem can be tackled by first computing the direction of the cylinder axis starting from the triangle normals. Roughly speaking, we note that the normal field of the cluster is similar to the one of an ideal cylinder if the *normal variation* is roughly null in one direction and maximal and roughly constant in all the orthogonal directions computed at each point of the cluster. Since the cluster is piecewise-linear, the only points at which the normals change are points on edges, and the normal variation can be integrated over the whole cluster and represented in matrix form as:

$$Cov_c = \sum_i |e_i| \beta(e_i) \bar{e}_i \bar{e}_i^T$$

where e_i is an internal edge of the cluster, $|e_i|$ is its length, $\bar{e}_i$ is a unit vector parallel to e_i and $\beta(e_i)$ is the signed angle between the normals of the two triangles sharing e_i (positive if convex, negative if concave). Notice that when a cluster is obtained as the intersection of a triangle mesh with a ball of radius B centered at a vertex v, Cov_c can be divided by the area of the cluster and the result is the curvature tensor at v as defined in [1], as long as the cluster is homeomorphic to a disk. The direction n of minimal normal variation of the cluster is identified by the eigenvector corresponding to the maximum eigenvalue of the symmetric and positive semi-definite matrix Cov_c.
Having established the direction n of the fitting cylinder, the remaining parameters to be determined are its radius and its center. We note that this operation can be reduced to the bi-dimensional best-fitting circle problem by projecting the vertices on the plane having n as normal and passing through the center of mass of the cluster. Specifically, let c_m be the center of mass, we compute an orthonormal basis $<n, e_x, e_y>$ (e_x and e_y may be the remaining two eigenvectors of Cov_c or any other pair of orthogonal vectors which are also orthogonal to n) and transform each vertex v_i to $\wp(v_i) = <(v_i-c_m)e_x, (v_i-c_m)e_y>$. The center $(\hat{c}_x, \hat{c}_y)$ of the 2D best fitting circle to such a set of transformed points is then transformed back to the original coordinate system, that is, $c = c_m + e_x \hat{c}_x + e_y \hat{c}_y$.
The parameters of the 2D fitting circle can be calculated as described previously for fitting spheres by simply reducing the dimensionality of the problem.
Having established the parameters n, c and r of the cylinder, the L^2 fitting error can be computed through:

$$L^2 = \sum_{i=1}^{n} a(v_i)\left(\|(v_i - c) \times n\|_2 - r\right)^2$$

where $\times$ denotes the cross product.
In the context of fitting cylinders to point clouds, significant previous results have been obtained by Várady et al. [28] and by Chaperon and Goulette [5]. Both these works make use of the Gaussian image of estimated surface normals. Specifically, they observe that the normals of a cylinder generate a circle on the Gaussian sphere, and thus look for such circles

to estimate the axis of the fitting cylinder. The direction of such an axis is determined as the normal of a plane passing through the origin and such that the sum of squared distances to the points in the Gaussian sphere is minimized. This approach works well in the context addressed in [28] and [5], where surface normals are estimated using an arbitrarily large support to cope with noise. This is made possible by the assumption that raw data have been roughly segmented through a prior feature-based method. Within a hierarchical approach such as ours, however, such an assumption does not hold, and a relatively small amount of noise may heavily spoil the results provied by the method used in [28] and [5] (see Figure 3). Conversely, the method based on the analysis of the normal variation described above proven to be rather robust even when the amount of noise grows significantly. Furthermore, it must be considered that in our context the evaluation of noise is more delicate due to the multiresolution nature of the clustering, that is, what is considered *noise* at a given scale may be seen as a *surface feature* at finer resolutions.

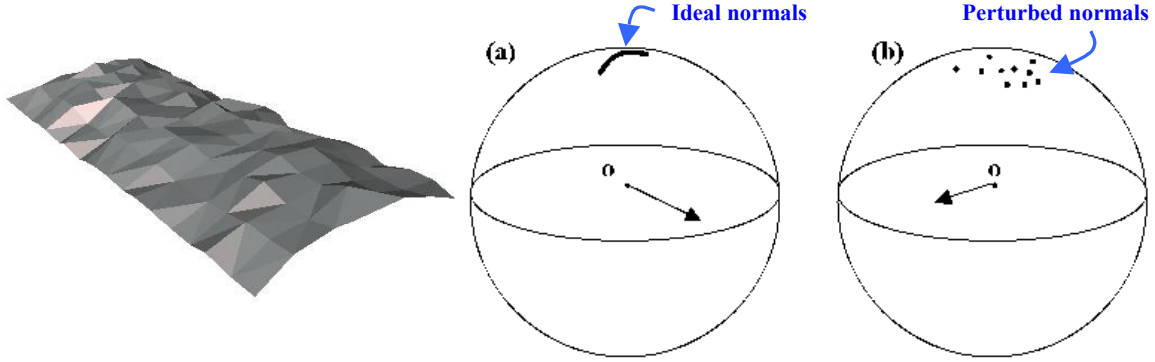

Figure 3: *The nearly cylindrical patch shown on the left has been obtained by perturbing an ideal cylindrical patch with a small amount of noise. The normals of the ideal patch have been mapped on the Gaussian sphere (a), and the best fitting plane passing through the center* ***o*** *correctly identifies the axis of the cylinder. The perturbed patch has been mapped too (b) but, in this case, the normal of the best fitting plane is nearly orthogonal to the cylinder's axis.*

6. Implementation details

At each step of the algorithm a dual edge is collapsed into a single representative node corresponding to a new cluster that, in its turn, corresponds to a fitting primitive. The costs of all the dual edges incident to the new node must be updated. Thus, for each of them new fitting primitives must be estimated that approximate the set of all the triangles belonging to the two clusters, and the approximation errors must be re-computed based on these new primitives. Without particular attention the computational cost of these operations can easily become prohibitive, so it is important to incrementally compute most of the entities (i.e. the matrices Cov_c, A^TA, A^Tb, …) and store partial results within the data structure. Moreover, the tree of clusters may become unnecessarily unbalanced and cause a huge number of unused updates in the queue. For example, in a mesh which is perfectly flat the error is null (or nearly null due to numerical imprecision) for each dual contraction. In such a case it may happen that a single cluster grows unnecessarily and forms a dual node having an extreme degree. In our implementation we used a threshold value[1] below which the error is considered numerical noise and thus is snapped to zero. During the re-ordering of the queue, if two edges have both a null cost then, instead of choosing one of them randomly, we give priority to the one whose contraction generates the node with lowest degree.

[1] We found that on the Linux-i686 system that we used for testing, a threshold of $1.0*10^{-15}$ is a good choice when using integral L^2 error metrics. It must be considered, however, that a different system may have a different robustness.

On a P4 1.7GHz machine equipped with 512Mb of RAM and running Linux, our implementation is able to build the complete binary tree of clusters of a triangle mesh made of 100K faces in about 8 seconds.

7. Applications

Reverse Engineering

The most natural applications of our framework are to be searched in reverse engineering [27]. When a product is digitized to be re-used within a CAD environment, it is often necessary to recover how the original model was designed. In this context, most of the previously published work make extensive use of heuristics and/or requires the manual selection of a number of parameters whose effect is not always clear to the user. In contrast, our framework is completely automatic and, at the same time, customizable. If the model is known to be made of a well defined set of primitives, as typical, for example, for mechanical objects, the algorithm may accept a *plug-in* for each of them in which the computation of the fitting parameters and the error are implemented. Moreover, being a greedy method, the level of accuracy is somehow reflected by the cluster hierarchy which, once computed, may be interactively navigated by the user through a *slider* which sets the desired number of clusters or a threshold error (see Figure 4 and Figure 11).

Note that, by definition, at the beginning each triangle constitutes a single planar cluster, thus each cluster is best fitted by a plane. Differently from other methods [9], however, in our approach clusters of different type may be straightforwardly aggregated to form a bigger cluster; the type of such a new big cluster is independent of the type of the constituting ones, while it is determined exclusively based on which primitive best fits all its triangles. This makes our method particularly robust in case of noisy input meshes, in which at early stages small clusters may happen to be of the "wrong" type (see Figure 4, d) due to the noise but, as the clustering proceeds, their type rapidly converges to the expected one (Figure 4, c). Clearly, when too few clusters remain, the corresponding fitting primitives can no longer approximate the shape in a fair way (Figure 4, a and b), and deciding how many clusters are necessary for such a fair approximation is a matter of threshold errors which, for now, must be defined by the user based on his/her knowledge.

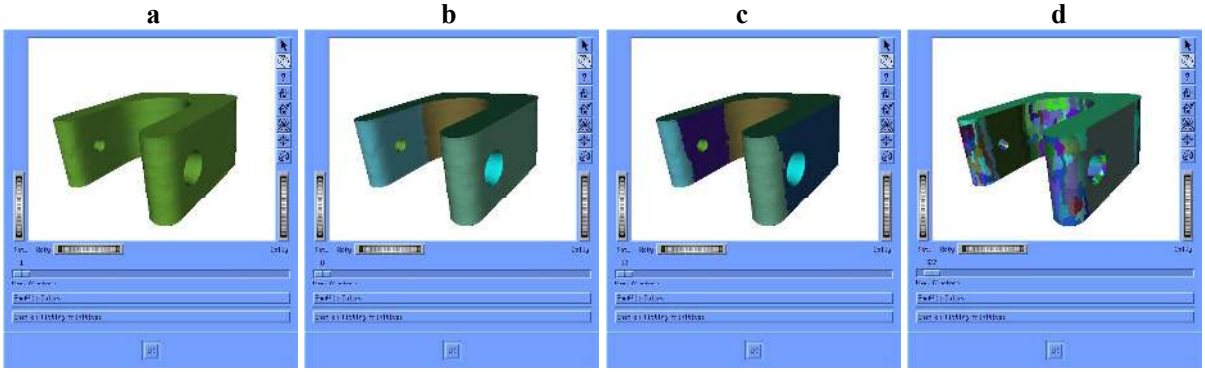

Figure 4: *Example of interactive navigation of the cluster tree through the slider labeled "Num. Clusters".*

Although this procedure does not compute **the** correct segmentation automatically, the possibility to browse the hierarchy at interactive speed is a powerful support to the user that, based on the knowledge that the error grows proportionally to the number of clusters, may easily locate the required level in the hierarchy as shown in Figure 4, c.

Mesh denoising and fairing

To improve the quality of a triangle mesh obtained through non-contact acquisition devices such as laser scanners, one has to deal with two main problems: 1) the noise in the data and 2)

the lack of information about sharp features. If the model is known to be made of a well defined set of primitives, each vertex of the mesh may be snapped on the primitive fitting the corresponding cluster; if the vertex belongs to the boundary of a cluster, then it may be snapped once for each of the clusters incident to the vertex. Such a procedure has the twofold effect of distributing the error and of sharpening the chamfered features (see Figure 5).

Figure 5: *Example of a resampled model in which the sharp edges have been chamfered (on the left). On the right, the same model is shown after the de-noising process based on a segmentation in 12 regions.*

To have a quantitative point of view, we have resampled some original meshes by intersecting a uniform grid with the surfaces. The resulting re-meshes have been perturbed with variable amounts of noise; finally, for each of them, we have computed the L^2 distortion against the corresponding original meshes. We have observed that using the de-noising procedure described above reduces the L^2 distortion up to one order of magnitude (see Figure 6).

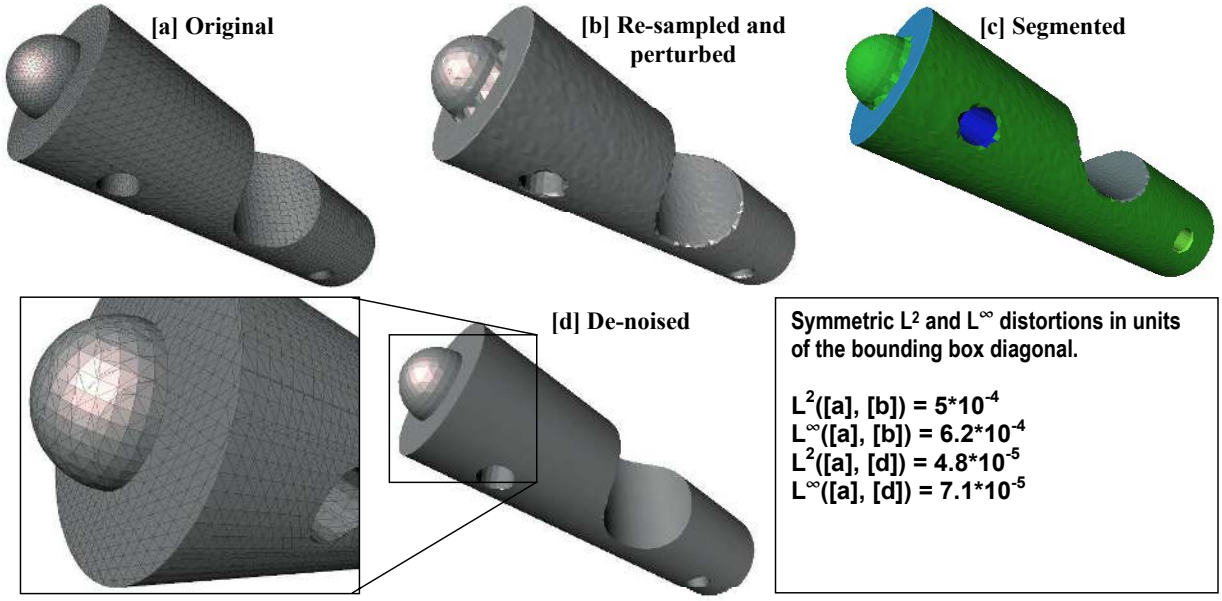

Figure 6: *An example showing the decrease of distortion obtained thourgh our de-noising procedure. The error values reported in the box have been computed through the publicly available Metro tool [6].*

Automatic segmentation and skinning for character animation

Natural shapes, and in particular humans and animals, form a class of 3D models whose shape can be effectively abtracted through approximating cylinders. If only this kind of primitive is

used, the clustering algorithm described so far represents an automatic way to derive a hierarchical view of the shape (see Figure 7) as described by Marr in [14].

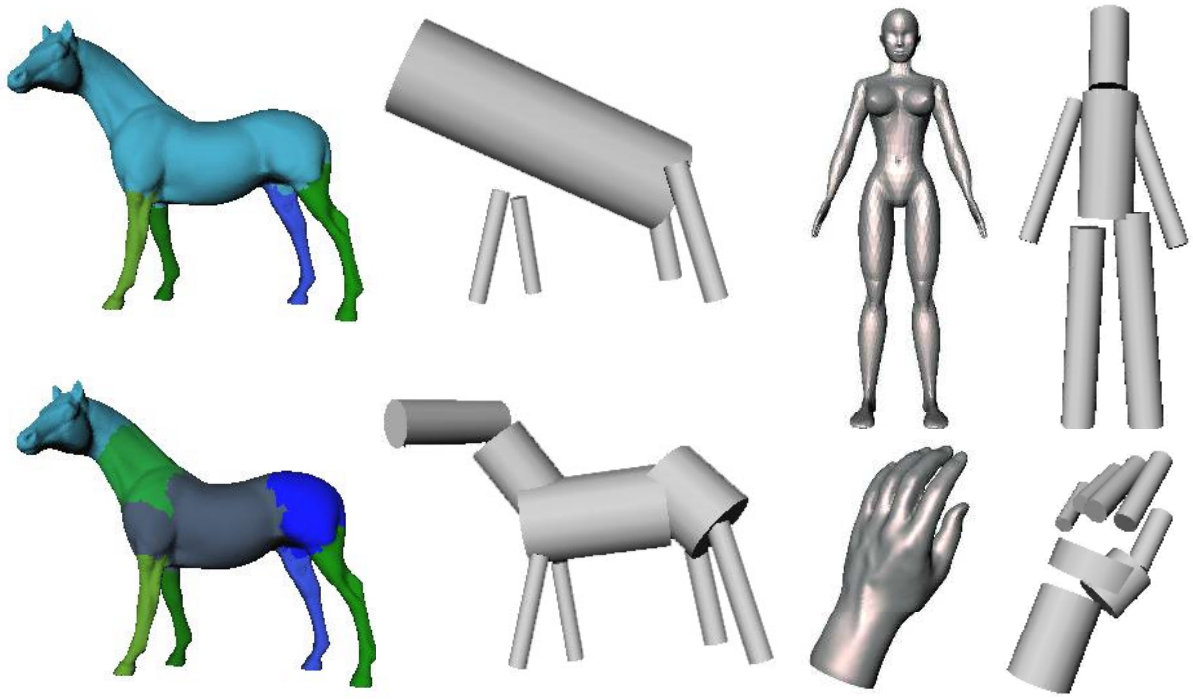

Figure 7: *Some models along with the corresponding abstractions represented by the fitting cylinders. For the horse model, two levels of resolution are shown along with the corresponding face clustering.*

In this context, our framework becomes particularly useful as a support for skinning purposes to eventually deform or animate such a class of triangle meshes in an intuitive and user friendly manner. Once the cluster hierarchy has been computed, in fact, it may be interactively navigated by the user through a *slider* which sets the desired number of clusters. By rendering triangles using a unique color for each cluster, the user has a visual feedback of the current mesh partitioning. When settled, the user may require the automatic creation of a deformation skeleton made of bones and joints, and use it to interactively deform the segmented mesh (see Figure 8).

We have implemented a minimal interactive GUI which allows the user to perform such a kind of editing, and briefly sketch here the foundations of the method.

Let *D* be the graph in which the clusters are nodes, and arcs join adjacent clusters (i.e. the dual graph defined in section 3). The combinatorial structure of the initial skeleton S = (J,B) is defined as the dual graph of D, that is, each joint in J corresponds to an arc of D, and each bone in B corresponds to a cluster of D. If a cluster is adjacent to only one other cluster, we add a *virtual* joint representing the other end-point of the bone. Clearly, such a skeleton may be rather complex when the number of clusters is high, and many bones and joints do not correspond to an intuitive notion of mesh part to be deformed. Hence it is convenient to tag skeleton elements so that non-intuitive bones and joints are not shown. We chose to classify joints into two categories: *sensitive* joints and *locked* joints. Sensitive joints may be moved by the user, and correspond to joints having at most two incident bones; all the other joints are locked, are rendered differently and cannot be moved. Moreover, to simplify the interactive environment in which the user acts, we chose to hide all the bones that cannot be moved, that is, the bones connecting two locked joints; finally, if all the bones incident at a joint are hidden, then that joint is hidden as well.

Once the skeleton is available, vertices can be attached to *influencing joints*, and their position after a deformation can be computed as described in [12].

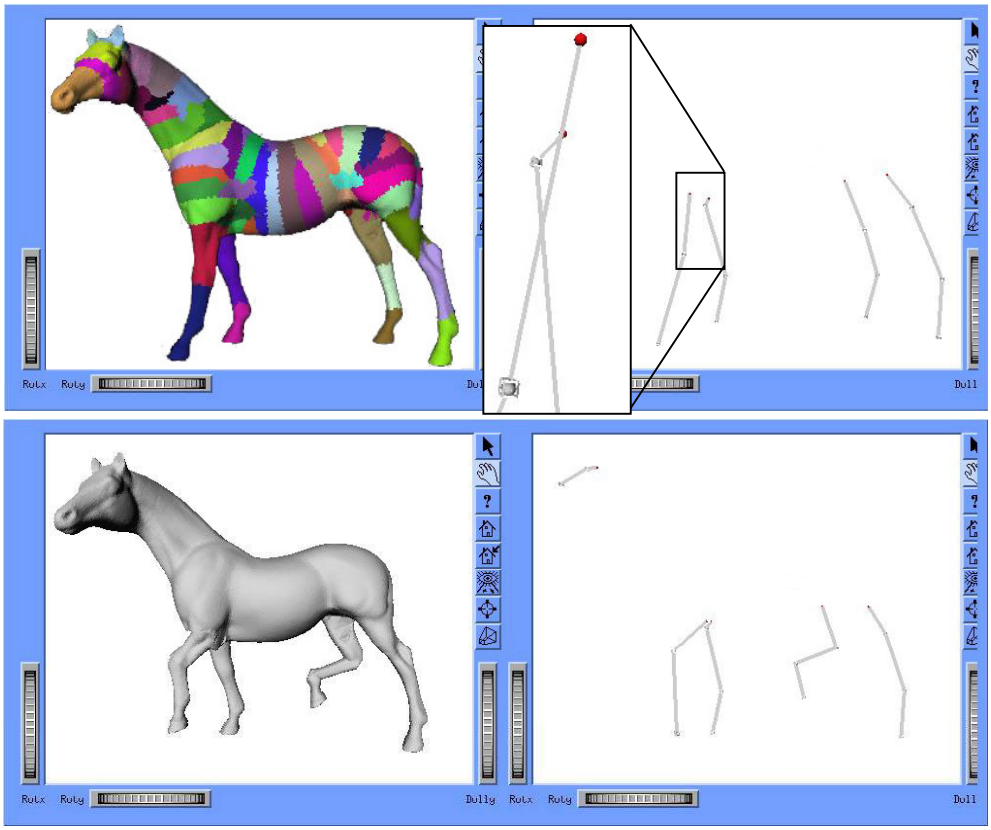

Figure 8: *The model of a horse has been segmented in 87 clusters out of which the deformation skeleton has been computed (top row). By dragging the sensitive nodes the geometry of the model has been interactively modified.*

8. Discussion

The hierarchical segmentation presented in this paper is fast (see Table 1), robust to noise, and suitable to produce useful segmentations of irregularly sampled models (Figure 9). Due to its greedy nature, however, it is worth to discuss how it behaves when compared to variational approaches.

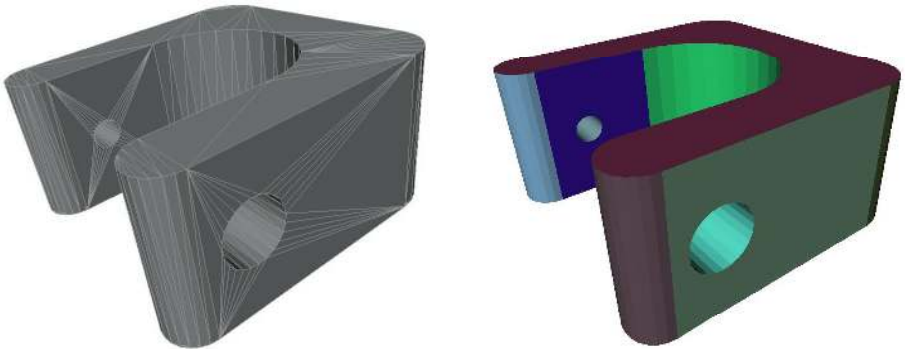

Figure 9: *Segmentation of a model with extremely non-uniform vertex distribution.*

We have experimented a variational version of the method that, starting from a given clustering resolution, attempts to improve the quality of the segmentation using Lloyd's partitioning approach [13], as previously done for fitting planes in [7]. We have concluded, however, that in practical cases the level of improvement is far too limited to justify such an expensive post-processing. In our experiments, in fact, the variational approach required nearly 20 times longer than the corresponding greedy version with comparable results. A method using such a variational approach is described in a very recent work by Wu and Kobbelt [31], where the authors report nearly three minutes to process models up to 100k faces.

Model name	Number of Triangles	Computing Seconds
Fandisk (Fig. 2, left)	12946	0.64
Casting (Fig.2, right)	10224	0.66
Drill (Fig. 4)	14090	1.29
Mechanical Part (Fig. 11)	24822	1.88
Bunny (Fig. 10)	69449	7.01
Fandisk remeshed 1	101207	8.51
Fandisk remeshed 2	207136	22.15

Table 1: *Time required to compute the whole hierarchy for some of the models presented in this paper. Time is relative to our prototype running on a P4 1.7 GHz, 512 Mb RAM and running Linux.*

There are cases, however, in which the **globally best fitting** *n* primitives need to be found for an object; in these cases, a greedy hierarchical approach such as ours is not appropriate, and well-designed variational methods provide better results (see Figure 10, top row). To show a qualitative comparison of our greedy method with the variational version of [31], we extracted 31 clusters out of the hierarchy computed for the *bunny* model (Figure 10, bottom row) and applied the same *vertex to proxy projection* described by Wu and Kobbelt to generate the stylized bunny depicted on the right.

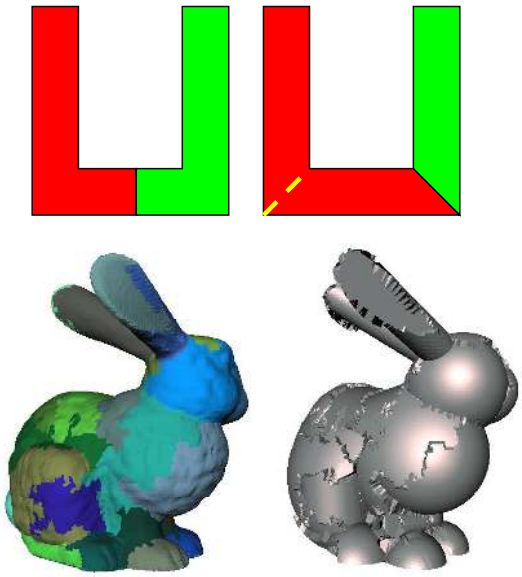

Figure 10: *Top row: The same 2D shape segmented using two fitting rectangles by a variational approach (left) and by a greedy hierarchical method such as ours (right). The dashed yellow line indicates where the red cluster would need to be cut to obtain a segmentation in three clusters. Bottom row: The bunny model was clustered (left) using 31 fitting primitives; each vertex was then snapped to the closest point (right) of the corresponding primitive.*

9. Conclusions and future research

The segmentation framework described in this paper represents a flexible and completely automatic way to partition the surface in a hierarchical manner. Being a greedy approach, there is no issue related to the choice of seed points, and the number of regions can be interactively selected by the user once the hierarchy is computed. No network of features is required for the segmentation, thus all of the problems related to discontinuous feature lines are avoided. Although it has been described for planes, spheres and cylinders, the framework is well-suited for including the definition of several other primitives to be fitted such as tori, cones, or geons, to cite a few. Various fitting primitives are well discussed in the literature [28][17], and we believe that adapting these methods to our hierarchical setting is a rather easy

task. Moreover, it is also easy to adapt them to handle different error metrics such as, for example, the L^{∞}.
All of these aspects form the base for our future research plans, which also include the study of the error evolution along the hierarchy. About this last aspect, we have observed that for regular objects there are some *jumps* in the graph representing the error at various clustering resolutions, and we plan to analyze such discontinuities to automatically detect the *most significant* resolutions (i.e. the number of regions) for a given shape.
It seems also promising to study how such a clustering can be applied in a geometry compression environment. The error-reduction due to the snapping of vertices onto fitting primitives, in fact, is expected to allow coarser quantizations of the geometric information. The larger error introduced may be then reduced to acceptable values through de-noising and feature recovering as described in section 7. Eventually, if necessary, such an approximated geometry may be refined in a lossless fashion by encoding small corrective vectors represented through local coordinate frames relative to the approximating primitives. The compressed representation, however, must necessarily encode the parameters of each primitive, thus keeping the number of primitives low is also important, and finding the best trade-off between this number and the size of the compressed parameterized vertices may be hard, but would promise very interesting compression rates for the class of models addressed.
Finally, although our method is an extension of [8], it is important to note that the two works have applications in different areas. Thus, if one looks for a clustering algorithm to implement efficient collision detection or multiresolution radiosity, the original method described in [8] is sufficient, while the extensions introduced in this paper are necessary to tackle problems such as the ones discussed in section 7.

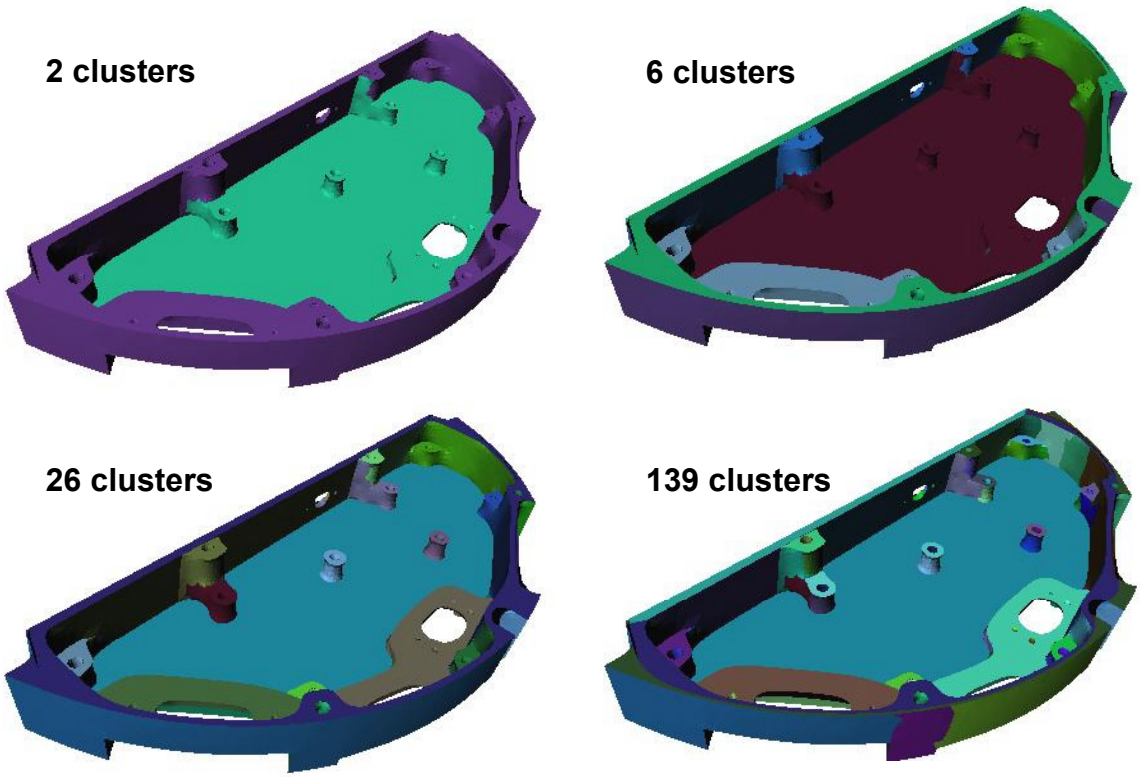

Figure 11: *Four segmentation levels of the same model extracted from the binary tree of clusters. Each level was obtained by specifying the desired number of clusters through an interactive slider.*

Acknowledgements

This work is partially supported by the EU Project AIM@SHAPE (Contract # 506766). Thanks are due to all the members of the Shape Modeling Group of the IMATI-GE/CNR, to the reviewers for their helpful advice, and to all the people that, through discussions and suggestions, made it possible to write this article.

References

[1] Alliez, P., Cohen-Steiner, D., Devillers, O., Levy, B. and Desbrun, M. 2003. *Anisotropic Polygonal Remeshing*. ACM Transactions on Graphics, 22, 3, 485-493.

[2] Attene, M., Biasotti, S. and Spagnuolo, M. 2003. *Shape understanding by contour-driven retiling*. The Visual Computer, 19, 2-3, 127-138.

[3] Attene, M., Falcidieno, B., Rossignac, J. and Spagnuolo, M. 2005. *Sharpen&Bend: Recovering curved sharp edges in triangle meshes produced by feature-insensitive sampling.* IEEE Transactions on Visualization and Computer Graphics, 11, 2, 181-192.

[4] Biasotti, S., Falcidieno, B and Spagnuolo, M. 2004. *Surface Understanding Based on Extended Reeb Graph Representation.* Topological Data Structures for Surfaces: An Introduction to Geographical Information Science. S. Rana Ed., Chapter 6, 87-102.

[5] Chaperon, T. and Goulette, F. 2001. *Extracting Cylinders in full 3D data using a random sampling method and the Gaussian image*. In Proceedings of VMV 2001, Stuttgart, Germany, 35-42.

[6] Cignoni, P., Rocchini, C. and Scopigno, R. 1998. *Metro: measuring error on simplified surfaces*. Computer Graphics Forum 17, 2 (Proceedings of Eurographics '98), 167-174.

[7] Cohen-Steiner, D., Alliez, P. and Desbrun, M. 2004. *Variational Shape Approximation*. ACM Transactions on Graphics, 23, 3, 905-914.

[8] Garland, M., Willmott, A. and Heckbert, P.S. 2001. *Hierarchical face clustering on polygonal surfaces*. In Proceedings of the ACM Symposium on Interactive 3D graphics, 49-58.

[9] Gelfand, N. and Guibas, L., J. 2004. *Shape Segmentation Using Local Slippage Analysis*. In Proc. of Eurographics Symposyum on Geometry Processing, 219-228.

[10] Glantz, S.A. and Slinker, B.K. 2000. *Primer of Applied Regression and Analysis of Variance*. McGraw-Hill Medical, ISBN 0071360867.

[11] Jolliffe, I. T. 1986. *Principal Component Analysis*. Springer-Verlag, New York.

[12] Lewis, J. P., Cordner, M., and Fong, N. 2000. *Pose space deformations: A unified approach to shape interpolation and skeleton-driven deformation*. In Proceedings of ACM SIGGRAPH 2000, 165 - 172.

[13] Lloyd, S. 1982. *Least square quantization on PCM*. IEEE Transactions on Information Theory, 18, 129-137.

[14] Marr, D. 1982. *Vision*. Freeman Publishers.

[15] Meyer, M., Desbrun, M., Schröder, P. and Barr, A.H. 2003. *Discrete Differential Geometry Operators for Triangulated 2-manifolds.* Visualization and Mathematics III, 35-57.

[16] Mortara, M., Patanè, G., Spagnuolo, M., Falcidieno, B. and Rossignac, J. 2003 *Blowing Bubbles for Multi-scale Analysis and Decomposition of Triangle Meshes.* Algoritmica, 38, 2, 227-248.

[17] Petitjean, S. 2002. *A Survey of Methods for Recovering Quardics in Triangle Meshes*. ACM Computing Surveys, 34, 2, 211-262.

[18] Ponce, J. and Brady, J. 1987. *Toward a surface primal sketch*. Three-Dimensional Machine Vision, T. Kanade, Ed. Kluwer Publishers, 195-240.

[19] Pottman, H., Leopoldseder, S., Hofer, M., Steiner, T. and Wang, W. 2004. *Industrial Geometry: Recent Advances and Applications in CAD*. Computer-Aided Design, 1, 513-522.

[20] Pratt, V. 1987. *Direct Least-Squares Fitting of Algebraic Surfaces*. Computer Graphics, 21, 4 (Proceedings of SIGGRAPH'87), 145-152.

[21] Renner, G., Várady, T. and Wiess, V. 1998. *Reverse engineering of free-form features*. In Proceedings of PROLAMAT 98, Trento, IFIP.

[22] Rössl, C., Kobbelt, L. and Seidel, H.-P. 2000. *Extraction of feature lines on triangulated surfaces using morphological operators*. In Proc. Of Smart Graphics'00, AAAI Spring Symposium, 71-75.

[23] Sander, P., Wood, Z., Gortler, S. J., Snyder, J. and Hoppe, H. 2003. *Multi-chart geometry images.* In Proceedings of the Eurographics Symposium on Geometry Processing, 146-155.

[24] Sapidis, N. and Besl, P. 1995. *Direct construction of polynomial surfaces from dense range images through region growing*. ACM Transactions on Graphics, 14, 2, 171-200.

[25] Scales, L. E. 1985. *Introduction to Non-linear Optimization*. Springer-Verlag, New York.

[26] Thompson, W.B., Owen, J.C., James de St. Germain, H., Stark, S.R. and Henderson, T.C. 1999. *Feature-based reverse engineering of mechanical parts*. IEEE Transactions on Robotics and Automation, 15, 1, 57-66.

[27] Várady, T. and Martin, R. 2002. *Reverse Engineering*. In Handbook of Computer Aided Geometric Design, 651-681.

[28] Várady, T., Benkö, P. and Kós, G. 1998. *Reverse Engineering Regular Objects: Simple Segmentation and Surface Fitting Procedures*. International Journal of Shape Modeling, 4, 3, 127-141.

[29] Vergeest, J.S.M., Horváth, I., Kuczogi, G., Opyio, E., Wiegers, T. 1999. *Reverse engineering for shape synthesis in industrial engineering*. In Proceedings of the 26th Int. Conf. of Computers in Industrial Engineering, 84-90.

[30] Watanabe, K. and Belyaev, A.G. 2001. *Detection of Salient Curvature Features on Polygonal Surfaces*. Computer Graphics Forum 20, 3 (Proceedings of Eurographics '01), 385-392.

[31] Wu, J. and Kobbelt, L. 2005. *Structure Recovery via Hybrid Variational Surface Approximation*. Pre-print. To appear in Proceedings of Eurographics'05, Trinity College, Dublin, Ireland, August 29-September 02, 2005.

[32] Yang, M and Lee, E. 1999. *Segmentation of measured point data using a parametric quadric surface approximation*. Computer Aided Design, 31, 7, 449-458.