Volume xx (200y), Number z, pp. 1–14

Hierarchical Structure Recovery of Point-Sampled Surfaces

Marco Attene and Giuseppe Patanè[†]

Abstract

We focus on the class of regular models *defined by Várady* et al. *for reverse engineering purposes. Given a 3D surface $\mathcal{M}$ represented through a dense set of points, we present a novel algorithm that converts $\mathcal{M}$ to a hierarchical representation $\mathcal{H}_{\mathcal{M}}$. In $\mathcal{H}_{\mathcal{M}}$, the surface is encoded through patches of various shape and size, which form a hierarchical atlas. If $\mathcal{M}$ belongs to the class of regular models, then $\mathcal{H}_{\mathcal{M}}$ captures the most significant features of $\mathcal{M}$ at all the levels of detail. In this case, we show that $\mathcal{H}_{\mathcal{M}}$ can be exploited to interactively select regions of interest on $\mathcal{M}$ and intuitively re-design the model. Furthermore, $\mathcal{H}_{\mathcal{M}}$ intrinsically encodes a hierarchy of useful* segmentations *of $\mathcal{M}$. We present a simple though efficient approach to extract and optimise such segmentations, and we show how they can be used to approximate the input point sets through idealised manifold meshes.*

Categories and Subject Descriptors (according to ACM CCS): Hierarchical clustering, segmentation, shape primitives, selection.

1. Introduction

In its earliest years, Computer Graphics was dominated by synthetic shapes, either created by designers through proper software tools or generated out of simulation of physical phenomena. Recent advances in 3D acquisition technologies have brought a gradual change in the way 3D models are produced and handled. Today, precise digitised models are extremely complex, and it is not rare to deal with objects made of millions of elements (e.g., points, triangles, etc). Such a complexity prompted the development of algorithms to partition large models in smaller parts, which are more manageable and significant.

Properly designed models can be easily modified by acting on few and well-placed control points. Since these models are typically composed of patches that correspond to structural features of the shape (e.g., planes, through holes, etc), it is relatively easy to replace such features with others and re-design the model. In contrast, even if digitised models are extremely flexible and detailed, they are hardly modifiable without a proper *segmentation* that identifies their structural parts. Thus, segmentation has become a fundamental step for several application contexts, including Product Modelling, where segmented surfaces are computed for reverse engineering purposes [VMC97], and Computer Graphics, where properly segmented shapes support modern modelling paradigms [FKS*04].

Several algorithms have been proposed to hierarchically segment 3D surface meshes [AFS06, KT03], and recently some work has been dedicated to volume meshes [AMSF08]. Although *point sets* are widely used and flexible surface representations, the computation of part hierarchies out of point-sampled surfaces is still open to efficient solutions. Therefore, this work tackles the problem of hierarchically segmenting point sets in useful shape parts, and strives to reduce the gap between a huge development of mesh-based techniques and a relatively low attention on multi-resolution decompositions of point sets. Note that the lack of connectivity and the atomic definition of point sets make the segmentation more complex than for simplicial meshes. In fact, mesh segmentation algorithms rely on an explicit surface connectivity [DKT05] and are often based on the properties of discrete differential operators.

It is worth mentioning that the hierarchical segmentation of point sets provides benefits for a plethora of applications. In Computer Vision, it makes it possible to convert a raw point cloud to a hierarchy of features for shape recognition [Mar82, PKKG03]. In Computer Graphics, a hierarchical organisation of the parts provides a support for the definition of regions of interest of variable size, which are useful for editing purposes such as deformation or copy-and-paste modelling [AMSF08]. Furthermore, it provides a multi-scale

[†] Istituto di Matematica Applicata e Tecnologie Informatiche, Consiglio Nazionale delle Ricerche, Italy.

shape abstraction that can be used to match point-sampled surfaces [DGG03].

Overview and contributions Building on previous research results obtained for triangle meshes [AFS06], we present a novel approach that employs a hierarchical clustering to efficiently and accurately construct a *multi-resolution representation* of a point set. In our representation, the model is encoded as a structured hierarchy of point-sampled patches of various shapes and sizes. By construction, each patch is fairly approximated through either a *planar*, a *spherical*, a *conical*, a *cylindrical* or a *toroidal* surface. Starting from such a representation, we easily compute a single-resolution segmentation of the point set in which each segment is effectively idealised through one of the five *geometric primitives* employed. This choice makes our method an efficient and hierarchical solution to the problem of recovering the structure of a so-called *regular model* [VMC97], i.e. a 3D shape described as an assembly of the aforementioned primitives.

With respect to existing approaches, and in particular to [AFS06], this article introduces several innovations: to perform the clustering, mesh connectivity is replaced by the k-nearest neighbour graph, which is also used to evaluate the differential properties that support the fitting stage. In order to deal with uneven sampling densities and to improve the fitting quality, we exploit the areas of properly computed splats as weights associated to the points. Another innovation is the selection of a broader set of primitives, which makes it possible to obtain comprehensive hierarchical abstractions of any regular model. Besides adapting the fitting strategy for cylinders to better perform on point sets, we introduce a new, robust, and efficient fitting of cones and tori. A further original contribution is our novel algorithm to refine cluster boundaries in segmentations extracted out of our hierarchies.

We also show that our hierarchical representation is particularly useful for CAD feature selection. We believe that the proposed selection mechanism may have a significant impact in the usability of forthcoming CAD systems dealing with reverse-engineered objects. Finally, note that reconstructing a mesh is not always an easy task due to possible missing data, noise, disconnected portions of the point set or chamfered sharp edges; thus, computing a mesh to be segmented may not be feasible. By assuming that the points are sampled on a regular model, however, we show that an idealised mesh can be robustly reconstructed starting from the fitting primitives encoded in our hierarchy.

Paper structure The remainder of the article is organised as follows: in Section 2, we briefly review previous work on point-sampled surfaces and shape segmentation. Section 3 introduces the hierarchical clustering applied to point-sampled surfaces, discusses how to extract a segmentation out of the resulting hierarchy, and how to refine it. Section 4 defines the geometric primitives that drive the clustering. Details of the approach, along with experimental results and comparisons, are discussed in Section 5, whereas Section 6 shows the main applications of the proposed approach. Finally, conclusions and future work are outlined in Section 7.

2. Theoretical background and related work

In the following, we briefly review previous work on point-sampled surfaces (Section 2.1) and segmentation of 3D shapes (Section 2.2). For more details on these topics, we refer the reader to the state-of-the-art reports [AGP*04, GPA*02] and [APP*07, Sha06], respectively.

2.1. Point set surfaces: definition and differential analysis

A point-based representation of a surface $\mathcal{M}$ is a finite set $\mathcal{P}$ of points sampled on $\mathcal{M}$. The surface $\mathcal{M}$ underlying $\mathcal{P}$ is commonly estimated using the *moving least-squares* [ABCO*01, AK04, Lev03] and the *implicit* [AA03] *approximations*. Given a point set $\mathcal{P} := \{\mathbf{p}_i\}_{i=1}^{N}$, in the *k-nearest neighbour graph* $\mathcal{T}$ of $\mathcal{P}$ each point $\mathbf{p}_i \in \mathcal{P}$ is linked to its k nearest points in $\mathcal{P}$, which constitute the *neighbourhood* $\mathcal{N}_{\mathbf{p}_i} := \{\mathbf{p}_{j_s}\}_{s=1}^{k}$ of $\mathbf{p}_i$. For dense point sets, this graph provides enough information to approximate the local geometric and topological structure of $\mathcal{M}$ without meshing the whole point set. The computation of $\mathcal{T}$ requires $O(N \log N)$ time [AMN*98].

Point set surfaces The MLS surface underlying $\mathcal{P}$ is defined by minimising an energy function, which is the sum of weighted squared distances of points in $\mathcal{P}$ to a plane [AK04, Lev03]. Changing either the energy function or the normal field provides variants of the MLS surface [AA03, GG07]. For instance, the RMLS variant [FCOAS03] preserves sharp features of $\mathcal{M}$, which are commonly chamfered in standard MLSs. In this article, we only need to estimate the normals at the points of the input set $\mathcal{P}$. To this end, for each point $\mathbf{p}$, the un-oriented normal $\mathbf{n}(\mathbf{p})$ is defined as the unit eigenvector related to the smallest eigenvalue of the 3×3 symmetric covariance matrix C defined as

$$C := \sum_{i=1}^{N} (\mathbf{p}_i - \mathbf{p})(\mathbf{p}_i - \mathbf{p})^T \theta(\|\mathbf{p} - \mathbf{p}_i\|_2).$$

Here, θ is a decreasing weighting function, e.g. the Gaussian map $\theta(t) := \exp(-t^2/h^2)$, where h is the scale parameter. For each point, the value h is either set as the distance of the farthest of its k-nearest neighbours in $\mathcal{P}$ or kept constant by computing the average of these values on $\mathcal{P}$ [DS05]. Since the map θ rapidly decreases with the increase of the Euclidean distance among points, we substitute the indices in the previous sums with those of the points of $\mathcal{P}$ that belong to the k-nearest neighbours of $\mathbf{p}$. Neglecting the contributions of the points that do not belong to $\mathcal{N}_{\mathbf{p}}$ makes the computation

of $\mathbf{n}(\mathbf{p})$ much faster at the cost of an acceptable approximation error. Finally, the normals at the $\mathbf{p}_i$'s are coherently oriented by propagating the orientation of a seed normal along arcs of the *Riemannian graph* of the point set [HDD*94].

Principal curvatures and directions for point-sampled surfaces Let $\pi_\mathbf{p}$ be the tangent plane to $\mathcal{P}$ at $\mathbf{p}$, $\mathbf{n}(\mathbf{p})$ the corresponding normal vector, and $\mathbf{e}$ an arbitrary directional vector contained in $\pi_\mathbf{p}$. Then, we consider the intersection curve γ between the surface $\mathcal{M}$ underlying $\mathcal{P}$ and the plane that is orthogonal to $\mathbf{n}(\mathbf{p}) \times \mathbf{e}$ and interpolates $\mathbf{p}$. The value of the curvature of γ at $\mathbf{p}$ is called *normal curvature of* $\mathcal{M}$ at $\mathbf{p}$ along the direction $\mathbf{e}$ and it is uniquely determined by $\mathbf{e}$. As $\mathbf{e}$ varies among all the directional vectors in $\pi_\mathbf{p}$, the *principal curvatures* κ_1, κ_2 are defined as the maximum and minimum of the normal curvatures. In the following, the notation $\overline{\mathbf{n}}$ refers to the normalised vector $\overline{\mathbf{n}} := \mathbf{n}/\|\mathbf{n}\|_2$ of unit length. According to [YQ07, LP05], the *directional curvature* in $\mathbf{p}_i$ along $\mathbf{d} := \overline{\mathbf{e}_{ij}}$, $\mathbf{e}_{ij} := \mathbf{p}_j - \mathbf{p}_i$, is defined as

$$\kappa_{ij} := 2\frac{\langle \mathbf{n}(\mathbf{p}_i), \mathbf{p}_j - \mathbf{p}_i \rangle_2}{\|\mathbf{p}_j - \mathbf{p}_i\|_2^2}, \quad \mathbf{p}_j \in \mathcal{N}_{\mathbf{p}_i}.$$

Let $\overline{\mathbf{e}_{ij}^{\tan}}$ be the normalised tangential part of $\mathbf{e}_{ij}$ and $\overline{\mathbf{e}_{ij}^{\tan\perp}}$ its orthogonal component; as done in [LP05], let w_{ij} be a weight associated with the direction $\mathbf{e}_{ij}$ which depends on the local sampling density. Integrating the directional curvatures, we get an approximation W_i of the *Weingarten map* at $\mathbf{p}_i$; i.e.,

$$W_i := \sum_{\mathbf{p}_i \in \mathcal{N}_{\mathbf{p}_j}} w_{ij}\kappa_{ij}\overline{\mathbf{e}_{ij}^{\tan}}^T\overline{\mathbf{e}_{ij}^{\tan\perp}}.$$

Then, the principal curvatures κ_1, κ_2 of $\mathcal{P}$ at $\mathbf{p}_i$ are computed as the eigenvalues of W_i. In Section 4.4, the principal curvatures are used to identify the parameters that fully characterise the toroidal primitives.

2.2. 3D Shape segmentation and structure recovery

3D segmentation techniques can be classified in two main categories: *surface-based* and *part-based* approaches [APP*07, Sha06]. A surface-based segmentation is a decomposition of the input into a family of patches that have a uniform behaviour with respect to a specific surface property such as Gaussian curvature [YGZS05], approximating primitives [CDST97, VMC97, AFS06], flatness [GWH01], geodesic distances [KT03, ZTS02]. Other approaches employ spectral analysis [LZ04, YNBH06], scale space clustering [BSRS04], scissoring [LLS*05], and topological properties [PSF04, ZMT05, ZH04]. In contrast, part-based segmentation algorithms consider the input to be a solid, and the extracted parts are characterised by their volume [LJS97, LSA94, AMSF08]. Both surface and part-based methods can be either direct or hierarchical. In a direct method, a single segmentation is extracted and possibly post-processed (e.g., through variational optimization). Conversely, hierarchical approaches produce

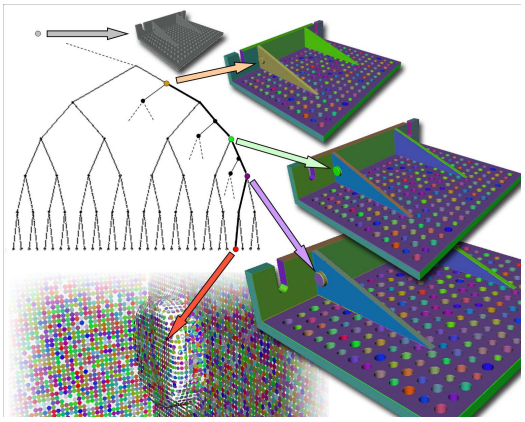


Figure 1: *Structural features encoded as* clusters of points *along the paths connecting the root of the binary tree with the leaves. Besides the point itself (red leaf), features of increasing size are captured; e.g., the planar top of the bolt, the bolt itself, the stiffener, and so on, up to the whole model.*

multiple segmentations where segments at finer levels are hierarchically nested in segments at coarser levels. In the following, we review direct (Section 2.2.1) and hierarchical segmentation algorithms (Section 2.2.2).

2.2.1. Regular models and direct segmentation

Since our work targets mostly *regular models*, the review of previous work is mainly focused on this class of objects. In [VMC97], regular model has been defined as a 3D shape that can be described as an assembly of primitive surfaces, which may be parts of planes, spheres, cylinders, cones, and tori. Várady *et al.* also proposed a surface-based algorithm to retrieve the constituting patches starting from a raw surface mesh. After a first coarse segmentation based on the identification of feature lines, their method classifies each region as *simple* or *multiple*, depending on whether or not the region can be effectively approximated by one of the primitives. In the latter case, each region is analysed and partitioned through dimensionality filtering on the Gaussian sphere. Note that regular models must not be confused with regular sets used for solid modelling applications.

In [SRR07], the segmentation is achieved by randomly selecting minimal sets, which are defined as the smallest number of points that define a pre-defined primitive, and by computing those primitives that approximate most of the points. The segmentation is robust to noise but the overall framework requires several parameters to guide the identification of the primitives. Furthermore, the randomness of the point sampling might provide different results depending on the thresholds used to define and minimise the score function, which measures the quality of a shape candidate.

In [VB04], first and second order surface properties guide

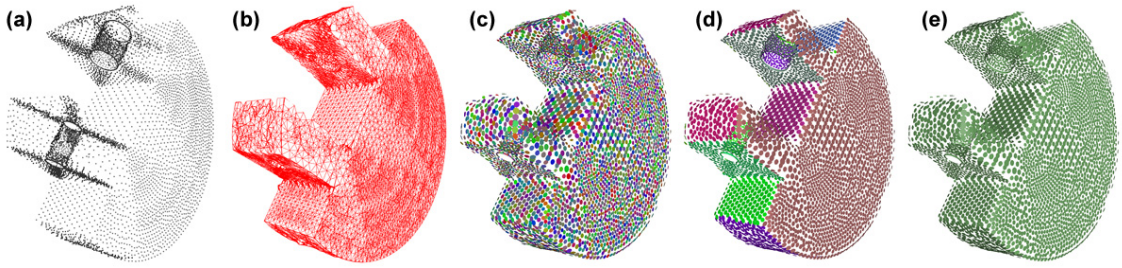


Figure 2: *Main steps of the proposed approach. (a) Input point set $\mathcal{P}$; (b) k-nearest neighbour graph of $\mathcal{P}$; (c) initial partition where each point is associated to its own singleton cluster. (d) Intermediate level of the clustering in which the point-sampled patches successfully capture the structural parts of the object. (e) Final single cluster constituted by $\mathcal{P}$.*

the direct segmentation of a point set into planes, spheres, cylinders, and cones. After a first phase where a coarse segmentation is computed, a variational approach is applied to optimize the boundaries of the features extracted in the first phase. A variational optimization approach is also used in [CSAD04] to approximate complex polygon meshes with few planar patches; this method is based on the $k-means$ clustering, and it has been extended in [WK05] to treat also spherical, cylindrical, and rollingball-like features. The k-means clustering is exploited in [MFX*07] too, where neighboring points are grouped according to their reciprocal Euclidean distances and the angular differences among the normal directions.

2.2.2. Hierarchical segmentation

In [GWH01], a hierarchical clustering of triangulated surfaces in nearly planar patches was introduced. Arcs of the *dual graph* of the mesh are iteratively *collapsed*, and the order of such operations is determined by the planarity of the corresponding clusters of triangles in the primal graph, i.e. the input mesh. Similarly, in [Ino01] the clustering is used to simplify the structure of CAD models, and the edge-contraction order is driven by the area of the clusters, their flatness, and the smoothness of their boundaries.

In [AFS06], the work of [GWH01] was extended to fit spherical and cylindrical patches as well. The cost associated to each contraction is the minimum L^2 residual among those of the cluster's best-fitting plane, sphere, and cylinder. Unfortunately, this technique can approximate only a subset of the primitives used in [VMC97]. Finally, in [GG04] the hierarchical clustering is guided by *slippable motions*, i.e. rigid transformations that make the surface slide against the stationery version without forming any gaps. In other words, the motion of each point is required to be tangent to the mesh. A slippable component is a collection of vertices which can be approximated through a slippable motion. A hierarchical clustering such as in [GWH01] is exploited to compute splippable components which, in their turn, can constitute planes, spheres, cylinders, linear extrusion surfaces, surfaces of revolution and helical surfaces.

Though it was not designed to specifically treat CAD models, the algorithm presented in [KT03] proved to provide useful segmentations for this class of shapes too. Instead of using a *bottom-up* hierarchical clustering approach, this method is based of a fuzzy k-means decomposition that proceeds *top-down* by iteratively splitting the whole shape along lines of deep concavity.

Unfortunately, though some work have been dedicated to the segmentation of point sets [VB04, YNBH06, MFX*07], no existing approach has been designed to automatically compute a hierarchy of features constituting a comprehensive abstraction of a regular model.

3. Hierarchical point set clustering

The basic idea of the hierarchical clustering is to iteratively merge neighbouring elements into representative clusters as long as possible, eventually representing the whole shape through a single cluster. Depending on the shape representation, elements to be aggregated can be triangles [GWH01, AFS06], tetrahedra [AMSF08], or simply points. The proposed approach belongs to the latter case, and two points are neighbours if they are connected through an arc within the k-nearest neighbour graph. The result of the iterative clustering is a binary tree of clusters providing a hierarchical representation of the shape, namely:

1. each single point is a leaf of the tree;
2. the whole shape is the root;
3. a non-leaf node exists if the clusters represented by its two children were merged into a single cluster.

Since the order in which clusters are merged influences the resulting binary tree, the ordering criterion must make the resulting tree (Figure 1) useful for some specifically targeted applications. For a regular model, the extracted clusters should capture the structure of the underlying surface.

To this end, we consider the same set of primitives dealt with in [VMC97]; i.e., *planes*, *spheres*, *cylinders*, *cones*, and *tori*. Then, we strive to sort the merging operations so that each of the resulting clusters is effectively approximated by

Algorithm 1 Main steps of the hierarchical clustering.

Require: A point set $\mathcal{P}$.
Ensure: A binary tree of clusters representing subsets of $\mathcal{P}$.

1: Compute the k-nearest neighbour graph $\mathcal{T}$ of $\mathcal{P}$.
2: Compute an unoriented version $\mathcal{U}$ of $\mathcal{T}$ in which the arc (v_i, v_j) exists if either (v_i, v_j) or (v_j, v_i) are in $\mathcal{T}$.
3: Associate a *cost* to each arc e in $\mathcal{U}$; we indicate this value as $Cost(e)$.
4: Create a priority queue h containing all the arcs of $\mathcal{U}$ sorted according to their cost.
5: **while** $h \neq \emptyset$ **do**
6: remove the first arc $e = (v_i, v_j)$ from h;
7: contract e to a single node v. This means that e, v_i, and v_j are all removed from $\mathcal{U}$ and replaced by the single new node v;
8: remove possible duplications of arcs incident at v;
9: update the cost associated to all the arcs incident at v and update their position in h accordingly.
10: **end while**

one of the geometric primitives employed. In the following, we provide an overview on the main steps of the proposed approach (Section 3.1); then, we discuss how to extract a segmentation out of the resulting hierarchy (Section 3.2) and how to optimise it (Section 3.3).

3.1. Algorithm overview

After having computed the k-nearest neighbour graph $\mathcal{T}$ of the input point cloud $\mathcal{P}$, an un-oriented version $\mathcal{U}$ of $\mathcal{T}$ is built in which the arc (v_i, v_j) exists if either (v_i, v_j) or (v_j, v_i) are in $\mathcal{T}$. In this setting, $\mathcal{U}$ identifies the connectivity of the point set and, in analogy to [GWH01], the algorithm iteratively contracts all the arcs of $\mathcal{U}$ to generate the hierarchy of clusters (Algorithm 1). Initially, each node of $\mathcal{U}$ identifies a single point of $\mathcal{P}$ and the contraction of the first arc corresponds to the union of two points within a single new cluster. Hence, the node replacing the contracted edge is a cluster with two points. Note that $\mathcal{U}$ represents the connectivity of the clusters at each step of the algorithm.

To perform the contractions in the desired order, each arc is associated a *cost* and all the arcs are inserted into a priority queue sorted according to increasing cost values. At each step, the first arc is removed from the queue and contracted (i.e., $\mathcal{U}$ is modified), the two corresponding clusters are merged, and all the arcs incident to the new node are re-positioned within the queue according to their updated cost.

Computation of the cost associated to an arc. Let $e = (v_i, v_j)$ be an arc in $\mathcal{U}$, and let $C(v)$ denote the set of points aggregated so far within the cluster represented by the node v of $\mathcal{U}$. Also, let $Q(e) = C(v_i) \cup C(v_j)$ be the set of points that would form a cluster as the result of the contraction of e. Let $L^2_{plane}(Q(e))$ denote the fitting error of the best-fitting plane through $Q(e)$; this value is computed as the sum of the squared residuals between the points of $Q(e)$ and their best-fitting *plane*. Using an analogous notation for all the other primitives employed, we associate to e the following cost

$$Cost(e) = \min\{L^2_{plane}(Q(e)), L^2_{sphere}(Q(e)), \ldots \\ L^2_{cylinder}(Q(e)), L^2_{cone}(Q(e)), L^2_{torus}(Q(e))\}.$$

Each such quantity is computed as described in Section 4. For simplicity of the exposition, we did not include explicitly the computation of each set $C(v)$ in Algorithm 1. These sets, however, must be computed as singletons during the initialisation, and incrementally updated at each contraction. The binary tree of clusters generated by the algorithm can be drawn within a 2D coordinate frame so that the y-coordinate of each node corresponds to the cost of aggregating its sub-clusters. Such a diagram (*dendrogram*), can be cut through a horizontal line at the desired y-value to produce a single-resolution partitioning or, equivalently, a segmentation of the point set. In Figure 2, three segmentations are shown as they result by cutting the dendrogram at the level of the leaves (c), at an intermediate level (d), and at root level (e).

3.2. Extraction of a segmentation

To compute a segmentation out of the hierarchy, one may select a desired number of clusters and stop the aggregation when such a number is reached. This choice, however, would be somehow against the philosophy of using a hierarchical setting, in which a preferred number of clusters is not known *a priori*. Moreover, in these cases k-means based partitioners would probably perform a better job.

Assuming that the fitting error monotonically grows as the clustering proceeds with the bottom-up construction of the tree, another solution is to select a maximum admissible approximation error and stop the algorithm when the next aggregation would exceed such an error. The monotonicity of the error growth is verified as follows. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two clusters and $\mathcal{C}_{12}$ be their union. Denoting the fitting primitive of $\mathcal{C}_i$ with $F_{\mathcal{C}_i}$ and its residual error with $d(\mathcal{C}_i, F_{\mathcal{C}_i})$, we have that $d(\mathcal{C}_{12}, F_{\mathcal{C}_{12}}) \geq d(\mathcal{C}_1, F_{\mathcal{C}_1}) + d(\mathcal{C}_2, F_{\mathcal{C}_2})$. Indeed, the primitive $F_{\mathcal{C}_{12}}$ is, by definition, the one that minimises the fitting error. Since the error is a sum of terms indexed on the points of the extracted primitives, if we use $F_{\mathcal{C}_{12}}$ to approximate $\mathcal{C}_1$ and $\mathcal{C}_2$ then $d(\mathcal{C}_{12}, F_{\mathcal{C}_{12}}) = d(\mathcal{C}_1, F_{\mathcal{C}_{12}}) + d(\mathcal{C}_2, F_{\mathcal{C}_{12}})$. Otherwise, using their own *best* fitting primitives $F_{\mathcal{C}_1}$ and $F_{\mathcal{C}_2}$, the total residual error can only be reduced or left unchanged.

To take into account the model size and the fact that we use weights while computing fitting errors (Section 4), in our implementation the threshold is related to the length of the bounding box diagonal l_{bb} as follows. Let $L^2(\mathcal{P})$ be the fitting error, and let $\tilde{d}^2 := \frac{1}{\sum_i w_i} L^2(\mathcal{P})$ be the average squared

Algorithm 2 Optimisation of an initial partitioning of $\mathcal{P}$.

Require: A point set $\mathcal{P}$ segmented into a set of clusters; let $\mathbf{p}.C$ be the cluster associated to the point $\mathbf{p}$.
Ensure: An optimised assignment of the points to clusters.

```
Initialise the number n_i of iterations to be zero; i.e., n_i := 0.
Choose n_max be the maximum number of iterations.
for n_i < n_max do
    n_i++;
    compute the best-fitting primitive B_c for each cluster;
    for e := (i, j) ∈ T such that p_i.C ≠ p_j.C do
        if d(p_j, B_{p_i.C}) < d(p_j, B_{p_j.C}) then
            p_j.C := p_i.C;
        end if
        if d(p_i, B_{p_j.C}) < d(p_i, B_{p_i.C}) then
            p_i.C := p_j.C;
        end if
        if no points were re-assigned then
            break;
        end if
    end for
end for
```

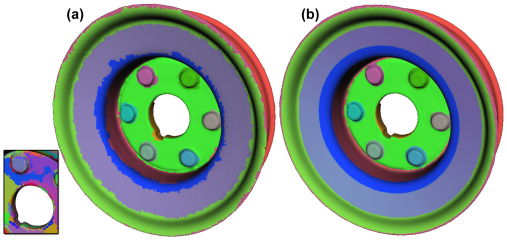


Figure 3: *Segmentation of a mechanical part (a) before and (b) after variational optimisation. In (a), boundary jaggedness is due to the aggregation of smaller clusters from previous levels in the hierarchy (*bottom-left *box).*

distance of the cluster from its best fitting primitive. We stop the algorithm when the next aggregation would make the value $\frac{\sqrt{d^2}}{l_{bb}}$ exceed an *absolute* threshold ε. In this way, if we scale the input while keeping ε constant, the resulting segmentation does not change. To extract the segmentations of Figures 3, 6, 7, and 8, the parameter ε was set to 0.01.

Note that some segmentations extracted as described here may not be easily obtained indirectly by identifying the regions' boundaries. Regions that are interesting from a CAD user point of view, indeed, may not be bounded by sharp features or by lines easily detectable on the basis of morphological characteristics (e.g., the red cluster in Figure 6(a) and the fillets in Figure 8, *right*).

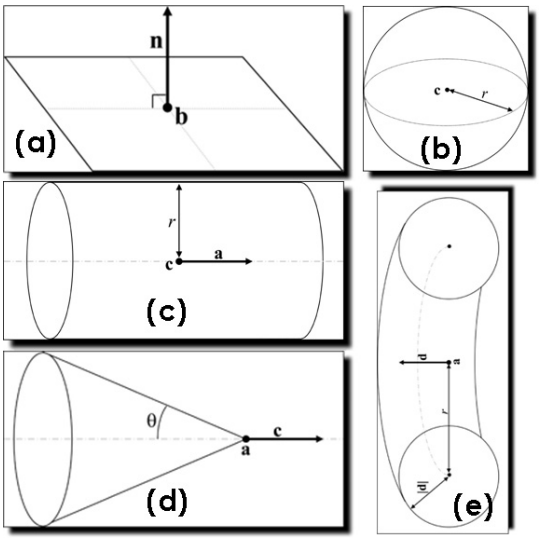


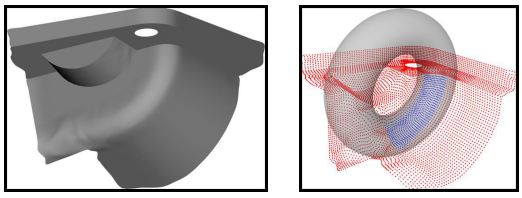
Figure 4: *Fitting primitives and parameters: (a)* plane*, (b)* sphere*, (c)* cylinder*, (d)* cone*, and (e)* torus.

3.3. Region optimisation

At the beginning of the aggregation, several primitives are eligible to interpolate the points being clustered. In fact, seven "well-placed" points are required to uniquely identify a torus, which means that if a cluster contains less than eight points we can assume that the best-fitting primitive interpolates these points. Thus, all the contractions are assigned a null cost, the aggregation proceeds in a random order and, even when clusters become large enough to avoid any ambiguity, we inherit an irregular distribution of points near the boundaries between adjacent segments (Figure 3(a)).

In order to get rid of this irregularity, we propose a post-processing based on a variational optimisation of an initial partitioning. Roughly speaking, we re-compute the *best-fitting primitive* (*BFP*, for short) for each current cluster. Then, we consider a point $\mathbf{p}$ close to the boundary between two clusters and re-assign it to the cluster whose *BFP* is closest to $\mathbf{p}$. The process is repeated for each point having at least a neighbour assigned to a different cluster. When no more points can be re-assigned, we re-compute the best-fitting primitives and proceed to the assignment phase once again. The process stops after a prescribed number of iterations or when, even after the update of the *BFP*s, no more re-assignment takes place. The optimisation algorithm is summarised in Algorithm 2, where $d(\mathbf{p},B) := \min_{\mathbf{q}\in B} \|\mathbf{p}-\mathbf{q}\|_2$ is the Euclidean distance between the point $\mathbf{p}$ and its closest point on the primitive B. Our tests have shown that the optimisation never needs more than four iterations to converge. Comparing Figure 3(a) with (b) shows the effect of the optimisation.

Though this approach is close in spirit to [WK05], it is



(a) (b)

Figure 5: *(a) Input point-sampled surface and (b) toroidal primitive that best fits the blue points.*

substantially different; at each step of our method, only the points on the border of clusters are allowed to *jump* into adjacent clusters. In contrast, at each step of [WK05], every cluster is *deleted* and re-grown starting from the triangle which is closest to the proxy. This difference makes our algorithm less "aggressive", and typically the modifications remain closer to the original boundaries. As for [CSAD04, WK05], Algorithm 2 may lead to disconnected clusters, but this is not a problem for the applications discussed in this article. With respect to the approach proposed in [VB04], our region optimisation does not require any threshold to be set, and can treat toroidal surfaces (not supported in [VB04]).

4. Fitting primitives

Herewith, $\mathcal{P}$ denotes the set of points in a cluster. We consider a set of five geometric primitives, namely *planes*, *spheres*, *cylinders*, *cones*, and *tori*. Each primitive is unambiguously identified by its parameters (Figure 4); e.g., a centre point and a radius uniquely identify a sphere. For each primitive, our objectives are the computation of (a) the parameters that identify the primitive and (b) the distance between $\mathcal{P}$ and the primitive. For planes [GWH01] and spheres [Pra87], algorithms are known to efficiently compute their parameters by minimising the *fitting error*. This error is defined as the sum $L^2_{primitive}(\mathcal{P})$ of the squared residuals between $\mathcal{P}$ and its best-fitting primitive.

The proposed clustering deals with rather dense point clouds, and the hierarchy to be recovered is actually interesting just in those parts where clusters contain numerous points each (i.e., the top of the dendrogram). Thus, optimal fitting strategies are not essential as long as the approximation discriminates among the various primitives. For example, if $\mathcal{P}$ is a set of sufficiently numerous points sampled on a sphere, then $\mathcal{P}$ must be closer to the approximately best-fitting sphere than to all the other best-fitting primitives.

If the point set $\mathcal{P} := \{\mathbf{p}_i\}_{i=1}^N$ is uniformly sampled, then the *weight* w_i of each $\mathbf{p}_i$ is set equal to one; otherwise, it is set equal to the area of the splat associated to $\mathbf{p}_i$ and computed as done in [WK04]. In this way, those points of $\mathcal{P}$ located in regions with a low sampling density are assigned to higher weights, as they represent larger portions of the underlying surface $\mathcal{M}$. Unless stated differently, in the remainder all the sums range from $i = 1$ to N. The definition of planar and spherical primitives is discussed in Section 4.1; the procedure for cylinders, cones, and tori is detailed in Section 4.2, 4.3, and 4.4, respectively.

4.1. Planes and spheres

The *center* of the planar primitive is identified by $\mathbf{b} := \frac{\sum_i w_i \mathbf{p}_i}{\sum_i w_i}$ and its normal is the eigenvector of the covariance matrix $M := \sum_i w_i(\mathbf{p}_i - \mathbf{b})(\mathbf{p}_i - \mathbf{b})^T$ corresponding to its minimum eigenvalue. In this case, the fitting error is $L^2_{plane}(\mathcal{P}) := \sum_i w_i |\langle \mathbf{n}, \mathbf{p}_i - \mathbf{b} \rangle_2|^2$.

For a spherical primitive (Figure 4(b)), the *centre* $\mathbf{c} := [c_x, c_y, c_z]^T$ and *radius* r are computed by minimising the algebraic distance of the sphere from $\mathcal{P}$ [AFS06]. Indicating with $\mathbf{p}_i := (p_{ix}, p_{iy}, p_{iz})$, $i = 1, \ldots, N$, the coordinates of the points of $\mathcal{P}$, these parameters are given by

$$[c_x, c_y, c_z, r^2 - c_x^2 - c_y^2 - c_z^2]^T = (A^T A)^{-1} A^T \mathbf{b}, \qquad (1)$$

where $W := \mathrm{diag}(w_1, \ldots, w_N)$ is the diagonal matrix whose non-null entries are the weights $\{w_i\}_{i=1}^N$ and

$$A := W \begin{bmatrix} 2p_{1x} & 2p_{1y} & 2p_{1z} & 1 \\ 2p_{2x} & 2p_{2y} & 2p_{2z} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 2p_{Nx} & 2p_{Ny} & 2p_{Nz} & 1 \end{bmatrix}, \quad \mathbf{b} := W \begin{bmatrix} \|\mathbf{p}_1\|_2^2 \\ \|\mathbf{p}_2\|_2^2 \\ \vdots \\ \|\mathbf{p}_N\|_2^2 \end{bmatrix}.$$

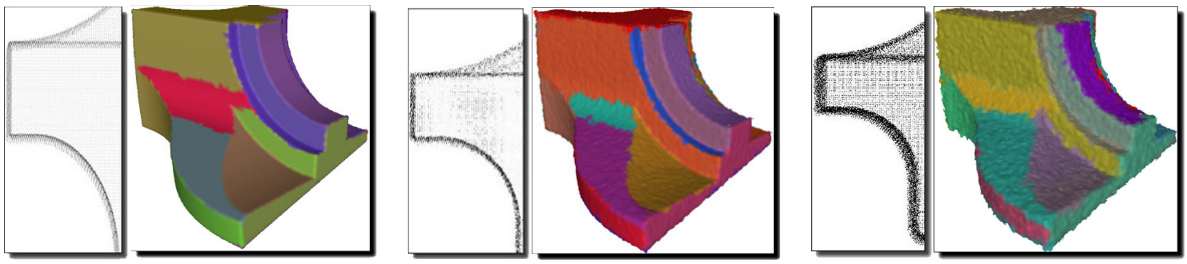
The fitting error is defined as

$$L^2_{sphere}(\mathcal{P}) := \sum_i w_i (\|\mathbf{p}_i - \mathbf{c}\|_2 - r)^2.$$

4.2. Cylinders

A cylinder is conveniently identified through the following parameters: a *unit vector* $\mathbf{a}$ specifying the direction of the axis; a *centre point* $\mathbf{c}$ on the axis; a *radius* r (Figure 4(c)). As shown in [CG01, PGK02, VMC97, WK05], if normal information is available and reliable, then the problem of computing fitting cylinders is solved rather robustly. Thus, our solution is inspired by these approaches and normal vectors to the points of $\mathcal{P}$ are estimated as reviewed in Section 2.1. The first step computes the cylinder axis as the unit vector which is the *most orthogonal* to all the normal vectors $\mathbf{n}_i := \mathbf{n}(\mathbf{p}_i)$ at $\mathbf{p}_i$. To do this, we define the positive semi-definite matrix $C := \sum_i w_i(\mathbf{n}_i \mathbf{n}_i^T)$ and assign to $\mathbf{a}$ the eigenvector of C corresponding to its minimum eigenvalue. Also, let $\mathbf{d}_x$ and $\mathbf{d}_y$ be the other two eigenvectors of C, and let $\tilde{\mathbf{p}}_i := [\langle \mathbf{p}_i, \mathbf{d}_x \rangle_2, \langle \mathbf{p}_i, \mathbf{d}_y \rangle_2]$ be the 2D projection of $\mathbf{p}_i$ on a plane orthogonal to $\mathbf{a}$ and passing through the origin.

Using a 2D version of (1), the centre $\tilde{\mathbf{c}} := [\tilde{c}_x, \tilde{c}_y]$ and radius r of a circle that fits the weighted $\tilde{\mathbf{p}}_i$s are computed. While the radius of such fitting circle is identified with the radius of the cylinder, its centre $\tilde{\mathbf{c}}$ must be transformed back



(a) (b) (c)

Figure 6: *Stability of the segmentation with respect to a varying amount of noise, which grows from (a) to (b) and (c). While increasing the noise, the number, type, and shape of each primitive do not change.*

to 3D space, i.e., $\mathbf{c} := \tilde{c}_x\mathbf{d}_x + \tilde{c}_y\mathbf{d}_y$. Once the parameters of the fitting cylinder have been found, the fitting error is

$$L^2_{cylinder}(\mathcal{P}) := \sum_i w_i(\|(\mathbf{p}_i - \mathbf{c}) \times \mathbf{a}\|_2 - r)^2.$$

4.3. Cones

A cone is conveniently identified through the following parameters: a *unit vector* $\mathbf{c}$ specifying the direction of the axis; an *apex* $\mathbf{a}$; and the *tangent* of the semi-apical angle $\tan\theta$ (Figure 4(d)). For the computation of the cone that best fits a point set, the iterative algorithms presented in [KL87, LMM98] are too slow for our purposes. Thus, as we did for cylinders, we exploit normal information to cast the problem to a robust, approximate, and analytic approach.

Firstly, we compute the cone axis $\mathbf{c}$. Since the Gauss map of a cone is a circle orthogonal to $\mathbf{c}$, we compute the best-fitting plane of the Gauss map of $\mathcal{P}$ and consider the normal of such a plane to be the direction $\mathbf{c}$ of the axis. Let $\mathbf{n}_i := \mathbf{n}(\mathbf{p}_i)$ be the normal vector at $\mathbf{p}_i$ and $\mathbf{b} := \frac{1}{\sum_i w_i}\sum_i w_i\mathbf{n}_i$ the weighted average of these normals. We define a positive semi-definite matrix $C := \sum_i w_i(\mathbf{n}_i - \mathbf{b})(\mathbf{n}_i - \mathbf{b})^T$ and assign to $\mathbf{c}$ the coordinates of the eigenvector of C corresponding to its minimum eigenvalue. Let $\mathbf{d}_i := \overline{(\mathbf{n}_i \times \mathbf{c}) \times \mathbf{n}_i}$ be a normalised tangent vector at $\mathbf{p}_i$ coplanar with $\mathbf{c}$. If $\mathbf{p}_i$ belongs to a cone, then the line passing through $\mathbf{p}_i$ with direction $\mathbf{d}_i$ contains the apex of the cone and $\mathbf{d}_i$ is the direction of zero curvature. Thus, we define the apex $\mathbf{a}$ as the point which is closest to all such straight lines in the least squares sense. The squared distance of a point $\mathbf{q}$ from the straight line L_i defined by a point $\mathbf{p}_i$ and a unit directional vector $\mathbf{d}_i$ is

$$d^2(\mathbf{q}, L_i) := \|(\mathbf{q} - \mathbf{p}_i) \times \mathbf{d}_i\|_2^2.$$

Our objective is to find $\mathbf{a}$ such that the quadratic convex functional $E(\mathbf{a}) = \sum_i d^2(\mathbf{a}, L_i)$ is minimised or, equivalently, such that all the partial derivatives of $E(\mathbf{a})$ are null. Taking into account the weights of the $\mathbf{p}_i$s leads to a linear system $(\sum_i w_i M_i)\mathbf{a} - (\sum_i w_i\mathbf{b}_i) = \mathbf{0}$, where the matrix M_i and the vector $\mathbf{b}_i$ are defined for each $\mathbf{p}_i$ as follows

$$M_i := \begin{bmatrix} d_{iy}^2 + d_{iz}^2 & -d_{ix}d_{iy} & -d_{ix}d_{iz} \\ -d_{ix}d_{iy} & d_{ix}^2 + d_{iz}^2 & -d_{iy}d_{iz} \\ -d_{ix}d_{iz} & -d_{iy}d_{iz} & d_{ix}^2 + d_{iy}^2 \end{bmatrix},$$

$$\mathbf{b}_i := \begin{bmatrix} p_{ix}d_{iy}^2 - p_{iy}d_{ix}d_{iy} - p_{iz}d_{ix}d_{iz} + p_{ix}d_{iz}^2 \\ p_{iy}d_{iz}^2 - p_{iz}d_{iy}d_{iz} - p_{ix}d_{iy}d_{ix} + p_{iy}d_{ix}^2 \\ p_{iz}d_{ix}^2 - p_{ix}d_{iz}d_{ix} - p_{iy}d_{iz}d_{iy} + p_{iz}d_{iy}^2 \end{bmatrix}.$$

Hence, the apex $\mathbf{a}$ of the conical primitive is computed as $\mathbf{a} := (\sum_i w_i M_i)^{-1}(\sum_i w_i\mathbf{b}_i)$. When the parameters of the fitting cone are all available, the fitting error is

$$L^2_{cone}(\mathcal{P}) := \frac{1}{1+\tan^2\theta}\sum_i w_i(\|(\mathbf{p}_i - \mathbf{a}) \times \mathbf{c}\|_2 + - |\langle \mathbf{p}_i - \mathbf{a}, \mathbf{c}\rangle_2| \tan\theta)^2.$$

We now calculate the parameter $\tan\theta$ so that the above equation is minimised. This is equivalent to finding $\tan\theta$ so that the derivative of $L^2_{cone}(\mathcal{P})$ is equal to zero; i.e.,

$$\tan\theta = \frac{\sum_i w_i\|(\mathbf{p}_i - \mathbf{a}) \times \mathbf{c}\|_2}{\sum_i w_i|\langle \mathbf{p}_i - \mathbf{a}, \mathbf{c}\rangle_2|}.$$

4.4. Tori

A torus is identified through the following parameters: a *centre point* $\mathbf{a}$; a *height vector* $\mathbf{d}$; and a *radius* r (Figure 4(e)). Note that r is the radius of the circular axis of the torus, $\|\mathbf{d}\|_2$ is the radius of the generating circle, and $\overline{\mathbf{d}}$ is the normalised direction of the axis of symmetry. To compute these parameters, we observe that one of the principal curvatures of a torus is constant and its reciprocal is the radius $\|\mathbf{d}\|_2$ of the circle that generates the torus. Let k_{i1} and k_{i2} be the (signed) principal curvatures at $\mathbf{p}_i$ (Section 2.1), with $k_{i1} \le k_{i2}$. The variance of the principal curvatures within $\mathcal{P} := \{\mathbf{p}_i\}_{i=1}^N$ is

$$\begin{cases} Var(k_1) := \frac{N}{|\sum_i k_{i1}|}\sum_i k_{i1}^2 - |\sum_i k_{i1}|, \\ Var(k_2) := \frac{N}{|\sum_i k_{i2}|}\sum_i k_{i2}^2 - |\sum_i k_{i2}|. \end{cases}$$

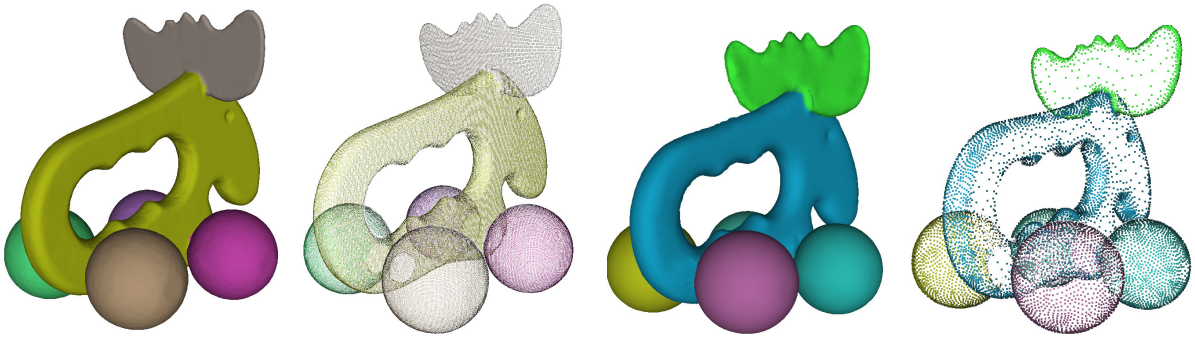


Figure 7: *Segmentation robustness with respect to a fairly different sampling density.*

Let $\tilde{k}_1 := \frac{1}{N}\sum_i k_{i1}$ be the average minimum curvature, and let $\tilde{k}_2$ be the average maximum curvature within $\mathcal{P}$. The radius $\|\mathbf{d}\|_2$ of the generating circle is assigned the absolute value of the reciprocal r_d of the principal curvature that changes the less, that is

$$r_d := \begin{cases} 1/\tilde{k}_1 & \text{if } Var(k_1) < Var(k_2), \\ 1/\tilde{k}_2 & \text{otherwise.} \end{cases}$$

and $\|\mathbf{d}\|_2 := |r_d|$. Let $\mathbf{q}$ be a point belonging to the torus, and let $\mathbf{n}(\mathbf{q})$ be the normal at $\mathbf{q}$. Note that when $\mathbf{q}$ is moved towards the opposite direction of $\mathbf{n}(\mathbf{q})$ at a signed distance r_d, its new position is on the circular axis of the torus. Indeed, the circular axis coincides with the locus of points obtained by moving all the points of the torus as described.

We now move all the $\mathbf{p}_i$s using this process and compute their best-fitting plane (Section 4.1) and consider the normal of such a plane as the direction of the height vector $\mathbf{d}$. Furthermore, we project these transformed points on the fitting plane and compute their best-fitting circle, whose radius and centre correspond to the radius r and centre $\mathbf{a}$ of the torus.

Let us consider the projection $\tilde{\mathbf{p}}_i := \mathbf{p}_i - r_d\mathbf{n}_i$ of the point $\mathbf{p}_i$, where $\mathbf{n}_i$ is the normal at $\mathbf{p}_i$ and the point $\tilde{\mathbf{b}} := \frac{1}{\sum_i w_i}\sum_i w_i\tilde{\mathbf{p}}_i$ is the barycentre of the $\tilde{\mathbf{p}}_i$s. Let $\tilde{M} := \frac{1}{\sum_i w_i}\sum_i w_i(\tilde{\mathbf{p}}_i - \tilde{\mathbf{b}})(\tilde{\mathbf{p}}_i - \tilde{\mathbf{b}})^T$ be the covariance matrix of the $\tilde{\mathbf{p}}_i$s. Indicating with $\{\overline{\mathbf{d}}, \mathbf{d}_x, \mathbf{d}_y\}$ the eigenvector of the matrix $\tilde{M}$ corresponding to the eigenvalues $\lambda_1 \le \lambda_2 \le \lambda_3$, the height vector of the torus is assigned the value $\mathbf{d} := \|\mathbf{d}\|_2\overline{\mathbf{d}}$. Now, let $\tilde{\mathbf{p}}_i := [\langle\tilde{\mathbf{p}}_i, \mathbf{d}_x\rangle_2, \langle\tilde{\mathbf{p}}_i, \mathbf{d}_y\rangle_2]$ be the projection of $\tilde{\mathbf{p}}_i$ on a plane orthogonal to $\mathbf{d}$. As we did for cylinders, we reduce the dimensionality of (1) to find the centre $\tilde{\mathbf{c}} := [\tilde{c}_x, \tilde{c}_y]$ and radius r of a circle that fits the weighted $\tilde{\mathbf{p}}_i$s. The centre of the torus is obtained by transforming $\tilde{\mathbf{c}}$ back to 3D space as $\mathbf{c} := \tilde{c}_x\mathbf{d}_x + \tilde{c}_y\mathbf{d}_y$ and the radius r of the torus corresponds to the radius of the fitting circle. Finally, the fitting error is computed as

$$L^2_{tori}(\mathcal{P}) := \sum_i w_i \left[(\|(\mathbf{p}_i - \mathbf{a}) \times \overline{\mathbf{d}}\|_2 - r)^2 + \right.$$
$$\left. + |\langle \mathbf{p}_i - \mathbf{a}, \overline{\mathbf{d}}\rangle_2|^2 \right]^{1/2} - \|\mathbf{d}\|_2)^2.$$

5. Results and discussion

Our prototype has been implemented in C++ and tested on an Intel Core 2 PC equipped with 2 Gb of RAM. To compute the k-nearest neighbour graph, we used a function provided by the ANN library [AMN*98] that employs kd-trees to efficiently compute exact nearest neighbours. In all our experiments, we used $k = 10$ to calculate the connectivity graph $\mathcal{U}$, while a larger value of k was used to compute the normal vectors and the curvature tensors. This latter value of k ranged from 20 when the cloud was affected by low or no noise (e.g., Figure 6(a,b)), to 40 when the point-set was significantly noisy (e.g., Figure 6(c)). In general, we have experimented that a proper increase of k within such a range makes our algorithm sufficiently stable in presence of noisy data. Furthermore (Section 4), we have verified that weighting the points during the computation of the fitting primitives and the corresponding residuals makes our approach suitable to treat point sets with fair variations in sampling densities (Figure 7). However, the cloud must be locally dense enough to permit the use of the k-nearest neighbours as acceptable substitutes of an explicit connectivity. *PointShop3D* was used to produce the illustrations depicting point-sampled surfaces. The point sets illustrated in Figures 3, 7(*left*), 8(*left*), 9, and 11 are all aligned range scans produced by laser digitisation. As for data noise, the intrinsic coherent misalignment among the various scans might impact the order of the very first point aggregations, but it is far too small to have an influence on any *meaningful* segmentation.

For the sake of simplicity, the input point set is assumed to represent a single connected object. If multiple components must be treated, we analyse the connectivity of $\mathcal{U}$ and then

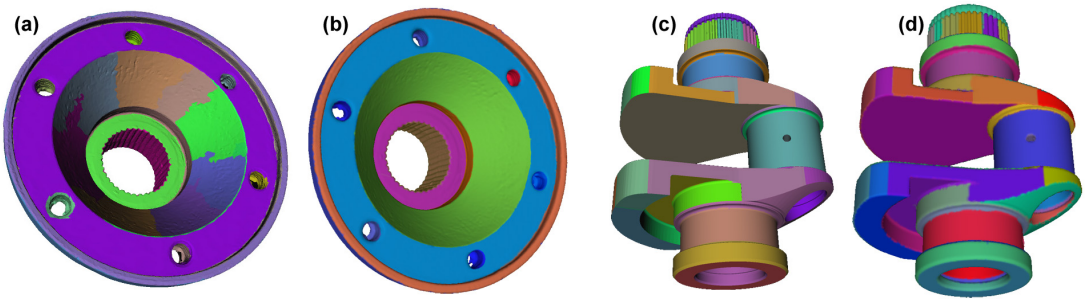


Figure 8: *(a) The* carter *model could not be segmented properly through [AFS06], whereas (b) the complete set of primitives employed by our approach makes the segmentation more significant. If the primitives used by [AFS06] are sufficient, then the* crank *model could be segmented properly by both [AFS06] (c) and our method (d).*

run the clustering separately on the various components. Alternatively, $\mathcal{U}$ is enriched with the arcs of the *Euclidean minimum spanning tree* (EMST, for short) of $\mathcal{P}$ and the algorithm is run once on the resulting single component. In this latter case, if the connected components are assumed to be features of the shape and not induced by a poor sampling, then the arcs of the EMST that were not already in $\mathcal{U}$ may be contracted after all the others. In the following, we discuss the computational complexity (Section 5.1), the comparison with previous work, and the main limitations (Section 5.2) of the proposed approach.

5.1. Computational complexity and performances

We analyse the algorithmic complexity of our clustering algorithm by firstly assuming that the tree is perfectly balanced, and then by generalising the result also in light of experimental observations. Without loss of generality, we suppose that the number N of points of $\mathcal{P}$ is a power of two. Since the number E of edges in $\mathcal{U}$ is bounded by $2kN$, where k is the number of neighbours used to compute $\mathcal{U}$, we have that $O(E) = O(N)$. To count the number of operations needed to construct the binary tree, we scan the hierarchy level by level from the leaves (level n in the tree) to the root (level 1), with $n = log_2 N$. To construct level $n-1$, we generate the fitting primitives of E potential clusters made of 2^1 points each. For level $n-2$, we need to compute the primitives of $E/2$ potential clusters made of 2^2 points each and so on. In general, for the level $n-i$ we need to compute $E/(2^{i-1})$ potential clusters made of 2^i points each. The computation of each fitting primitive is linear, hence for 2^i points it requires $O(2^i)$ operations. Thus, the total number of operations necessary to construct the hierarchy is proportional to

$$\sum_{i=1}^{n-1} \frac{E}{2^{i-1}} 2^i = 2E(n-1). \qquad (2)$$

Since $N = 2^n$ and $E \in O(N)$, (2) becomes $2E(n-1) \in O(En) = O(2^n n) = O(N \log N)$ and it reasonably matches the actual time required by our implementation on various examples (Table 1, Figure 10). For the worst-case complexity, which occurs when the tree is completely unbalanced, the number of levels n is equal to the number of points N. Following a similar path as above, the worst case complexity is $O(N^2)$. We conclude that the complexity varies between $N \log N$ and N^2, depending on how balanced the tree is. Experiments tend to show that in practice we are closer to the balanced tree (i.e., favorable) situation.

5.2. Comparison with the state of the art

We have compared our approach with state-of-the-art algorithms. The first comparison is against the method proposed in [AFS06], in which the triangles of a mesh are hierarchically clustered while minimising the approximation error with geometric primitives. Besides being able to work on point sets without any explicit connectivity, a significant advantage of our method is the broader set of primitives used for approximation, which makes it possible to obtain useful segmentations for a larger class of models. For example, the conical feature in the *carter* model (Figure 8, *left*) could not be captured by [AFS06], whereas our approach successfully provides a useful segmentation. If the object contains only planar, spherical and cylindrical features, then the two methods provide comparable results (Figure 8, *right*). When the number of elements (i.e., triangles in [AFS06] and points in our method) being clustered is comparable, we observed that our algorithm is slightly slower, as a consequence of the larger number of primitives.

In a second experiment, we have compared our approach with the algorithm proposed in [SRR07], which provides an optimal partitioning of the input point set according to a criterion that is similar to the one that drives our clustering. We have observed that by interrupting the clustering as described in Section 3.2, we obtain segmentations comparable with the results reported in [SRR07] (Figure 11). This is a noticeable fact because our approach was not studied to create a single optimal segmentation, while it was designed to

Figure 9: *Segmentation of a basilica with 2M points.*

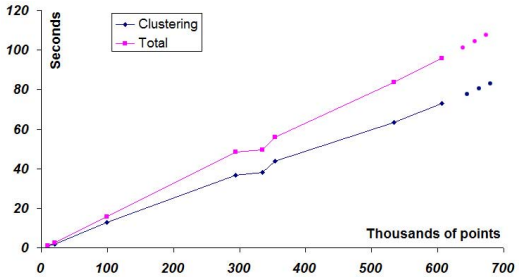


Figure 10: *With reference to Table 1, this graph highlights that our algorithm computes the hierarchy in a time that is (nearly) linearly proportional to the number of input points.*

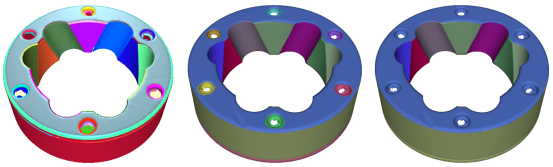


Figure 11: *Segmentation obtained by [SRR07] (*left*) and (middle,* right*) two resolutions of our hierarchy.*

produce a whole hierarchy of segmentations which are used in a much broader spectrum of applications.

Limitations Our approach has been designed to reconstruct a hierarchical organisation of the features composing a regular model; if the input does not belong to this class, then our method does not provide significant results. In principle, it should not be difficult to extend the algorithm to cope with a larger class of models, perhaps including quadric surfaces [YLW06]. However, the *freedom* in shape patches characterised by numerous parameters would probably lead to higher computational costs. Another drawback is the use of a global priority queue that makes it difficult to parallelise the algorithm to deal with large clouds of points.

6. Applications

In this section, our hierarchical segmentation is exploited to implement a powerful region-selection mechanism (Section 6.1) and to reconstruct manifold meshes (Section 6.2).

6.1. Interactive region selection with applications to model re-design

When our method is applied to regular models, clusters often correspond to structural features of the object which are hierarchically nested within the binary tree. For example, a cylinder may correspond to a through hole, a plane to a thin slab, a torus to a fillet, and so on. Furthermore, the fillet itself may be part of a slot that, in its turn, is part of a slab in the hierarchy. This fact suggested us to exploit our hierarchical clustering for the interactive selection of shape features. According to previous results on tetrahedral meshes [AMSF08], we have developed an intuitive system that interactively selects parts of the shape by traversing the binary tree of clusters. The cost used to sort merging operations, as well as the greedy and incremental nature of the algorithm itself, makes the resulting hierarchies contain virtually all the interesting features of the shape at all the scales. Thus, by browsing the hierarchy, it is likely that a user finds several "interesting" features that are eligible for editing.

On this premise, we have developed an interaction mechanism to select a shape feature by clicking on a point (i.e., a pixel rendering a splat), and by rotating the mouse wheel to select the size of the feature containing the point clicked. The feature hierarchy is computed once as a pre-processing step. Then, a mouse click on a point corresponds to selecting a leaf of the cluster tree, and the rotation of the mouse wheel causes a motion upwards/downwards along the path connecting the selected leaf to the root of the tree. The nodes along this path are the clusters in the hierarchy; in particular, the path represents a sequence of clusters approximated by geometric primitives of increasing size and each cluster properly contains its predecessor in the sequence (Figure 1).

In the example of Figure 12, the user may click on a point on the bolt and rotate the mouse wheel to select among the bolt itself, the whole stiffener containing the bolt, and so on, up to the whole shape. Once a feature has been selected, its boundary is refined on the fly through a local run of the optimisation described in Section 3.3 where only the selected feature and its adjacent clusters are modified.

Though Figure 12 shows an example of re-design, our purpose is not to describe a re-design system: this would require to describe too many aspects (e.g., interaction techniques to shift the features, algorithms to resample the resulting holes, ...) that are out of the scope of the article. The focus of the figure, in fact, is only to show a simple application of our mechanism for feature selection. Also, note that using

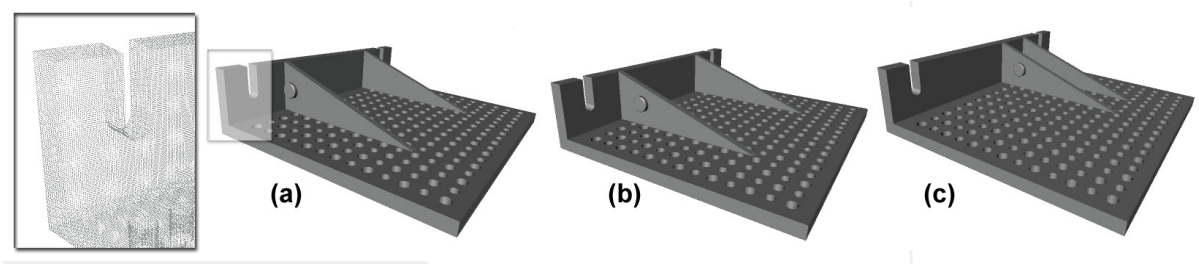


Figure 12: *After having computed the hierarchy of the model (a), the user clicked on the bolt, rotated the mouse wheel to select all its points, and moved the bolt to a new position (b) on the stiffener. Through a further wheel rotation, the user increased the size of the selection to cover the whole stiffener. (c) Finally, the stiffener along with its bolt were moved to a new position.*

Table 1: Timings. *Number N of points and seconds required to perform the clustering: the total time is the sum of this value plus k-nearest neighbours ($k := 20$), normals and curvature tensor computation. See also Fig. 10.*

Test	N	Clustering	Total
Figure 2	10000	0,9	1,17
Figure 3	293672	36,64	48,39
Figure 6	98556	12,9	15,81
Figure 7(a)	353334	43,8	55,97
Figure 7(b)	20776	1,95	2,59
Figure 8 *left*	533746	63,3	83,77
Figure 8 *right*	334511	38,25	49,48
Figure 9	2187366	281,52	381,5
Figure 11	606542	72,88	95,75

a hierarchy provides more flexibility than a single-resolution segmentation. In Figure 12, a single segmentation would capture either the bolt only or the group *stiffener+bolt*, but not both as needed in our redesign application. The use of a hierarchy allows one to abstract the geometry depending on the scale, so that the group *stiffener+bolt* is considered as an approximate stiffener at one scale, and as an actual pair *stiffener* plus *bolt* at a finer scale.

6.2. Idealised mesh reconstruction

When our hierarchical clustering is used to extract a segmentation, clusters may be exploited to create a patch-based manifold connectivity among the points. To this end, let us suppose that the input point set $\mathcal{P}$ has been subdivided into a family $\{R_i\}_{i=1}^{k}$ of clusters, such that $R_i \subseteq \mathcal{P}$ and each cluster R_i has been associated to its best-fitting primitive Σ_i and parameters (Section 4). Focusing on the cluster R_i, we project each point $\mathbf{p}_{j_s} \in R_i$, $s = 1,\ldots,l$, onto the point $\tilde{\mathbf{p}}_{j_s} \in \Sigma_i$ that minimises the Euclidean distance $\|\mathbf{p}_{j_s} - \tilde{\mathbf{p}}_{j_s}\|_2$. This step is easily done analytically based on the parameters defining the primitive. Once each point of R_i has been mapped on Σ_i, we reconstruct a local manifold connectivity among the projected points using the algorithm in [GKS00], which computes the surface normals by assuming specific sampling conditions. In contrast, our implementation associates to each projected point the normal of the fitting primitive at that point, thus making the reconstruction more robust. After this initial reconstruction, which results in a triangle mesh that we call K_i, each triangle t_{i_j} is analysed. If at least one of its three edges does not belong to the k-nearest neighbour graph of $\mathcal{P}$, then t_{i_j} is removed from K_i.

In some cases, the resulting meshes are glued together to form a single connected component. To do this, we exploit the graph $\overline{\mathcal{U}}$ that codes the adjacency relations among the clusters when Algorithm 1 is terminated to extract the segmentation. Also, we use the original graph $\mathcal{U}$ that codes the initial point-to-point connectivity at the beginning of Algorithm 1. Then, for each arc $e = \{R_a, R_b\} \in \overline{\mathcal{U}}$ we proceed as follows: first, we search in $\mathcal{U}$ the shortest arc s connecting two points $\mathbf{p}_{a_i}$ and $\mathbf{p}_{b_j}$ belonging to R_a and R_b respectively and being boundary vertices in K_a and K_b. Then, we merge K_a and K_b in a new single mesh K_{ab} by connecting $\mathbf{p}_{a_i}$ and $\mathbf{p}_{b_j}$ through two new triangles as described in [AF06]. After having connected all the K_is using the aforementioned technique, the remaining holes are triangulated as done by Barequet and Sharir in [BS95]. Finally, sharp features are reconstructed by running *EdgeSharpener* [AFRS05]. The whole process is depicted in the example of Figure 13.

Clearly, the union of the initial K_is succeeds only if they are actually parts of a whole manifold and orientable surface (Figure 13). Actually, the projection of points on their corresponding primitive may generate intrinsically non-manifold configurations, as in the example of Figure 1 where the stiffener is idealised with a single plane. In these cases, our mesh reconstruction stops after the creation of the K_is, and lets the user choose whether to join manually the patches.

Note that if the input is sampled on a regular model, this approach to mesh reconstruction has the important effect of removing (not only reducing) the noise through a least-squares approximation. Furthermore, the user may act on the threshold error to stop the clustering so that unimportant tiny features are removed as well, thus producing an idealized

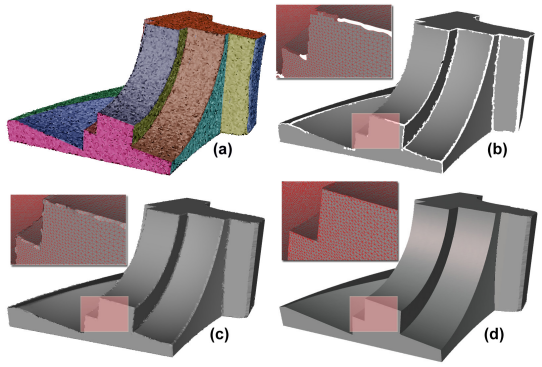


Figure 13: *Once a proper segmentation out of (a) a noisy point set has been extracted, each point is projected on the best-fitting primitive of its cluster, for which a local mesh (b) is constructed. (c) Cluster-patches are connected through strips of triangles to form a single component and (d) sharp edges are reconstructed using [AFRS05].*

surface out of the input points. Clearly, these considerations do not apply if the input is not a regular model.

7. Conclusions and future work

We have presented an algorithm to construct a multi-resolution representation of a point-sampled surface and to effectively implement powerful mechanisms for region selection, which can be exploited when re-designing reverse-engineered objects. Then, we have shown that starting from our hierarchies manifold meshes can be reconstructed through single-resolution segmentations. Our future work will be focused on the inclusion of new fitting surfaces and on the speed-up of the algorithm. For this last aim, we will exploit a random sampling [SRR07] or an initial segmentation that splits the input properly to parallelise the clustering [VB04, VMC97]. Finally, the surface reconstruction can be improved by exploiting the intersection of the extracted fitting primitives instead of using the EdgeSharpener algorithm.

Acknowledgements This work has been partially supported by FOCUS K3D, FP 7 CA. We acknowledge the *PointShop3D* software and the ANN library. Special thanks are given to Bianca Falcidieno and the members of the *Shape Modelling Group* at IMATI-CNR, Genova-Italy for helpful discussion on shape segmentation. Models are courtesy of the AIM@SHAPE repository, the Institut fuer Photogrammetrie of the Stuttgart University, and Drexel University.

References

[AA03] ADAMSON A., ALEXA M.: Approximating and intersecting surfaces from points. In *Symp. on Geometry Processing* (2003), pp. 230–239.

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *IEEE Visualization* (2001), pp. 21–28.

[AF06] ATTENE M., FALCIDIENO B.: Remesh: An interactive environment to edit and repair triangle meshes. In *Shape Modeling and Applications* (2006), pp. 271–276.

[AFRS05] ATTENE M., FALCIDIENO B., ROSSIGNAC J., SPAGNUOLO M.: Sharpen and bend: Recovering curved sharp edges in triangle meshes produced by feature-insensitive sampling. *IEEE Trans. on Visualiz. and Comp. Graphics 11*, 2 (2005), 181–192.

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

[AGP*04] ALEXA M., GROSS M., PAULY M., PFISTER H., STAMMINGER M., ZWICKER M.: Point-based computer graphics. In *ACM Siggraph Course Notes* (2004).

[AK04] AMENTA N., KIL Y. J.: Defining point-set surfaces. In *ACM Siggraph* (2004), pp. 264–270.

[AMN*98] ARYA S., MOUNT D. M., NETANYAHU N. S., SILVERMAN R., WU A. Y.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM 45*, 6 (1998), 891–923.

[AMSF08] ATTENE M., MORTARA M., SPAGNUOLO M., FALCIDIENO B.: Hierarchical convex approximation for fast region selection. *Computer Graphics Forum 27*, 5 (2008), 1323–1333.

[APP*07] AGATHOS A., PRATIKAKIS I., PERANTONIS S., SAPIDIS N., AZARIADIS P.: 3D mesh segmentation methodologies for cad applications. *Computer-Aided Design and Applications 4*, 6 (2007), 827–841.

[BS95] BAREQUET G., SHARIR M.: Filling gaps in the boundary of a polyhedron. *Computer-Aided Geometric Design 12*, 2 (1995), 207–229.

[BSRS04] BESPALOV D., SHOKOUFANDEH A., REGLI W. C., SUN W.: Local feature extraction using scale-space decomposition. In *Design Engineering Technical Conferences* (2004).

[CDST97] CHAZELLE B., DOBKIN D., SHOURHURA N., TAL A.: Strategies for polyhedral surface decomposition: An experimental study. *Computational Geometry: Theory and Applications 7*, 4-5 (1997), 327–342.

[CG01] CHAPERON T., GOULETTE F.: Extracting cylinders in full 3D data using a random sampling method and the gaussian image. In *Vision Modeling and Visualization* (2001), pp. 35–42.

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. *ACM Transactions on Graphics 23*, 3 (2004), 905–914.

[DGG03] DEY T. K., GIESEN J., GOSWAMI S.: Shape segmentation and matching with flow discretization. *Lecture Notes in Computer Science 2748* (2003), 25–36.

[DKT05] DESBRUN M., KANSO E., TONG Y.: Discrete differential forms for computational modeling. In *ACM Siggraph 2005 Courses* (2005), p. 7.

[DS05] DEY T. K., SUN J.: An adaptive MLS surface for reconstruction with guarantees. In *Symp. on Geometry Processing* (2005), pp. 43–52.

[FCOAS03] FLEISHMAN S., COHEN-OR D., ALEXA M., SILVA C. T.: Progressive point set surfaces. *ACM Transactions on Graphics 22*, 4 (2003), 997–1011.

[FKS*04] FUNKHOUSER T., KAZHDAN M., SHILANE P., MIN P., KIEFER W., TAL A., RUSINKIEWICZ S., DOBKIN D.: Modeling by example. *ACM Transactions on Graphics 23*, 3 (2004), 652–663.

[GG04] GELFAND N., GUIBAS L. J.: Shape segmentation using local slippage analysis. In *Symp. on Geometry Processing* (2004), pp. 214–223.

[GG07] GUENNEBAUD G., GROSS M.: Algebraic point set surfaces. In *ACM Transactions on Graphics* (2007), vol. 26, p. 23.

[GKS00] GOPI M., KRISHNAN S., SILVA C.: Surface reconstruction based on lower dimensional localized delaunay triangulation. *Computer Graphics Forum 19*, 3 (2000), 467–478.

[GPA*02] GROSS M., PFISTER H., ALEXA M., PAULY M., STAMMINGER M., ZWICKER M.: Point-based computer graphics. *Eurographics Tutorial*, 2002.

[GWH01] GARLAND M., WILLMOTT A., HECKBERT P.: Hierarchical face clustering on polygonal surfaces. In *ACM Symp. on Interactive 3D Graphics* (2001), pp. 49–58.

[HDD*94] HOPPE H., DEROSE T., DUCHAMP T., HALSTEAD M., JIN H., MCDONALD J., SCHWEITZER J., STUETZLE W.: Piecewise smooth surface reconstruction. In *Computer Graphics and Interactive Techniques* (1994), pp. 295–302.

[Ino01] INOUE K.; TAKAYUKI I. A. Y. T. F. K. S.: Face clustering of a large-scale cad model for surface mesh generation. *Computer-Aided Design 33* (2001), 251–261.

[KL87] KELKER D., LANGENBERG C. W.: A mathematical model for orientation data from macroscopic elliptical conical folds. *Mathematical Geology 19*, 8 (1987), 729–743.

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

[Lev03] LEVIN D.: Mesh-independent surface interpolation. *Geometric Modeling for Scientific Visualization 3* (2003), 37–49.

[LJS97] LEONARDIS A., JAKLIC A., SOLINA F.: Superquadrics for segmenting and modeling range data. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19* (1997), 1289–1295.

[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.

[LMM98] LUKÁCS G., MARTIN R., MARSHALL D.: Faithful least-squares fitting of spheres, cylinders, cones and tori for reliable segmentation. *Lecture Note in Computer Science, Computer Vision 1406*, 1 (1998), 671–686.

[LP05] LANGE C., POLTHIER K.: Anisotropic smoothing of point sets. *Computer Aided Geometric Design 22*, 7 (2005), 680–692.

[LSA94] LEONARDIS A., SOLINA F., ALENKA M.: A direct recovery of super-quadric models in range images using recover-and-select paradigm. In *Proc. of Conf. on Computer Vision* (1994), pp. 309–318.

[LZ04] LIU R., ZHANG H.: Segmentation of 3D meshes through spectral clustering. In *Pacific Graphics* (2004), pp. 298–305.

[Mar82] MARR D.: *Vision*. Freeman Publishers, New York, 1982.

[MFX*07] MIAO Y.-W., FENG J.-Q., XIAO C.-X., PENG Q.-S., FORREST A. R.: Differentials-based segmentation and parameterization for point-sampled surfaces. *Journal of Computer Science and Technology 22*, 5 (2007), 749–760.

[PGK02] PAULY M., GROSS M., KOBBELT L. P.: Efficient simplification of point-sampled surfaces. In *Proc. of the conference on Visualization* (2002), pp. 163–170.

[PKKG03] PAULY M., KEISER R., KOBBELT L. P., GROSS M.: Shape modeling with point-sampled geometry. *ACM Transactions on Graphics 22*, 3 (2003), 641–650.

[Pra87] PRATT V.: Direct least-squares fitting of algebraic surfaces. *ACM Siggraph 21*, 4 (1987), 145–152.

[PSF04] PATANÈ G., SPAGNUOLO M., FALCIDIENO B.: Paragraph: graph-based parameterization of triangle meshes with arbitrary genus. *Computer Graphics Forum 23*, 4 (2004), 783–797.

[Sha06] SHAMIR A.: Segmentation and shape extraction of 3D boundary meshes. In *Eurographics 2006 State of the Art Reports* (2006), pp. 137–149.

[SRR07] SCHNABEL R., R. W., R. K.: Efficient RANSAC for point-cloud shape detection. *Computer Graphics Forum 26*, 2 (2007), 214–226.

[VB04] VANCO M., BRUNNETT G.: Direct segmentation of algebraic models for reverse engineering. *Computing 72*, 1-2 (2004), 207–220.

[VMC97] VÁRADY T., MARTIN R. R., COX J.: Reverse engineering of geometric models - an introduction. *Computer-Aided Design 29*, 4 (1997), 255–268.

[WK04] WU J., KOBBELT L.: Optimized sub-sampling of point sets for surface splatting. *Computer Graphics Forum 23*, 3 (2004), 643–652.

[WK05] WU J., KOBBELT L.: Structure recovery via hybrid variational surface approximation. *Computer Graphics Forum 24*, 3 (2005), 277–284.

[YGZS05] YAMAUCHI H., GUMHOLD S., ZAYER R., SEIDEL H.-P.: Mesh segmentation driven by gaussian curvature. *The Visual Computer 21*, 8-10 (2005), 659–668.

[YLW06] YAN D., LIU Y., WANG W.: Quadric surface extraction by variational shape approximation. *Lecture Notes in Computer Science 4077* (2006), 73.

[YNBH06] YAMAZAKI I., NATARAJAN V., BAI Z., HAMANN B.: Segmenting point sets. In *Proc. of Shape Modeling and Applications* (2006), pp. 4–13.

[YQ07] YANG P., QIAN X.: Direct computing of surface curvatures for point-set surfaces. In *Symp. on Point-based Graphics* (2007), pp. 605–608.

[ZH04] ZHOU Y., HUANG Z.: Decomposing polygon meshes by means of critical points. In *Multimedia Modeling Conference* (2004), pp. 187–195.

[ZMT05] ZHANG E., MISCHAIKOW K., TURK G.: Feature-based surface parameterization and texture mapping. *ACM Transactions on Graphics 24*, 1 (2005), 1–27.

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