Blending Surface Segmentation and Editing for 3D Models

Long Zhang, Jianwei Guo, Jun Xiao, Xiaopeng Zhang, Dong-Ming Yan

Abstract—Recognizing and fitting shape primitives from underlying 3D models are key components of many computer graphics and computer vision applications. Although a vast number of structural recovery methods are available, they usually fail to identify blending surfaces, which corresponds to small transitional regions among relatively large primary patches. To address this issue, we present a novel approach for automatic segmentation and surface fitting with accurate geometric parameters from 3D models, especially mechanical parts. Overall, we formulate the structural segmentation as a Markov random field (MRF) labeling problem. In contrast to existing techniques, we first propose a new clustering algorithm to build superfacets by incorporating 3D local geometric information. This algorithm extracts the general quadric and rolling-ball blending regions, and improves the robustness of further segmentation. Next, we apply a specially designed MRF framework to efficiently partition the original model into different meaningful patches of known surface types by defining the multilabel energy function on the superfacets. Furthermore, we present an iterative optimization algorithm based on skeleton extraction to fit rolling-ball blending patches by recovering the parameters of the rolling center trajectories and ball radius. Experiments on different complex models demonstrate the effectiveness and robustness of the proposed method, and the superiority of our method is also verified through comparisons with state-of-the-art approaches. We further apply our algorithm in applications such as mesh editing by changing the radius of the rolling balls.

Index Terms—Mesh segmentation, Structure recovery, Superfacets, Rolling-ball blending surface, Markov random field.

1 Introduction

The acquisition of 3D geometry with high resolution and complexity is simplified through advances in 3D scanning and multiview reconstruction techniques. Although the obtained geometric models are usually represented by raw 3D data (*e.g.*, point clouds or triangular meshes), accurately segmenting these models and recovering the inherent structures are vital to the success of better shape understanding and high-level model processing, such as geometry compression [1], hybrid shape representations [2], and reverse engineering [3], [4]. Conducting such a procedure for model reconstruction is also inevitable [5], [6], [7].

Many mesh segmentation and shape recovery methods have been proposed to find a concise and accurate approximation of a given model. Existing approaches can be divided into several types according to the theories they employ. Greedy approaches use heuristics based on local cues to obtain several clusters until convergence is reached [8], [9], whereas variational approaches [1], [10] use interleaving segmentation and approximation to obtain optimal geometric proxies for the input surface. In addition, another popular set of approaches [2], [11], [12], [13] introduce energy functions to label the surface elements, thereby greatly accelerating the surface approximation procedure.

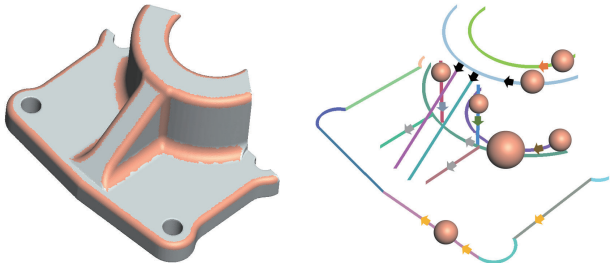

Fig. 1. Illustration of the rolling-ball blending surface: (a) a 3D mesh model with the rolling-ball blending surfaces highlighted in dark orange. (b) The rolling-balls and their moving trajectories to construct the blending surfaces.

Although previous approaches can address segmentation in models that contain simple primitives (*e.g.*, plane, cylinder, sphere, or quadric surfaces), they cannot produce satisfactory results in models that contain blending surfaces. Here, blending is an operation of creating a smooth and continuous transition between adjacent primary surfaces. Blending appears frequently in Computer-aided design (CAD) systems for aesthetic, functional improvement, stress-concentration reduction and manufacturability [14], [15] reasons. Although there are different methods to create blends via various mathematical forms [16], the rolling-ball method, where blends are generated by sweeping a rolling ball moving in contact with two neighboring surfaces, is the most common method for creating blends due to its simplicity and intuitive behavior [17], [18], [19]. Fig. 1 illustrates an example of rolling-ball blending surfaces. Recovering constant radius rolling-ball blends is a very important reverse

- *L. Zhang, J. Xiao are with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: zzlzlzl001@gmail.com, xiaojun@ucas.ac.cn. (Long Zhang and Jianwei Guo are joint first authors with equal contribution. Jun Xiao is the corresponding author.)*
- *J. Guo, X. Zhang, D.-M. Yan are with NLPR, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, and the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: jianwei.guo@nlpr.ia.ac.cn, Xiaopeng.Zhang@ia.ac.cn, yandongming@gmail.com.*

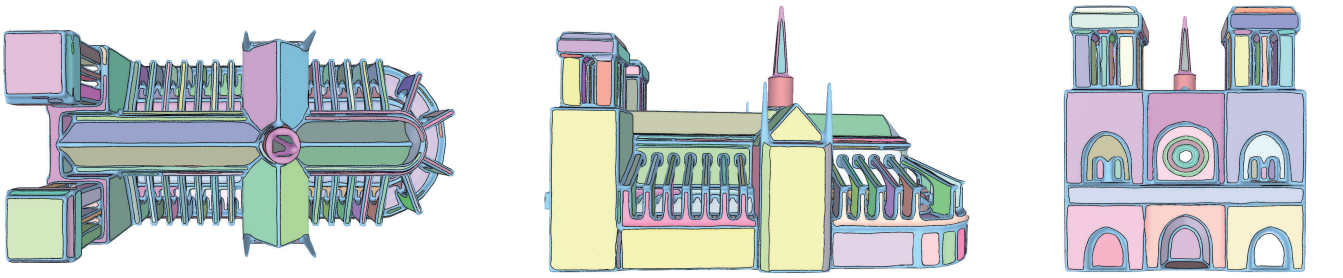

Fig. 2. Our segmentation result on the "Church" model, shown in different views: up (left), front (middle), and left (right). Different patches with known surface types are coded using random colors.

engineering problem.

Few prior works recover the constant radius of rolling-ball blends from 3D point data [20] and mesh surfaces [10]. These approaches are based on two basic steps, namely, radius estimation and spine/rolling-trajectory reconstruction. Radius estimation uses the average principal curvature to estimate the blending radius and initial spine position. Subsequently, the spine curve is updated iteratively. Unfortunately, the curvature estimation step is slow and inaccurate, thereby resulting in large calculation errors in later steps. In addition, curvature estimations cannot handle complex blending surface shapes, such as the crossover region intersected by two rolling trajectories. Moreover, these methods assume that the input has been pre-segmented and the primary surfaces are already known.

In this study, we propose a more general Markov random field (MRF)-based segmentation framework with emphasis on blending surface identification. We aim to segment the mesh-based models into consistent patches according to their geometric attributes while precisely obtaining the rolling-ball blending patches and the parameters of the blending radius and rolling trajectory. Different from traditional labeling approaches, we first construct superfacets, which refer to the clusters of triangular faces on the mesh. Thus, we can preliminarily eliminate the small-scale noise in the input mesh and greatly accelerate the segmentation procedure. Then, each superfacet is given a label of a known surface type by minimizing a multi-label energy function that encodes probabilistic formulations. Fitting algorithms are finally performed to reveal the structure of the input model and are used for mesh blending. In summary, the main contributions of this work include the following:

- A new MRF formulation for segmenting 3D models into meaningful patches and finding a faithful approximating primitive for each patch, especially the rolling-ball blending patches.
- An improved method to build mesh superfacets with boundaries that are well aligned with the features in the underlying surface. This method is tailored for our MRF segmentation and reasonably robust to noise in the input surface.
- An efficient optimization algorithm based on robust skeleton extraction to fit rolling-ball blending patches by accurately recovering the parameters of the rolling center trajectory and blending radius. The algorithm is demonstrated by conducting mesh smoothing and sharpening applications.

2 Related work

In this section, we present a brief overview of the key mesh segmentation and surface fitting techniques for 3D models presented in past years. For more comprehensive discussions, we refer the reader to the survey papers [21], [22], [23], [24].

Greedy segmentation. Greedy based approaches have been widely used for mesh segmentation, where the object partition is created by greedily clustered polygonal elements. These traditional algorithms usually cluster geometric elements with similar properties, such as region growing, hierarchical clustering, and hierarchical decomposition. In region growing methods [9], [25], a set of triangular faces are first selected as seeds, then for each seed grows a region based on some local properties until all the faces are assigned to a region.

The hierarchical clustering method iteratively merges two adjacent small patches according to the least merging error to a larger patch. This merging process is repeated until some stopping criteria are met. For example, Kim *et al*. [26] proposed using a merging criteria based on element orientations and merged over segmented regions in an iterative process. Attene *et al*. [27] presented a hierarchical segmentation framework that partitions a given shape into connected regions approximated by primitives belonging to a given set. Zhang *et al*. [28] introduced another novel hierarchical shape segmentation method based on splats for 3D shapes. Yang and Jia [29] recognized simple primitives by introducing a strategy for primitive priority and a new scheme for boundary smoothness between adjacent clusters. However, these methods show some weakness in segmenting blending regions because they lack good strategies for dealing with the merging of two smooth surfaces during the early stages.

In contrast, the hierarchical decomposition segment meshes into meaningful components from the top to bottom hierarchically. The method presented by Mangan and Whitaker [30] generalized the morphological watersheds to 3D surfaces. Katz and Tal [31] proposed to compute a decomposition which generally refers to segmentation at regions of deep concavities, and avoids jaggy boundaries between the components. Lai *et al*. [32] worked on feature-sensitive isotropic remeshing to generate a mesh hierarchy especially suitable for the segmentation of large models with regions at multiple scales. This kind of approaches easily over-segment some meaningful parts because of its tendency to segment surfaces at concave regions.

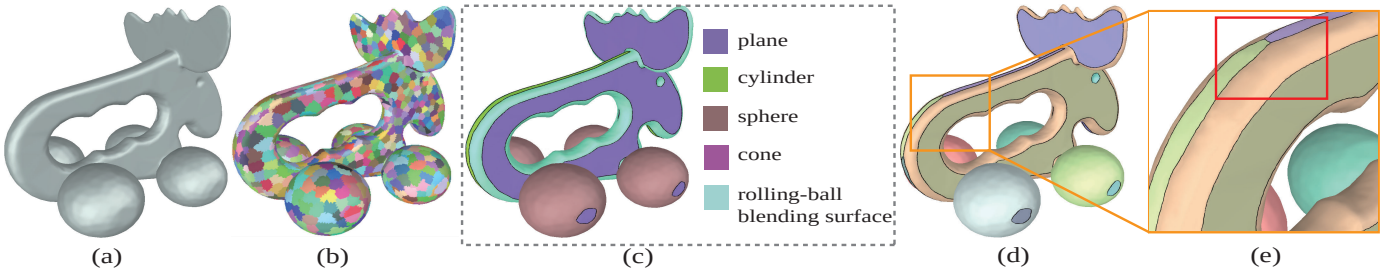

Fig. 3. The pipeline of our segmentation framework: (a) the input mesh model of "elk"; (b) construction of mesh superfacets; (c) segmentation by labeling each superfacet using MRF-based function while partitioning the rolling-ball blending surfaces correctly (purple color for planes, lawngreen for cylinders, dark brown for spheres, pink for cones and cyan for rolling-ball blending patches); (d) final result by segmenting each kind of surfaces into different instances, where the color of each instance is randomly generated for visualization purpose; (e) zoom-in view of the rolling-ball blending patches.

Variational segmentation. Different from greedy approaches, variational methods treat segmentation as an energy minimization problem. It shows powerful approximation ability and achieves better segmentation quality. Based on Lloyd's clustering algorithm [33], Cohen-Steiner *et al*. [1] cast shape approximation as a variational geometric partitioning problem. In this method, mesh partition and surface fitting are repeated mutually to minimize a global approximation error until convergence. After that, Wu and Kobbelt [10] extended this method by introducing planes, spheres, cylinders, and rolling-ball blending patches as primitive proxies. But this variant tends to over-segment the regions between patches. Yan *et al*. [34] further used general quadratic primitives to represent the geometric proxy of a surface region. Sander *et al*. [35] use Lloyd's clustering algorithm [33] to partition the model into charts to obtain the flexibility needed to parametrize meshes of arbitrary complexity, high genus, and multiple components with less distortion. In addition, Simari and Singh [36] addressed the restructuring of dense polygon meshes using a number of ellipsoidal regions. Julius *et al*. [37] introduced a novel metric of surface developability, then based on this metric they presented an algorithm named D-charts to segment meshes into (near-)developable charts. Le and Duan [38] presented a new geometric proxies method that can be easily extendable to surfaces of revolution. However, these approaches have a problem in correctly partitioning blending patches.

CVT-based segmentation. Centroidal Voronoi tessellation (CVT) can also be applied to mesh segmentation because of its nature of space partition. The theory and computation of CVT have been extensively studied in previous works [39] [40]. Then, Liu *et al*. [41] proposed a numerical framework for CVT computation based on quasi-Newton methods with higher efficiency. Edwards *et al*. [42] proposed the κCVT to achieve topological correctness in all flat regions while sparsely sampling. Rong *et al*. [43] and Fei *et al*. [44] implement L-BFGS methods on the GPU to obtain faster speeds for computing CVT. In general, these methods cluster point sets according to their spatial distribution, so their segmentation results are not accurate on the geometric boundary, and cannot identify the blending surface, as compared in Sec. 5.1.

Data-driven segmentation. Data-driven methods, which have advantages in discovering the geometric, structural, and semantic relationships of 3D shapes, are suitable for object segmentation. Kalogerakis *et al*. [45] first used a Conditional random field-based objective function for part segmentation. Xu *et al*. [46] and Rodrigues *et al*. [47] conducted in-depth research on this method. Furthermore, Qi *et al*. successively proposed PointNet [48] and PointNet++ [49] to directly conduct the semantic segmentation of a point cloud. Yu *et al*. [50] hierarchically divided 3D shapes into finer structural parts using PartNet. Hu *et al*. [51] proposed a multiscale framework for the semantic segmentation of the 3D point clouds of indoor scenes. Wang *et al*. [52] proposed the DGCNN that applies a new "EdgeConv" module to solve the tasks of point cloud segmentation and classification. Hanocka *et al*. [53] proposed a new neural network, called MeshCNN, that operates directly on triangular meshes, where the convolution and pooling operations are tailored to irregular and non-uniform structures. However, these methods mostly focus on high-level semantic segmentation and lack the accurate perception of geometric boundaries. Some other methods sought to learn the geometric features of CAD datasets. For example, Li *et al*. [54] introduced the Supervised Primitive Fitting Network (SPFN) to detect primitives at different scales. Sharma *et al*. [55] proposed the ParSeNet to decompose 3D point clouds into basic geometric primitives and B-spline patches. However, these methods cannot detect and fit a blending surface, which is the focus of our method.

Surface fitting. Surface fitting for 3D models has been an active area of reverse engineering and structure reconstruction. There are also many approaches that directly perform the primitive type recognition and primitive fitting of cloud point data [56], [57], [58], [59], [60], [61], which are out of the scope of this paper. We shall focus on the extraction of quadric surfaces from triangular meshes [62]. Chen and Liu [63] presented a surface extraction method based on the genetic algorithm, which can extract all types of quadric surfaces with a single surface representation. Kós *et al*. [20] recovered constant radius rolling-ball blends from a triangular mesh, but they assumed that the input has been preprocessed and segmented, making this problem much easier than that we want to solve. Ahn *et al*. [64] presented an algorithm that fits implicit surfaces and plane curves by minimizing the square sum of the orthogonal error distances between the model feature and the given data points. Kanai *et al*. [65] described an efficient method for the hierarchical approximation of implicit surfaces from polygonal meshes

using an error function. Lafarge *et al*. [11] proposed an original hybrid modeling process of urban scenes that represents 3D models as a combination of mesh-based surfaces and geometric 3D primitives, later this method is extended to be hierarchical in [2]. Li *et al*. [5] considered the problem of approximating an arbitrary generic surface with a given set of simple surface primitives. Wang *et al*. [3] presented a reconstruction method that has potential applications to a spectrum of engineering problems with impacts on rapid design and prototyping, shape analysis, and virtual reality. Based on primitive fitting, Du *et al*. [4] presented a pipeline that generates constructive solid geometry (CSG) trees from noisy input mesh models.

3 Overview

In this work, we propose segmenting and reconstructing 3D manufactured models via an MRF framework. The input to our algorithm is a two-manifold boundary mesh with triangular faces, $\mathcal{M} = (V, E, F)$, where $V = \{v_i | i = 1, ..., n\}$, E and F are the sets of vertices, edges and triangular faces, respectively. We aim to segment the input mesh into a collection of patches, $\{\mathcal{P}_i\}_{i=1}^{n}$, and find the surface that best fits each patch, especially focusing on the type of rolling-ball blending surface.

As shown in Fig. 3, our algorithm consists of three main stages. First, to improve the robustness of the segmentation, we introduce a new clustering algorithm to build mesh superfacets, which decrease the number of elements to be labeled from hundreds of thousands or even more to just a few hundred/thousand. The algorithm also eliminates the effects of noises, which usually lead to some undesirable segmentation results in previous approaches. Next, we apply a specially designed MRF segmentation method, which includes an energy function that consists of probability analysis based on a data term and continuity analysis based on a smooth term, to the mesh superfacets. Each superfacet, including the rolling-ball blending surface, is assigned a label that corresponds to its surface type by optimizing this energy function. Finally, we use an existing method [34] to fit the usual quadric surfaces, such as planes, spheres and cylinders. However, it is still challenging to recover the geometric parameters of rolling-ball blending surfaces. Therefore, we propose a new method that uses a mesh contraction method [66] to estimate the initial trajectory and initial radius of the rolling ball. Then, we iteratively update the trajectory and the rolling ball's radius by introducing a trajectory estimation equation.

4 Methodology

4.1 Mesh Superfacets

The 3D meshes reconstructed from scanned or multi-view stereo recovered point clouds always incorporate undesirable small-scale oscillation. The presence of such oscillation severely degrades the local properties of the mesh, thus having a negative effect on the mesh segmentation and reconstruction accuracy. To overcome this problem, we integrate the concept of mesh superfacets in the segmentation process to avoid the disturbance of noises contained in the input model. In addition, the elements that should be labeled during segmentation can decrease from hundreds of thousands or even more to just a few hundred/thousand, thereby greatly accelerating the remaining segmentation processes. Inspired by image superpixels, some approaches for building mesh superfacets, such as [67], [68], have already been presented. Unfortunately, these methods are not suitable for our problem because they usually generate blur and imprecise clusters on the boundaries between quadric surfaces and rolling-ball blending surfaces.

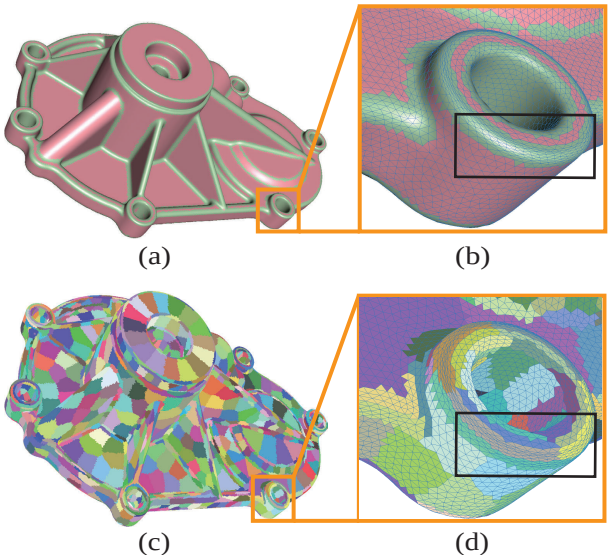

Fig. 4. Illustration of building mesh superfacets: (a) the initial MRF partitioning result of the input mesh; (b) the zoom-in view of (a); (c) the mesh superfacets; (d) the zoom-in view of the same regions with (b).

In this work, we adopt a framework similar to that in [67], but we make important modifications to the key components to make the framework more appropriate for our problem. We formulate the mesh superfacet construction as shortest path clustering that operates on the face graph of the mesh. Here, the face graph takes the triangular faces as nodes and adds an edge between two nodes if they are adjacent (*i.e.*, they share a common mesh edge) in the input mesh. As is standard with k-means style algorithms, our approach involves the following steps.

4.1.1 Initialization

Instead of directly sampling random seeds as initial cluster centers [67], we first use an MRF formulation to roughly partition the input mesh into three kinds of regions with labels $L = \{1, 2, 3\}$, as shown in Fig. 4 (top). Our motivation is that we want to place sufficient seeds in the rolling-ball blending regions and generate anisotropic-like clusters that will better align with the boundaries. We denote k_1 and k_2 as the maximum and minimum principal curvatures defined on a triangle (the local curvature is computed using the method of [69]), respectively. Then, each triangle f_i can be assigned to a surface type label by considering the curvature

distributions:

$$P_r\left(l_i \mid k_1^{(i)}, k_2^{(i)}\right) = \begin{cases} G_{\sigma_1}\left({k_1}^{(i)}\right) G_{\sigma_1}\left({k_2}^{(i)}\right) & if\ l_i = 1 \\ G_{\sigma_1}\left(1-{k_1}^{(i)}\right) G_{\sigma_1}\left({k_2}^{(i)}\right) & if\ l_i = 2 \\ G_{\sigma_1}\left(1-{k_1}^{(i)}\right) G_{\sigma_1}\left(1-{k_2}^{(i)}\right) & if\ l_i = 3 \end{cases} \quad (1)$$

where P_r expresses the probability of each label at the triangle, and $G_{\sigma_1}(k) = exp(-k^2/2{\sigma_1}^2)$ is the non-normalized centered Gaussian distributions with a standard deviation σ_1. The regions with a label of 1 do not bend too much in both principal directions. Thus, they are more likely to be quadric surfaces. The regions with a label of 2 bend a considerable amount only in one principal direction and are usually considered the rolling-ball blending surfaces. The last type of regions bend a lot in both directions and are considered irregular elements with local noises or small-scale oscillation. The labeling process is performed by minimizing the following multilabel energy function:

$$U(l) = \sum_{i \in F} D_i(l_i) + \beta_1 \sum_{\{i,j\} \in E} V_{ij}(l_i, l_j), \quad (2)$$

where $D_i(l_i)$ is the consistency term, which measures the coherence of the label l_i at the triangle f_i:

$$D_i(l_i) = 1 - P_r\left(l_i \mid k_1^{(i)}, k_2^{(i)}\right). \quad (3)$$

$V_{ij}(l_i, l_j)$ is the topological smoothness constraints balanced by a parameter β_1, and we will describe this term in detail in Subsection 4.2.2.

Finally, to obtain the initial superfacets, we use the iterative farthest point optimization strategy [67], [70] to generate random seeds among these three kinds of regions. Specifically, for each triangle in the input mesh, we define the shortest geodesic distance from it to its nearest seed and assign the corresponding seed label to it. Initially, the distance and label are set to $+\infty$ and 'undefined', respectively, because no seed is generated thus far. We first select a random triangle as a seed in each region. Then, for each triangle in the r_s-radius neighborhood of each seed, we update its shortest geodesic distance and the corresponding label. Next, from the remaining triangles with the largest geodesic distance in each region, we select a random triangle as the new seed and update the information of the triangles in its r_s-radius neighborhood. This step continues until the necessary number of seed points is generated.

4.1.2 Iterative clustering

Next, we alternately repeat the following two steps: update the superfacets' centers and reclassify triangles. In the former, we compute a new center for each superfacet by computing the Euclidean area-weighted mean of all triangle centroids that belong to that superfacet. In the classification step, we adopt Dijkstra's algorithm to compute the shortest path distance along the face graph of the mesh by taking each superfacet's center as a source. Then, the cluster ID of each triangle is updated according to its nearest superfacet's center. To compute the edge weight among adjacent triangles, the original paper [67] combines the approximate geodesic and angular terms using a weighted sum. In our approach, we define a new face graph weight as:

$$w(f_i, f_j) = \frac{geo(f_i, f_j) + \alpha \cdot ang(f_i, f_j)}{d} + \epsilon_1(f_i, f_j), \quad (4)$$

where $geo(f_i, f_j)$ is the geodesic distance between any two triangles, $ang(f_i, f_j)$ measures the angular weight, α is a balance parameter ($\alpha = 100$ in our all experiments), and d is the bounding box diagonal of the input mesh. More detailed explanations of these two items can be found in [67]. However, if we only use these two items, then the classification is unstable (the triangle labels flip-flop) near the boundaries between quadric surfaces and the rolling-ball blending surfaces, thereby resulting in the non-convergence of the clustering. To solve this problem, we define a new penalty term $\epsilon_1(f_i, f_j)$ to preserve the boundaries. In particular, if two adjacent triangles, namely, f_i and f_j have the same label, then ϵ_1 is set to a smaller value ($\epsilon_1(f_i, f_j) = 0.2 * \frac{geo(f_i, f_j)}{d}$ in our all experiments), otherwise it should be set as a relatively larger value ($\epsilon_1(f_i, f_j) = 1.2 * \frac{geo(f_i, f_j)}{d}$).

Fig. 4 (c) shows an example of our mesh superfacets, where adjacent triangles with the same color are grouped into one superfacet. Given the high similarity in the geometric properties of the triangles that belong to the same superfacet, each superfacet can be used as an individual elementary unit in the next segmentation step. In addition, the biggest advantage of our approach is that the boundaries we achieved accurately separate the regions that belong to quadric surfaces or rolling-ball blending surfaces, which is still an unsolved issue in other superfacet algorithms. Fig. 4 (d) demonstrates that our approach yields a good partition between rolling-ball blending patches and quadric surface patches in the black box while keeping the boundaries of adjacent superfacets clean.

4.2 Mesh Segmentation

After building mesh superfacets, we define a new face graph $\mathcal{G} = \{C, E^{'}\}$ whose nodes C are the collection of superfacets and edges $E^{'}$ represent the adjacent-relationships between the nodes. We consider two superfacets to be adjacent if they share at least one triangle edge. In this section, we aim to segment the original mesh $\mathcal{M}$ into meaningful patches to conform to geometric shape constraints by performing MRF labeling on the new face graph $\mathcal{G}$.

4.2.1 Geometric properties of superfacets

To adopt the MRF framework, we first assign the geometric properties based on curvature analysis to each superfacet. It should be noted that a superfacet may contain triangles with different labels that indicate their surface types, as described in Sec. 4.1.1. To disambiguate the noisy local cues, we collect triangles (denoted as a set T) that belong to the class with the maximum number of triangles. Then, we denote the geometric properties of this superfacet as the mean of the geometric properties of these triangles. Specifically, we compute the average maximum and minimum principal curvatures K_1 and K_2, respectively, and their associated average direction vectors $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$, respectively. In addition, we also compute K_3, which is the variance of k_1 over all triangles in T; and $K_4 = K_1 - K_2$, which measures

the difference of two principal curvatures among the same superfacet.

4.2.2 MRF refined segmentation on superfacets

As mentioned above, we have partitioned the mesh into three kinds of regions: quadric, rolling-ball blending, and irregular patches. To further identify the specific type of quadric surfaces for each superfacet, the curvature property is used to label each superfacet according to the four types of shapes that are common in most mechanical objects and urban landscapes. First, a *developable surface* ($K_1K_2 = 0$) can be easily distinguished from a *non-developable surface* ($K_1K_2 \neq 0$) according to the Gaussian curvature. Among the developable surfaces, a plane is identified if $K_1 = K_2 = 0$ but differs from a cylinders and cones. To distinguish a cylinder from a cone, we need to examine the extent of the changes of K_1 in a local neighborhood. If the extent is small, then this superfacet should be part of a cylinder; otherwise, it should be a part of a cone. For a non-developable surface, we consider two cases: a sphere ($K_1 = K_2 > 0$) and other quadric surfaces (*e.g.*, two-radius-alike paraboloid or hyperboloid).

Now we define the label set $L^{'} = \{1, 2, 3, 4, 5\}$, in which the elements correspond to *planes, spheres, cylinders, other quadric surfaces, and cones*, respectively. The probability of a superfacet c_i is assigned a label l_i can be expressed as a combination of the curvature distributions:

$$Pr\left(l_i \mid K_1^{(i)}, K_3^{(i)}, K_4^{(i)}\right) = \begin{cases} G_\sigma\left(K_1{}^{(i)}\right) G_\sigma\left(K_3{}^{(i)}\right) G_\sigma\left(K_4{}^{(i)}\right) & if\ l_i = 1 \\ G_\sigma\left(1 - K_1{}^{(i)}\right) G_\sigma\left(K_3{}^{(i)}\right) G_\sigma\left(K_4{}^{(i)}\right) & if\ l_i = 2 \\ G_\sigma\left(1 - K_1{}^{(i)}\right) G_\sigma\left(K_3{}^{(i)}\right) G_\sigma\left(1 - K_4{}^{(i)}\right) & if\ l_i = 3 \\ G_\sigma\left(1 - K_1{}^{(i)}\right) G_\sigma\left(1 - K_3{}^{(i)}\right) G_\sigma\left(K_4{}^{(i)}\right) & if\ l_i = 4 \\ G_\sigma\left(1 - K_1{}^{(i)}\right) G_\sigma\left(1 - K_3{}^{(i)}\right) G_\sigma\left(1 - K_4{}^{(i)}\right) & if\ l_i = 5 \end{cases} \tag{5}$$

Again, we formulate the MRF labeling as the following energy function to predict the most probable label:

$$U^{'}(l) = \sum_{i \in C} D_i^{'}(l_i) + \beta \sum_{\{i,j\} \in E'} V_{ij}^{'}(l_i, l_j), \tag{6}$$

where the complete consistency term $D_i(l_i)$ is defined as:

$$D_i^{'}(l_i) = 1 - Pr\left(l_i \mid K_1^{(i)}, K_3^{(i)}, K_4^{(i)}\right). \tag{7}$$

Here, our consistency term is considered complete because the probability space of the five labels is complete, that is, $\sum_{i=1}^{5} Pr(l_i) = 1$. This is also the reason why we introduce the category of *other quadric surfaces*, which are unusual in mechanical objects.

To prevent the sudden change in the label in a small area that may cause noisy patches to the greatest extent, a topological smoothness constraint $V_{ij}^{'}(l_i, l_j)$ is proposed as:

$$V_{ij}^{'}(l_i, l_j) = \begin{cases} 1, & if\ \{l_i \neq l_j\} \\ min\left(1, \xi \left\|\boldsymbol{W}_i - \boldsymbol{W}_j\right\|_2\right), & otherwise \end{cases} \tag{8}$$

For each $\boldsymbol{W}$, it is given by:

$$\boldsymbol{W} = \begin{pmatrix} K_1 \cdot \boldsymbol{W_1} \\ K_2 \cdot \boldsymbol{W_2} \end{pmatrix}, \tag{9}$$

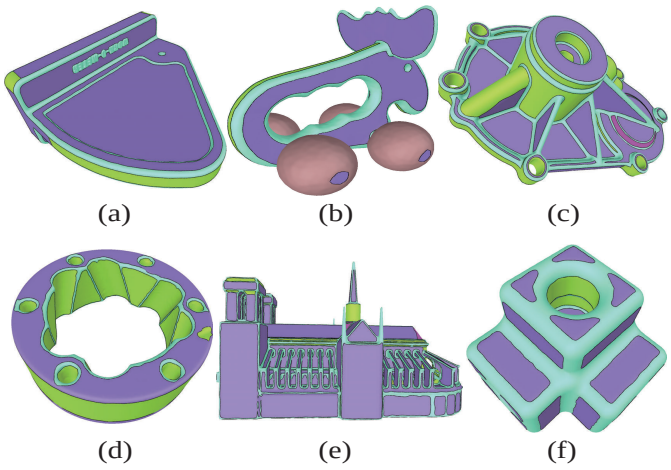

Fig. 5. Segmentation examples by our labeling approach. For each model we show its respective partitioning (please refer to Fig. 3 for the interpretation of the color).

where $\xi > 0$ is a parameter proportional to the average edge length of the mesh $\mathcal{M}$. By designing $V_{ij}^{'}(l_i, l_j)$, we provide each pair of adjacent elements that correspond constraints according to two different label cases ($l_i \neq l_j$ or $l_i = l_j$). To make the labels in a local area as similar as possible, we need to give a stronger penalty when $l_i \neq l_j$. In another case, for adjacent elements that are judged to have $l_i = l_j$ by the consistency term, if their geometry properties (*e.g.*, combination of K_1, K_2, $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$) differ greatly, we also penalize them. Therefore, this constraint will preserve the evident boundaries among different patches. Fig. 5 shows our final labeling results after energy minimization. Benefitting from the superfacets and topological smoothness constraints in our MRF formulation, we can eliminate the noisy patches that would often occur if we directly apply MRF labeling to the input mesh. Furthermore, given that we have an initial segmentation in Sec. 4.1, a strong smoothness concern is not needed in the current step, so the balanced parameter β can be set to a small value (0.8 by default) that could preserve more details.

It should be noted that after the aforementioned segmentation, different patch instances with the same label (*i.e.*, they belong to the same surface type) may be connected. For example, the cylinder patches encoded by the lawngreen color in Fig. 3 (c) should be separated into different cylinder instances, as shown in Fig. 3 (e). In our approach, we use the connectivity information, dihedral angles and curvature difference between the superfacets to separate different quadric patch instances. Finally, due to mesh discretization, patch boundaries are usually not smooth (jagged edges are observed on boundaries). As a postprocessing step, we use a boundary curve smoothing method presented by [71] to smooth the boundaries.

4.3 Geometric Surface Fitting

To conduct surface structure recovery, we provide a corresponding geometric primitive for each segmented patch. For the quadric surface type, each patch will be compared with a set of 3D primitives composed of planes, spheres, cylinders and cones. To detect such shapes, we apply the fitting

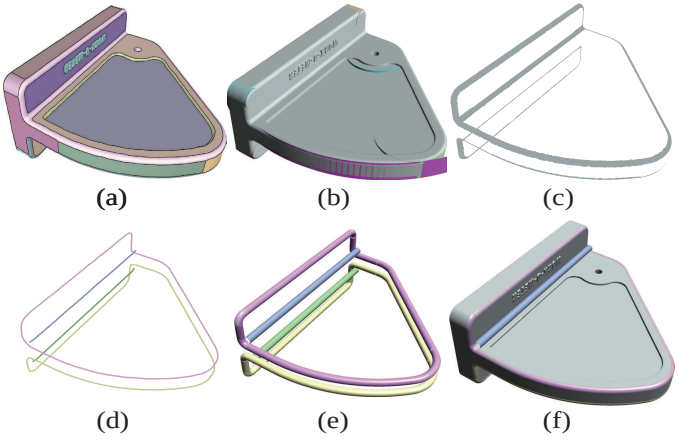

Fig. 6. Surface fitting: (a) our segmentation result with the rolling-ball blending patches colored in pink; (b) examples of quadric surfaces fitting; (c) the segmented rolling-ball blending patches; (d) the curve skeleton extracted by [66]; (e) recovered rolling center trajectories with accurate ball radii; (f) the fitted rolling-ball blending surfaces are overlaid on the input model.

algorithm introduced by [34], which proposes a general variational mesh segmentation framework by fitting quadric surfaces. In Fig. 6 (b), we could recover the quadric surface structures precisely. Unfortunately, this variational method can not obtain good performance on rolling-ball blending patches. In this section, we present a new method based on skeleton extraction to fit rolling-ball blending patches by rebuilding the rolling center trajectories and recovering the ball radius.

4.3.1 Rolling-ball blending surface fitting

We denote a rolling-ball blending patch as $P = (\boldsymbol{c}(t), r)$, where $\boldsymbol{c}(t)$ is the trajectory of the rolling ball's center and r is the ball's radius. However, fitting a rolling-ball blending patch is quite complicated; and to the best of our knowledge, no closed-form solution is available. To fit such a blending patch, we present an iterative optimization algorithm that consists of three steps: 1) trajectory initialization, 2) update the ball radius, and 3) update the trajectory. Here, steps 2) and 3) are alternately repeated until convergence is achieved.

Trajectory initialization. Obtaining a good initial position for trajectory would accelerate the overall fitting process. In 3D shape analysis, a skeleton is a thin centered structure that provides a compact and intuitive abstraction to capture the topology and the geometry of a shape. Inspired by this observation, we take the 3D skeleton as our initial trajectory. For stability purposes, we use the mesh contraction method [66] to extract the curve skeletons from the triangles of the patch. The curve skeleton is represented as $\boldsymbol{c}(t) = (\boldsymbol{U}, \boldsymbol{B})$ with skeleton nodes $\boldsymbol{U}$ and skeleton edges $\boldsymbol{B}$. We separate the trajectory $\boldsymbol{c}(t)$ into different sub-trajectories $\{\boldsymbol{c}_j(t)\}$ by checking the deviation of the angle between each pair of connected edges in $\boldsymbol{B}$ because a big change is usually found on the trajectory when the ball radius changes. Next, by finding the nearest skeleton edge for each triangle of the patch P, we also partition the rolling-ball blending patch into a set of sub-patches $\{P_j\}$.

Update the ball radius. After building the correspondence between $\boldsymbol{c}_j(t)$ and P_j, we compute the Euclidean distance (denoted as triangle-trajectory distance) between each triangle of P_j and the related edge in $\boldsymbol{c}_j(t)$. We set the radius r_j of the ball that rolls over $\boldsymbol{c}_j(t)$ to the mean of these distances. Then, we check if this new ball radius is different from the previous one. If no radius is changed, then the fitting process terminates; otherwise, we will continue to update the trajectories.

Update the trajectory. For each trajectory $\boldsymbol{c}_j(t)$, we optimize the position of its nodes to make each triangle-trajectory distance of P_j as close as possible to the ball radius obtained in the aforementioned step. To control the smoothness of the trajectory, our energy function is formulated using a data term and a smoothness constraint:

$$\begin{aligned} E_j = &\sum_{i=0}^{m_j-2}\sum_{k=0}^{n_j-1}\left|\sqrt{{A_{ik}}^2+{B_{ik}}^2+{C_{ik}}^2}-r_j\right| \\ &+\beta_2\sum_{i=1}^{m_j-2}\sqrt{{D_i}^2+{E_i}^2+{F_i}^2}, \end{aligned} \tag{10}$$

where m_j is the number of nodes in $\boldsymbol{c}_j(t)$, n_j is the number of triangles in P_j and β_2 is a balance parameter ($\beta_2 = 0.4$ in default). In this function, the first term calculates the sum of the errors between the triangle-trajectory distances and current ball radius r_j, whereas the second smoothness constraint calculates the sum of the distances from each node in U_j to the midpoint of the line between its preceding and successive nodes. The explanation for other variables is described as follows:

$$\begin{aligned} A_{ik} &= a_k - p_i\frac{p_i(a_k-x_i)+q_i(b_k-y_i)+w_i(c_k-z_i)}{p_i^2+q_i^2+w_i^2} - x_i, \\ B_{ik} &= b_k - q_i\frac{p_i(a_k-x_i)+q_i(b_k-y_i)+w_i(c_k-z_i)}{p_i^2+q_i^2+w_i^2} - y_i, \\ C_{ik} &= c_k - w_i\frac{p_i(a_k-x_i)+q_i(b_k-y_i)+w_i(c_k-z_i)}{p_i^2+q_i^2+w_i^2} - z_i, \\ D_i &= x_i - (x_{i+1}+x_{i-1})/2, \\ E_i &= y_i - (y_{i+1}+y_{i-1})/2, \\ F_i &= z_i - (z_{i+1}+z_{i-1})/2, \\ p_i &= x_{i+1} - x_i, \\ q_i &= y_{i+1} - y_i, \\ w_i &= z_{i+1} - z_i, \end{aligned} \tag{11}$$

where (x_i, y_i, z_i) is the coordinate of a node in $\boldsymbol{c}_j(t)$, (a_k, b_k, c_k) is the coordinate of a triangle centroid in P_j. Once the energy is below a pre-defined threshold, $\boldsymbol{c}_j(t)$ is updated.

Finally, we fit a B-spline curve to the nodes in $\boldsymbol{c}_j(t)$ to obtain a smoother trajectory and set the final radius of the blend patch to the last ball radius r_j. Here, we use the nodes in the sub-trajectory as control points and obtain a cubic B-spline curve through interpolation. Figs. 6 (c) to (f) show an example of fitting rolling-ball blending patches that comprise four rolling trajectories. Note we cannot guarantee that the mesh contraction [66] could provide a good starting point, as well as the iterative optimization steps could converge. However, since each rolling-ball blending patch

is on the same side of the ball's rolling direction and it has a roughly constant width with a similar bending degree, the mesh contraction can easily extract the skeleton that sits off the patch close to the center of the rolling ball. Therefore, it is effective to use the mesh contraction to provide the trajectory initialization. In practice, our proposed approach works well for all of the examples shown in the experiments.

5 Experimental Results

Our approach was tested on a number of 3D mesh models with different complexities, including CAD components and architectural models. Some of the models were downloaded from the Shape Repository of Digital Shape Workbench [1] and the Thingi10K dataset [72]. We evaluated our algorithm qualitatively and quantitatively through visually inspecting our results and conducted a complete comparison with state-of-the-art approaches. We also show an application where our method was used for mesh blending by fitting the rolling-ball blending surfaces. Our algorithm is implemented in C++ and relies on the CGAL Library [73] to compute the geometric attributes. All results presented in this study are obtained on a desktop computer equipped with an Intel i7-7700k processor clocked at 4.2GHz Ghz and 16 GB of RAM.

5.1 Evaluation

To demonstrate the effectiveness of the proposed approach, we first conducted experiments on several technical CAD models. Fig. 7 shows the segmentation results, where the zoom-in views indicate the detailed parts we are most interested in. For all of the five CAD mesh models, our method can segment the quadric surface patches accurately, whereas the rolling-ball blending patches are partitioned into new classes. In addition, undesirable noises or small-scale oscillations are avoided in the results, and the geometric details of the surface structure could be preserved.

Ablation study. Next, we perform an ablation study to evaluate the effects of the number of superfacets on segmentation results. Let N be the number of triangles of the input mesh, the number of superfacets is approximately set to $s \approx \frac{N}{\lambda}$, where λ is called down-sampling scale. Fig. 8 compares the segmentation results by using different down-sampling scale values in building mesh superfacets. In Fig. 8 (a) we directly perform segmentation on the input mesh without superfacets. We see that the segmentation is fuzzy, and many fragmentary patches exist because the presence of geometric noises degrades the local attributes of the triangles, thereby causing difficulty in recovering the correct label. Figs. 8 (c) to (e) illustrate the segmentation results where λ is set to 50, 100, 150, respectively. As the number of superfacets increases, more details are identified correctly because more elements represent the microstructure. However, the computation time needed in this clustering step also increases. Moreover, the segmentation accuracy is not always positively correlated to the number of mesh superfacets. In Fig. 8 (b), when we use a smaller down-sampling scale ($\lambda = 20$), the segmentation has a worse boundary adherence than Fig. 8 (c). According to our extensive experiments, we recommend $\lambda = 100$ as an appropriate value for achieving a good trade-off between the segmentation accuracy, detail recovery, and computational cost. All examples presented in this paper are obtained with $\lambda = 100$.

1. http://visionair.ge.imati.cnr.it/ontologies/shapes/

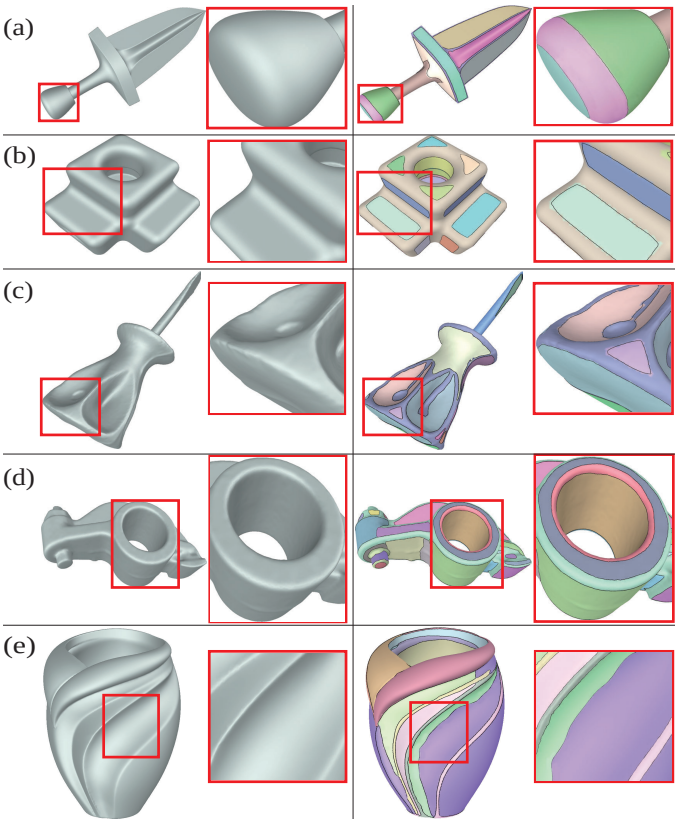

Fig. 7. A gallery of examples generated by our framework. For each model, we show the input CAD geometry, segmentation result, and zoom-in views.

We also compared our superfacet construction method with the CVT-driven approach [41]. In Fig. 9, we generated different numbers of CVT superfacets and fed them into our geometric surface segmentation stage. It can be seen that as the number of CVT clusters increases, more details are provided in the segmentation results. However, because the shape and size of CVT superfacets are very uniform, the boundary of each cluster cannot match the boundary of the blending surface. This leads to the serious destruction of the boundary of the blending surface in the segmentation.

Evaluation on noisy models. We further evaluated our method by using two noisy models. The segmentation results are shown in Fig. 10, in which we randomly added Gaussian noise to mesh vertices with a standard deviation of 0.03%, 0.06%, and 0.1% (see Fig. 10 (a), (b) and (c)) of the length of the bounding box diagonal. Compared to the approach without using superfacets, we have good robustness to noise since some small bumps and depressions are eliminated after constructing the superfacets. It should be noted that overcoming the noise influences the global constraints; thus, the patch boundaries may be insufficiently segmented.

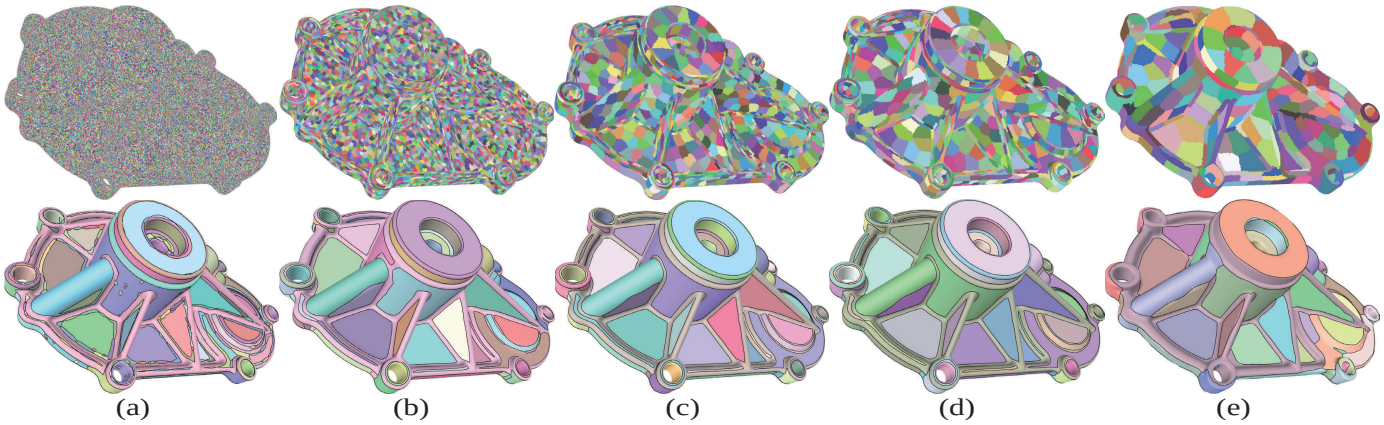

Fig. 8. Comparison of the segmentation results (bottom) by using different settings of the down-sampling scale (λ) when building mesh superfacets (top). From left to right: segmentation without superfacets which generates 678 patches (a), segmentation using $\lambda = 20$ which generates 135 patches (b), $\lambda = 50$ which generates 121 patches (c), $\lambda = 100$ which generates 116 patches (d), $\lambda = 150$ which generates 96 patches (e).

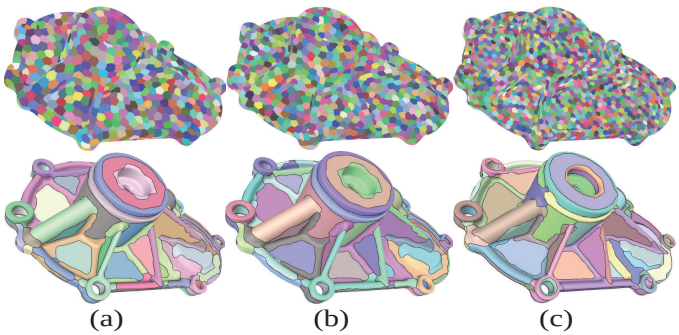

Fig. 9. Segmentation results (bottom) by using different settings of the down-sampling scale (λ) when computing the number of CVT seed points (top). From left to right: λ = 150 (a), λ = 100 (b), λ = 50 (c).

5.2 Comparisons

We now compare our approach with three representative quadric-fitting-based segmentation methods: a variational shape approximation (VSA) approach [1], a quadric surface extraction (QSE) approach [34], and an MRF-based hybrid model (Hybrid) [2].

Fig. 11 shows a complete visual comparison of experiments on five models, including two CAD mechanical components, a furniture model, and two buildings. We also display the detailed comparison by using zoom-in views. The VSA approach [1] utilizes the Lloyd algorithm to segment the model by minimizing a total approximation error iteratively. However, considering that VSA only fits planar proxies, it's structure recovery capability is limited, thereby resulting in unfavorable results on the curved surface regions(11). Evidently, VSA also cannot partition the blending regions. As an extension of VSA, QSE [34] employs general quadric surface fitting for segmenting the models. It can partition quadric regions correctly by approximating all kinds of primitives (*e.g.*, plane, sphere, cylinder, cone, etc.). However, the segmentation quality is heavily dependent on the seeds' initialization, and the results are not satisfactory as the number of proxies declines. In addition, it cannot segment rolling-ball patches. Even worse, when encountering a model with blending regions, the error accumulation also affects the segmentation accuracy of other quadric patches.

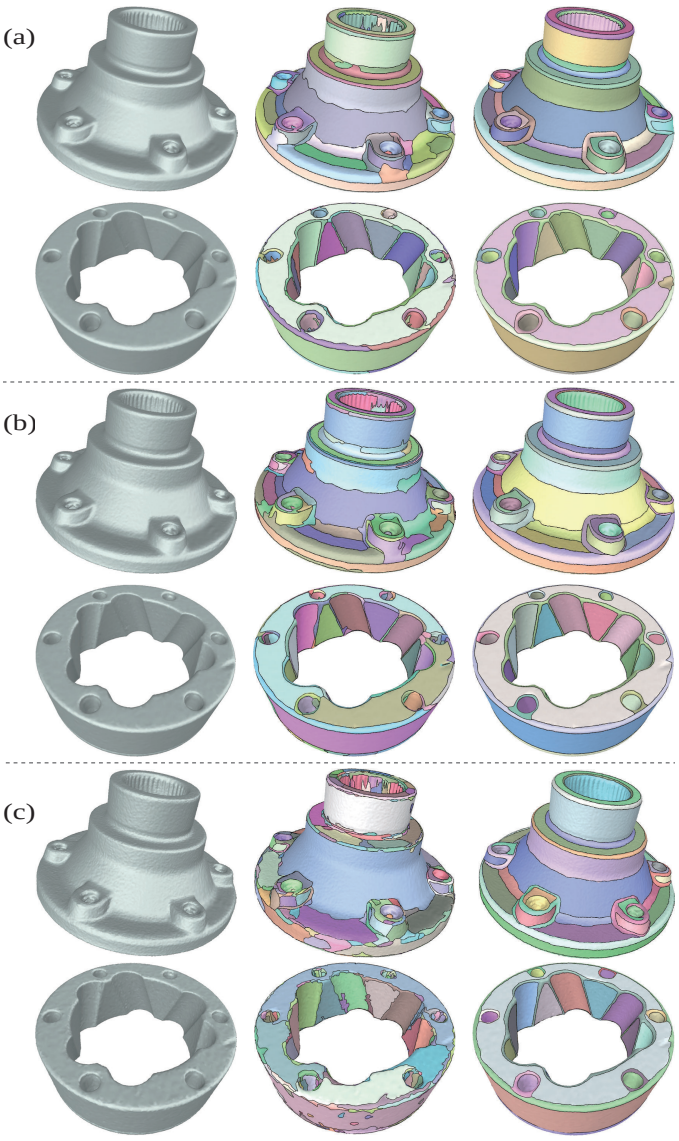

Fig. 10. Comparison of segmentation results on two noisy models (left) without using (middle) and using (right) superfacets. The noisy level in each model is 0.03% (a), 0.06% (b), and 0.1% (c), respectively.

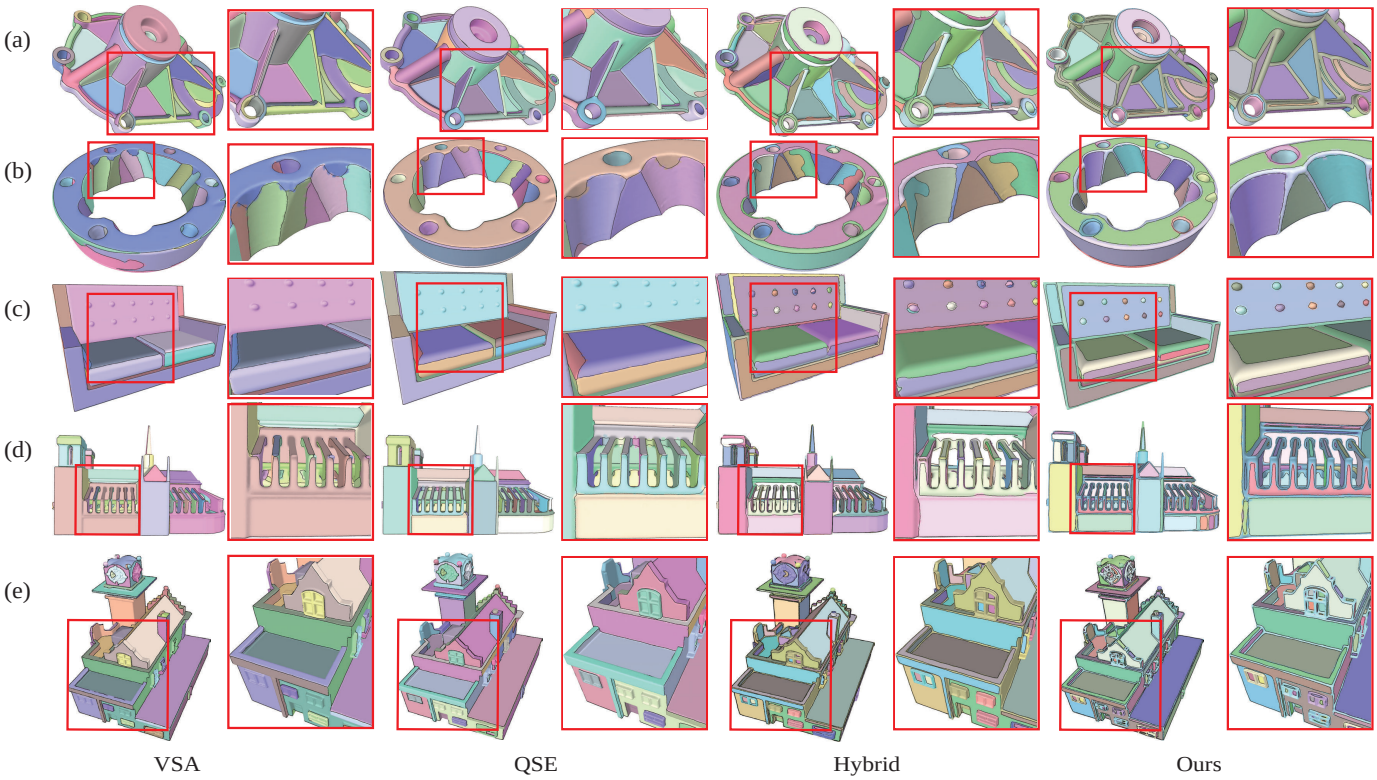

Fig. 11. Comparison with previous methods on different models. From left to right are results of VSA [1], QSE [34], Hybrid [2], and our approach. We also provide their zoom-in views.

The Hybrid approach [2] proposes a hierarchical framework in which the framework first performs a pre-segmentation on each layer using the MRF energy function. Then a tentative surface fitting is performed based on the previous segmentations. The result of each pre-segmentation can guide the sequence of the subsequent surface fitting directly. Fig. 11 shows that the pre-segmentation using MRF enables the algorithm to segment non-developable patches, but due to the lack of special consideration, it still cannot segment all of the blending regions accurately. In addition, some over/under-fitting phenomena occur for the segmentation of the quadric regions because of the limitation of initialization and error thresholding in the fitting process. As a comparison, we could obtain a cleaner and more accurate partition of the model, especially focusing on the extraction of the blending regions, by building the mesh superfacets and designing the segmentation process carefully. Although Wu and Kobbelt [10] introduced a heuristic approach to segment and fit rolling-ball blending patches, this approach is non-trivial and often requires user intervention to obtain the optimal result. In addition, finding the blend regions is difficult for complex models and the blend patch fitting procedure is also quite slow. In Fig. 12, we visually compare our approach with [10] on the rocker arm model. Reference [10] approximated the model with 30 surface proxies, whereas our segmentation automatically divides it into 37 surface proxies. Our method extracts the rolling-ball blending surfaces more accurately, as well as retain more details for the segmentation of other quadric surfaces.

We also report numerical statistics about the fitting error and running time for each method in Table 1. It is worth mentioning that we extended the definition of hybrid distance presented by [34], and introduced the calculation of the distance between the rolling-ball blending surfaces and the input mesh. The quantitative results show that our segmentation results are significantly improved in terms of accuracy compared with other methods. The computa-

TABLE 1
Numeric statistics comparison of different methods. $|f|$ is the number of triangles in each model, "Blend" indicates the ability to segment rolling-ball blending regions, $|P|$ is the number of segmented patches, and $|E|$ is the hybrid distance between the fitting surfaces and the input mesh, and T is the running time, respectively .

Model	$\|f\|$	Alg.	Blend	$\|P\|$	$\|E\|$	T
Fig. 11 (a)	241K	VSA	No	86	0.2923	**8.609**
		QSE	No	82	0.2068	23.860
		Hybrid	No	133	0.1945	18.032
		Ours	**Yes**	**116**	**0.1164**	15.845
Fig. 11 (b)	100K	VSA	No	44	0.1879	**15.709**
		QSE	No	35	0.1493	44.876
		Hybrid	No	59	0.2670	37.633
		Ours	**Yes**	**37**	**0.1169**	29.518
Fig. 11 (c)	159K	VSA	No	42	0.2107	**15.213**
		QSE	No	42	0.1695	43.344
		Hybrid	No	112	0.1563	36.673
		Our	**Yes**	**91**	**0.0834**	22.795
Fig. 11 (d)	393K	VSA	No	126	0.4777	**18.735**
		QSE	No	133	0.2939	46.998
		Hybrid	No	399	0.2407	40.441
		Ours	**Yes**	**473**	**0.1701**	33.113
Fig. 11 (e)	388K	VSA	No	110	0.3072	**15.394**
		QSE	No	104	0.3178	43.681
		Hybrid	No	671	0.3083	37.235
		Ours	**Yes**	**848**	**0.2250**	28.921

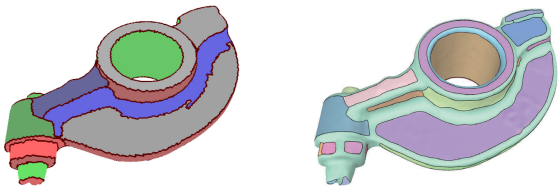

Fig. 12. Segmentation of Wu and Kobbelt [10] (left) and ours (right).

tion time of our method is faster than QSE and Hybrid but slower than VSA. However, the VSA segmentation has the worst segmentation and approximation capabilities. Therefore, our approach is the best in terms of the overall segmentation quality and computational cost. At the same time, because specifying the number of seed points in the fitting procedure is not needed, more accurate results are obtained in terms of the patch number.

Finally, we compare with four latest learning-based segmentation methods: DGCNN [52], MeshCNN [53], SPFN [54], and ParSeNet [55]. Among them, MeshCNN operates directly on triangular mesh edges, while other methods adopt point clouds as input. As suggested in their papers, we used the processed ANSI 3D dataset [54] to retrain SPFN and used the ABC dataset [74] to retrain DGCNN and ParSeNet. Besides, due to the introduction of labeling errors on patch boundary edges at the training dataset construction stage, we used the pre-trained model of MeshCNN to segment our models instead of re-training MeshCNN using the ABC dataset. Finally, since the output labels of these methods are corresponding to point clouds or mesh edges, we mapped their segmentation results back to the input mesh. As shown in Fig. 13 (a), the comparison result indicates that these methods cannot get effective segmentation results even for simple data, especially in the blending areas. The reason might be the lacking of a sufficient number of rolling-ball blending surfaces with appropriate labels in the training data. Additionally, these networks' ability to classify points is greatly affected by the complex model containing multiple patches, and their segmentation results are not as good as ours, as shown in Fig. 13 (b) and (c). Compared with these approaches, our method not only obtains more satisfactory segmentation results but also obtains accurate blending surface patches.

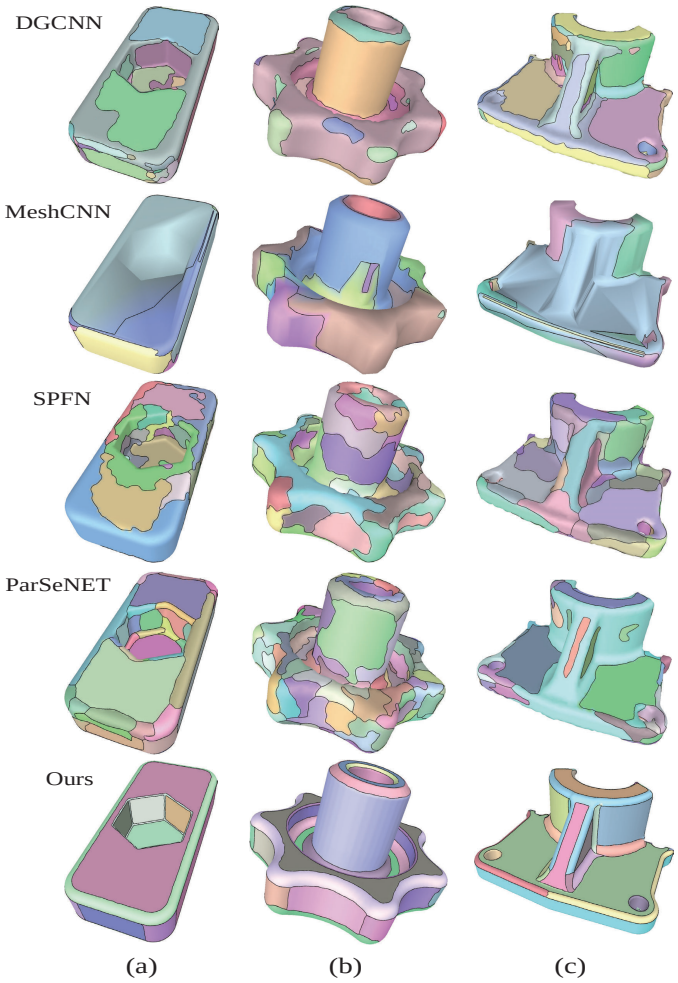

Fig. 13. Comparison with deep learning methods on different models. From top to bottom are results of DGCNN [52], MeshCNN [53], SPFN [54], ParSeNet [55] and our approach.

5.3 Applications

Based on our segmentation and fitting method for the rolling-ball blending patches, we propose an application to control the blending result by specifying different blending radii. In particular, for a rolling-ball blending surface $P = (\boldsymbol{c}(t), r)$, we assign a user-specified $r^{'}$ to the ball rolling on the trajectory $\boldsymbol{c}(t) = (\boldsymbol{U}, \boldsymbol{B})$ by optimizing the position of nodes in $\boldsymbol{U}$ while maintaining the tangency between the ball and its two adjacent quadric surfaces. Thus we could obtain a new rolling-ball blending surface $P^{'} = (\boldsymbol{c}^{'}(t), r^{'})$. To optimize the position of the nodes in $\boldsymbol{U}$, we apply an iterative optimization method where the energy function is given by:

$$E^{'}_{blending} = \sum_{i=0}^{m}(\Big|d(u_i, S_1) - r^{'}\Big| + \Big|d(u_i, S_2) - r^{'}\Big|), \quad (12)$$

where m is the number of nodes u_i in $\boldsymbol{U}$, and S_1 and S_2 are two quadric surfaces that are connected by the rolling-ball blending surface. We take $P = (\boldsymbol{c}(t), r)$ as the initial position of the new trajectory $P^{'}$. For all node u_i on P, we calculate the sum of the shortest distance d^i to their two adjacent surfaces S_1 and S_2. Our energy function $E^{'}_{blending}$ accumulates all the errors between each d^i and the radius $r^{'}$. By minimizing the energy $E^{'}_{blending}$, the ball is always tangent to S_1 and S_2 on the new nodes $\boldsymbol{U}^{'}$.

To illustrate the optimization process remarkably, we provide three simple representative examples in Fig. 14, where the 2D cross-sectional views are presented. In Fig. 14 (a), the rolling-ball blending surface connects an approximate plane S_1 and an approximate S_2 curved surface (*e.g.*, cylinder, cone, or sphere). If we make the ball radius larger which means $r = R_1$, $r^{'} = R_2$, then $d^i = \sum_{i=0}^{m}(2R_1^i + R_3^i)$. Fig. 14 (b) shows the rolling-ball blending patches that connect two approximate planes. Similarly, if we increase the ball radius, then $d^i = \sum_{i=0}^{m}(2R_1^i)$. Finally, in Fig. 14 (c), S_1 and S_2 are two approximate curved surfaces, and we compute $d^i_j = \sum_{i=0}^{m}(2R_1^i + R_3^i + R_4^i)$ to obtain a larger radius. For all these three cases, as the energy gradually decreases, the center of the ball moves along the red line

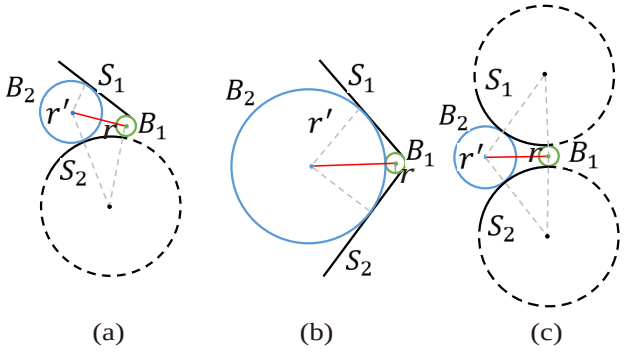

Fig. 14. 2D illustrations for changing the blending radii from B_1 to B_2: (a) S_1 is an approximate plane and S_2 is an approximate curved surface (cylinder/cone/sphere); (b) S_1 and S_2 are both approximate planes; (c) S_1 and S_2 are both approximate curved surfaces.

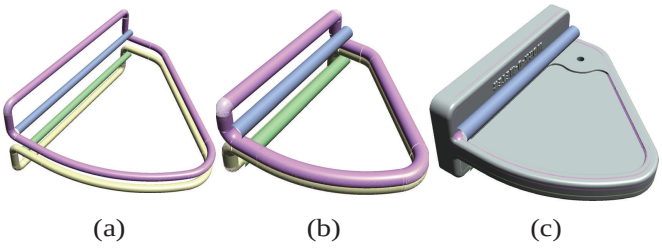

Fig. 15. Mesh editing by constructing smoother rolling-ball blending surfaces. (a) the original rolling-trajectories; (b) constructing the rolling-ball blending surfaces with two times larger blending radius, and they are overlaid on the input mesh (c).

from O_1 to O_2, that is, the green circle moves toward the blue circle. Fig. 15 shows a 3D case in which the ball radius has been doubled. Based on the new trajectory $\boldsymbol{c}^{'}(t)$, we can flexibly add more complicated intermediate blending surfaces into the input model by using the approach presented by Liu *et al*. [19].

To generate the final mesh model, we use the idea of [10] to sample a point cloud from the fitting surfaces and then generate a new mesh based on it, and we call the points in this point cloud "key points". Specifically, for planes and other general quadric surfaces, we directly project the corresponding mesh vertices to them to obtain key points. For the rolling-ball blending surface and the two primary surfaces adjacent to it, we designed a new method to generate or remove key points. For example, in the case that the radius of the rolling-ball changes from large to small (Fig. 16 (a)), we first remove the points on the original blending surface B_1 and sample along the direction in which the blending radius decreases ($r_t = r - \frac{t\left(r-r^{'}\right)}{10}, t = 0, 1, ..., 9, 10$, where the number of sampling steps is set to 10) to obtain the connecting points (key points) between the rolling-ball blending surface and the primary surfaces. Then, we sample the specified new rolling-ball blending surface B_2 to obtain the corresponding key points. Similarly, if the radius of the rolling-ball changes from small to large (Fig. 16 (b)), we remove the points on the original rolling-ball blending surface B_1, and some key points on the primary surfaces along the direction of the blending radius increase (by searching the neighboring points of the points connecting primary surfaces and the potential rolling-ball blending surface). Then, we sample the key points on the new rolling-ball blending surface B_2. Finally, we use the point cloud composed of the key points to generate the new mesh model. Next, we compare our method to an approach called Sharpen&Bend [75], which can also modify the transition regions. In [75], an EdgeSharpener filter recovers the sharp features by dividing the chamfered edges and restoring a piecewise linear approximation of the sharp edges. A Bender filter is proposed to smooth the surface and preserve the sharpness of the features while bending their polyline approximations into smooth curves. Fig. 17 shows the comparison result. The Bender in [75] can make the rolling-ball blending regions smoother, but the rolling-ball radius cannot be determined and modified. Furthermore, to obtain the sharp edges from blended regions (similar to the sharpening effect of our method), the EdgeSharpener requires a preprocessing step using the marching-intersections algorithm [76] to first reconstruct a simplified mesh. Thus, it cannot handle high-resolution meshes. By contrast, our method can change the rolling-ball blending surface by using a unified framework to modify the rolling-ball radius.

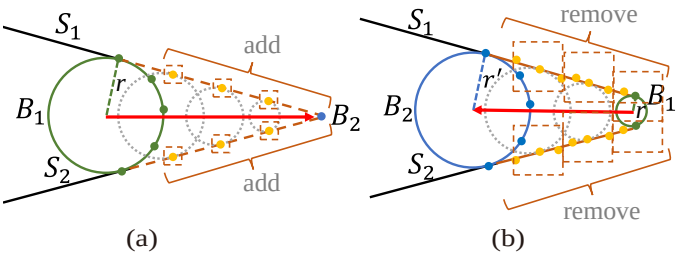

Fig. 16. 2D illustrations for key points extraction on the rolling-ball blending area: (a) generate key points on the rolling-ball blending surface and its adjacent primary surfaces; (b) generate key points on the rolling-ball blending surface while removing some key points on its adjacent primary surfaces.

In Fig. 18, we verified the effectiveness of our approach by considering more complex models. For each model, we show the original mesh, smoothing result, and sharpening result, as well as the zoom-in view details. To perform the smoothing operation, we set the blending radii in the model (a) twice that of the original rolling-ball blending patch and set the radius of the inner rolling-ball in the model (b) to be reduced by 0.8 times, and set the radius of the inner rolling-ball in the model (c) to be increased by 1.8 times. By sharpening, we perform the opposite operation to smoothing, where the rounded features can be sharpened by setting the new rolling-ball radius $r^{'} = 0$. The experimental results in Fig. 18 demonstrate our ability to control the blending shape.

Fig. 19 shows another mesh modeling application where we can create blending surfaces on sharp models or change blending surfaces with varying radii. By extending the energy function in Eq. 12, we define a new energy $E^{''}_{blending}$ as:

$$E^{''}_{blending} = \sum_{i=0}^{m}(\left|d(u_i, S_1) - r^{'}_{u_i}\right| + \left|d(u_i, S_2) - r^{'}_{u_i}\right|), \quad (13)$$

where $r^{'}_{u_i}$ means the desired local ball radius at each node u_i in $\boldsymbol{U}$. Take the input sharp model (the top row of Fig. 19

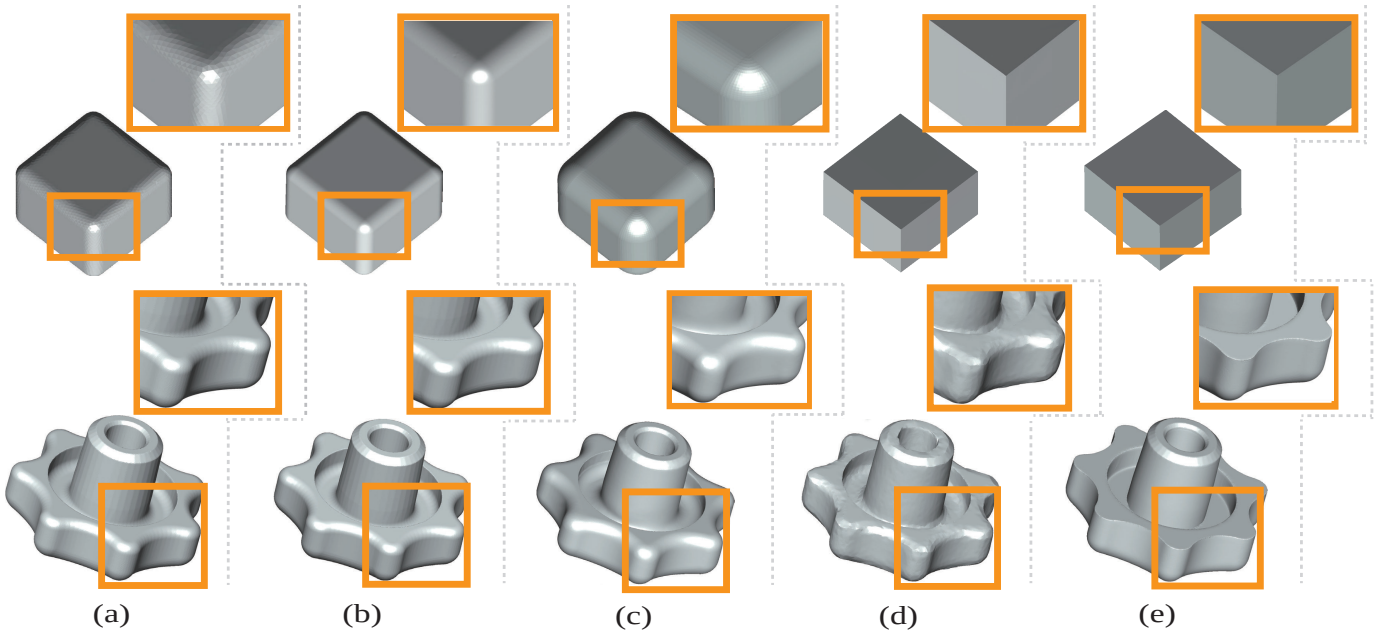

Fig. 17. Comparison to Sharpen&Bend [75]. From left to right are the input model (a), smoothing result using [75] (b), our smoothing result via adopting 2 times (top) and 1.5 times (bottom) larger radius (c), sharpening result using [76] and [75] (d), our sharpening result (e).

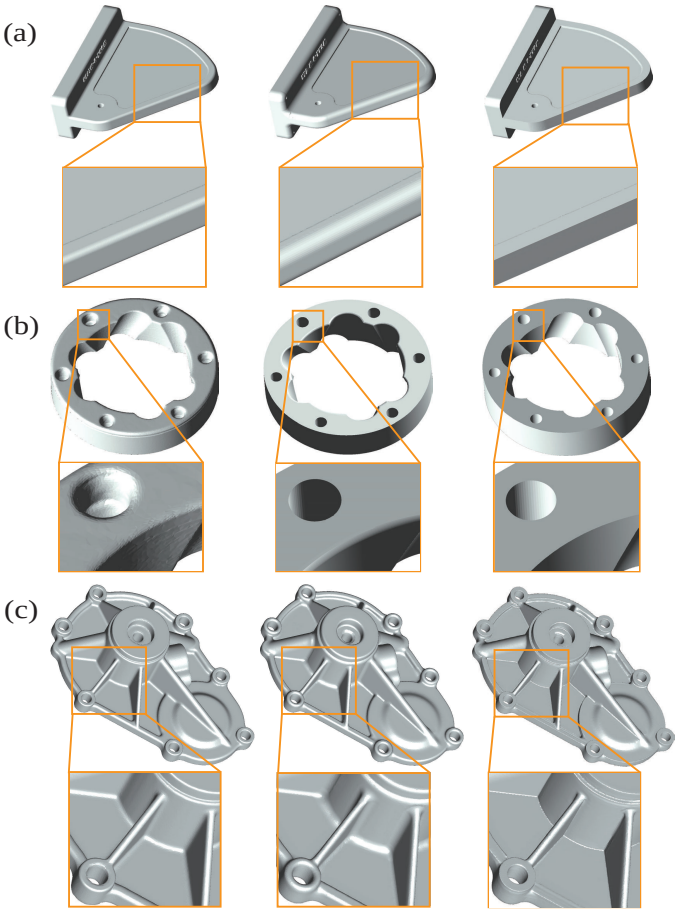

Fig. 18. Mesh blending application. From left to right: original meshes, smoothing and sharpening results.

(a)) as an example, we first extract feature curves as shown in Fig. 19 (b) and use them as the initial trajectories for next optimization steps. We then construct a rolling-ball blending surface with constant radius ($r'_{u_i} = 0.04$) as shown in Fig. 19 (c) and (d). Finally, in Fig. 19 (e) and (f), we set varying radii for two trajectories while the other trajectories have constant radii ($r'_{u_i} = 0.04$). For the yellow trajectory, we set $r'_{u_i} = 0.008 + \frac{i*0.032}{m}$ in the first half of its loop and $r'_{u_i} = 0.008 + (0.064 - \frac{i*0.032}{m})$ in the second half of this loop. Similarly, we set $r'_{u_i} = 0.015 + \frac{i*0.025}{m}$ and $r'_{u_i} = 0.015 + (0.05 - \frac{i*0.025}{m})$, respectively, for nodes of the gray trajectory.

5.4 Limitations

We successfully applied our method for the mesh segmentation and structure recovery of various 3D models. However, considering that we set a constant down-sampling scale value when building mesh superfacets, the method is not self-adaptive to different detail richness. In the top row of Fig. 20, we show an example of unsatisfactory segmentation where a very dense superfacet is needed on the long and narrow feature regions. Besides, although our method works well for extracting and fitting the usual quadric surfaces, we cannot guarantee satisfactory segmentation results on organic models (such as human bodies or animals) because they are usually composed of freeform shapes, as shown in the bottom of Fig. 20.

In addition, because the time cost of building superfacets is sensitive to the number of faces of the input mesh, our framework cannot handle large-scale urban scenes that may contain millions of triangle faces. Moreover, our segmentation approach only depends on low-level geometric properties, hence, it may not produce satisfying results when some high-level or semantic structure information is needed.

6 Conclusion

We have presented a new framework to segment 3D models and reveal the surface structures of the underlying shapes,

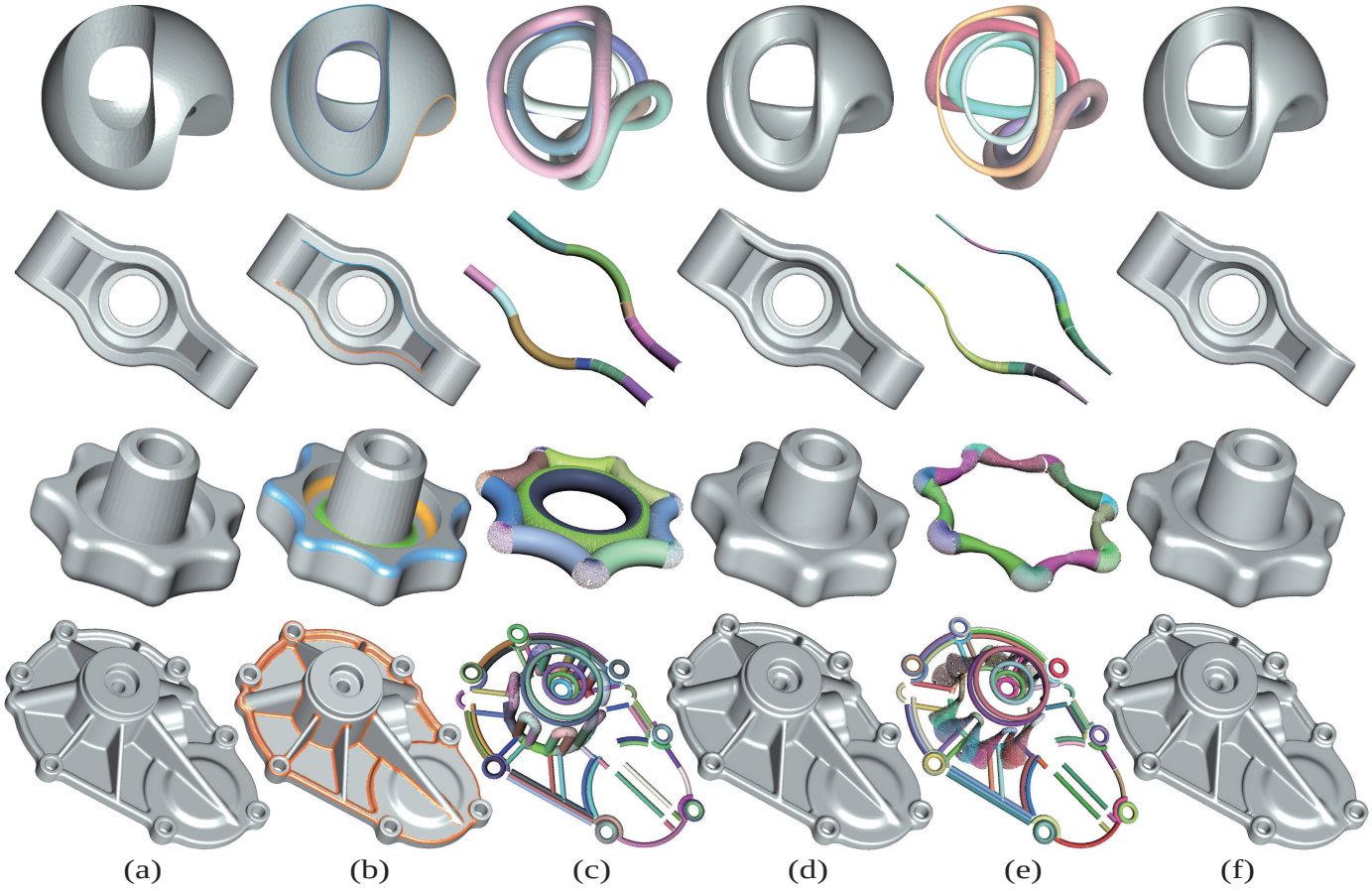
(a)
(b)
(c)
(d)
(e)
(f) especially for the rolling-ball blending surfaces. Robustness has been emphasized through the construction of mesh superfacets. Subsequently, an efficient MRF-based labeling strategy is applied to the superfacets. After segmentation, an iterative optimization algorithm is proposed to fit the rolling-ball blending patches by accurately recovering the blending parameters. We demonstrated the advantages of our method by conducting experiments on CAD mechanical objects, furniture and complex building models. We also propose an application to control the blending result by specifying different blending radii.

Fig. 19. In the first and second rows, we create rolling-ball blending surfaces on sharp models. In the bottom two rows, we change blending surfaces with varying radii. For each row, from left to right: (a) the input, (b) extracted feature curves or blending surfaces, (c) constructed constant-radius trajectories, (d) constant-radius blending surfaces, (e) constructed varying-radii trajectories, (f) varying-radii blending surfaces.

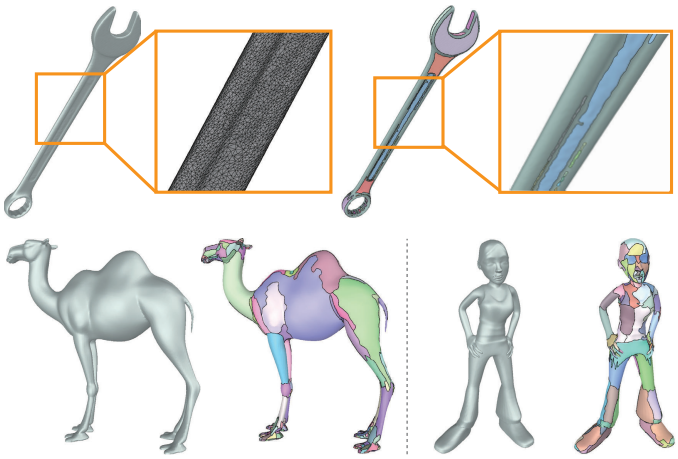

Fig. 20. We cannot produce satisfactory segmentation results on the long and narrow feature regions (top), as well as organic models containing irregular shapes (bottom).

In future work, we would first like to extend our algorithm to handle point cloud data. Additionally, although CNN-based deep learning approaches are currently being widely used to solve 3D mesh or point cloud segmentation tasks, we are not aware of any approach that could segment blending surfaces. The reason may be that these approaches mainly segment objects or scenes semantically, and the needed training dataset can not be collected easily. Therefore, we are interested in collecting or generating a large number of 3D models with ground-truth blending surface segmentation and exploring deep neural networks to identify other surface types of engineering objects.

ACKNOWLEDGMENTS

We thank anonymous reviewer for their valuable comments. This work is partially funded by the National Key R&D Program (2018YFB2100602), the National Natural Science Foundation of China (61802406, 61772523, U2003109), Beijing Natural Science Foundation (L182059), the Key Research Program of Frontier Sciences CAS (QYZDY-SSW-SYS004), the Strategic Priority Research Program of CAS

(XDA23090304), the Youth Innovation Promotion Association of CAS (Y201935), Open Project Program of State Key Laboratory of Virtual Reality Technology and Systems Beihang University (VRLAB2019B02), the Alibaba Group through Alibaba Innovative Research Program, and the Fundamental Research Funds for the Central Universities.

REFERENCES

[1] D. Cohen-Steiner, P. Alliez, and M. Desbrun, "Variational shape approximation," *ACM Trans. on Graphics*, vol. 23, no. 3, pp. 905–914, 2004.

[2] F. Lafarge, R. Keriven, and M. Bredif, "Insertion of 3-d-primitives in mesh-based representations: Towards compact models preserving the details," *IEEE Trans. on Image Processing*, vol. 19, no. 7, pp. 1683–1694, 2010.

[3] J. Wang, D. Gu, Z. Yu, C. Tan, and L. Zhou, "A framework for 3d model reconstruction in reverse engineering," *Computers & Industrial Engineering*, vol. 63, no. 4, pp. 1189–1200, 2012.

[4] T. Du, J. P. Inala, Y. Pu, A. Spielberg, A. Schulz, D. Rus, A. Solar-Lezama, and W. Matusik, "Inversecsg: Automatic conversion of 3d models to csg trees," *ACM Trans. on Graphics (Proc. SIGGRAPH Asia)*, p. 213, 2018.

[5] B. Li, R. Schnabel, S. Jin, and R. Klein, "Variational surface approximation and model selection," *Comp. Graph. Forum*, vol. 28, no. 7, pp. 1985–1994, 2009.

[6] H. Fang, F. Lafarge, and M. Desbrun, "Planar shape detection at structural scales," in *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2965–2973.

[7] Y. Verdie, F. Lafarge, and P. Alliez, "LOD Generation for Urban Scenes," *ACM Trans. on Graphics*, vol. 34, no. 3, 2015.

[8] G. Papaioannou, E. A. Karabassi, and T. Theoharis, "Segmentation and surface characterization of arbitrary 3d meshes for object reconstruction and recognition," in *International Conference on Pattern Recognition*, vol. 1, 2000, pp. 734–737.

[9] G. Lavoué, F. Dupont, and A. Baskurt, "A new {CAD} mesh segmentation method, based on curvature tensor analysis," *Computer-Aided Design*, vol. 37, no. 10, pp. 975 – 987, 2005.

[10] J. Wu and L. Kobbelt, "Structure recovery via hybrid variational surface approximation," *Comp. Graph. Forum (Proc. EUROGRAPHICS)*, vol. 24, no. 3, pp. 277–284, 2005.

[11] F. Lafarge, R. Keriven, and M. Brédif, "Combining meshes and geometric primitives for accurate and semantic modeling," in *Proc. BMVC*, 2009, pp. 38.1–38.11.

[12] J. Liu, J. Wang, T. Fang, C.-L. Tai, and L. Quan, "Higher-order crf structural segmentation of 3d reconstructed surfaces," in *IEEE International Conference on Computer Vision*, 2015, pp. 2093–2101.

[13] H. Fang, F. Lafarge, and M. Desbrun, "Planar Shape Detection at Structural Scales," in *IEEE Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, US, 2018.

[14] J. Rossignac and A. Requicha, "Constant-radius blending in solid modelling," in *Proc. Computers in Mechanical Engineering*, 1984, pp. 65–73.

[15] T. Várady, J. Vida, and R. R. Martin, "Parametric blending in a boundary representation solid modeller," in *Proc. IMA Conference on the Mathematics of Surfaces*. Clarendon Press, 1988, pp. 171–197.

[16] J. Vida, R. R. Martin, and T. Varady, "A survey of blending methods that use parametric surfaces," *Computer-Aided Design*, vol. 26, no. 5, pp. 341–365, 1994.

[17] B. K. Choi and S. Ju, "Constant-radius blending in surface modelling," *Computer-Aided Design*, vol. 21, no. 4, pp. 213–220, 1989.

[18] H. Ling, H. Songbo, Z. Xindong, and L. Yi, "Construction of blending surfaces," *COMPUTER AIDED DRAFTING, DESIGN AND MANUFCTURING*, no. 1, p. 3, 2000.

[19] Y.-S. Liu, H. Zhang, J.-H. Yong, P.-Q. Yu, and J.-G. Sun, "Mesh blending," *The Visual Computer*, vol. 21, no. 11, pp. 915–927, 2005.

[20] G. Kós, R. R. Martin, and T. Várady, "Methods to recover constant radius rolling ball blends in reverse engineering," *Comp. Aided Geom. Design*, vol. 17, no. 2, pp. 127–160, 2000.

[21] A. Shamir, "A survey on mesh segmentation techniques," *Comp. Graph. Forum*, vol. 27, no. 6, pp. 1539–1556, 2008.

[22] X. Chen, A. Golovinskiy, and T. Funkhouser, "A benchmark for 3d mesh segmentation," *ACM Trans. on Graphics*, vol. 28, no. 3, pp. 73:1–73:12, 2009.

[23] M. Attene, S. Katz, M. Mortara, G. Patane, M. Spagnuolo, and A. Tal, "Mesh segmentation - A comparative study," in *Shape Modeling International – SMI*, 2006, pp. 14–25.

[24] P. Theologou, I. Pratikakis, and T. Theoharis, "A comprehensive overview of methodologies and performance evaluation frameworks in 3d mesh segmentation," *Computer Vision and Image Understanding*, vol. 135, pp. 49–82, 2015.

[25] A. Jagannathan and E. L. Miller, "Three-dimensional surface mesh segmentation using curvedness-based region growing approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 12, pp. 2195–2204, 2007.

[26] D. Hwan Kim, I. Dong Yun, and S. Uk Lee, "Boundary-trimmed 3d triangular mesh segmentation based on iterative merging strategy," *Pattern Recognition*, vol. 39, no. 5, pp. 827–838, 2006.

[27] M. Attene, B. Falcidieno, and M. Spagnuolo, "Hierarchical mesh segmentation based on fitting primitives," *The Visual Computer*, vol. 22, no. 3, pp. 181–193, 2006.

[28] H. Zhang, C. Li, L. Gao, S. Li, and G. Wang, "Shape segmentation by hierarchical splat clustering," *Computers & Graphics*, vol. 51, pp. 136–145, 2015.

[29] X. Yang and X. Jia, "Simple primitive recognition via hierarchical face clustering," *Computational Visual Media*, pp. 1–13, 2020.

[30] A. P. Mangan and R. T. Whitaker, "Partitioning 3d surface meshes using watershed segmentation," *IEEE Trans. on Vis. and Comp. Graphics*, vol. 5, no. 4, pp. 308–321, 1999.

[31] S. Katz and A. Tal, "Hierarchical mesh decomposition using fuzzy clustering and cuts," *ACM Trans. on Graphics*, vol. 22, no. 3, pp. 954–961, 2003.

[32] Y.-K. Lai, Q.-Y. Zhou, S.-M. Hu, and R. R. Martin, "Feature sensitive mesh segmentation," in *Proc. Symposium on Solid and Physical Modeling*, 2006, pp. 17–25.

[33] S. Lloyd, "Least squares quantization in pcm," *IEEE Trans. on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[34] D.-M. Yan, W. Wang, Y. Liu, and Z. Yang, "Variational mesh segmentation via quadric surface fitting," *Computer-Aided Design*, vol. 44, no. 11, pp. 1072–1082, 2012.

[35] P. V. Sander, Z. J. Wood, S. Gortler, J. Snyder, and H. Hoppe, "Multi-chart geometry images," in *Proc. of Symp. of Geometry Processing*, 2003, p. 146–155.

[36] P. D. Simari and K. Singh, "Extraction and remeshing of ellipsoidal representations from mesh data," in *Proceedings of Graphics Interface*, 2005, pp. 161–168.

[37] D. Julius, V. Kraevoy, and A. Sheffer, "D-Charts: Quasi-Developable Mesh Segmentation," *Comp. Graph. Forum*, vol. 24, no. 3, pp. 981–90, 2005.

[38] T. Le and Y. Duan, "A primitive-based 3d segmentation algorithm for mechanical cad models," *Comp. Aided Geom. Design*, vol. 52, pp. 231–246, 2017.

[39] Q. Du, V. Faber, and M. Gunzburger, "Centroidal voronoi tessellations: Applications and algorithms," *SIAM review*, vol. 41, no. 4, pp. 637–676, 1999.

[40] Q. Du, M. Gunzburger, and L. Ju, "Advances in studies and applications of centroidal voronoi tessellations," *Numerical Mathematics: Theory, Methods and Applications*, vol. 3, no. 2, pp. 119–142, 2010.

[41] Y. Liu, W. Wang, B. Lévy, F. Sun, D.-M. Yan, L. Lu, and C. Yang, "On centroidal voronoi tessellation—energy smoothness and fast computation," *ACM Trans. on Graphics*, vol. 28, no. 4, pp. 1–17, 2009.

[42] J. Edwards, W. Wang, and C. Bajaj, "Surface segmentation for improved remeshing," in *Proceedings of International Meshing Roundtable*. Springer, 2013, pp. 403–418.

[43] G. Rong, Y. Liu, W. Wang, X. Yin, D. Gu, and X. Guo, "Gpu-assisted computation of centroidal voronoi tessellation," *IEEE Trans. on Vis. and Comp. Graphics*, vol. 17, no. 3, pp. 345–356, 2010.

[44] Y. Fei, G. Rong, B. Wang, and W. Wang, "Parallel l-bfgs-b algorithm on gpu," *Computers & Graphics*, vol. 40, pp. 1–9, 2014.

[45] E. Kalogerakis, A. Hertzmann, and K. Singh, "Learning 3d mesh segmentation and labeling," *ACM Trans. on Graphics (Proc. SIGGRAPH)*, pp. 1–12, 2010.

[46] K. Xu, V. G. Kim, Q. Huang, N. Mitra, and E. Kalogerakis, "Data-driven shape analysis and processing," in *SIGGRAPH ASIA 2016 Courses*, 2016, pp. 1–38.

[47] R. S. Rodrigues, J. F. Morgado, and A. J. Gomes, "Part-based mesh segmentation: a survey," *Comp. Graph. Forum*, vol. 37, no. 6, pp. 235–274, 2018.

[48] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 652–660.
[49] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Proc. International Conference on Neural Information Processing Systems (NIPS)*, 2017, p. 5105–5114.
[50] F. Yu, K. Liu, Y. Zhang, C. Zhu, and K. Xu, "Partnet: A recursive part decomposition network for fine-grained and hierarchical shape segmentation," in *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 9491–9500.
[51] S. Hu, J. Cai, and Y. Lai, "Semantic labeling and instance segmentation of 3d point clouds using patch context analysis and multiscale processing," *IEEE Trans. on Vis. and Comp. Graphics*, vol. 26, no. 7, pp. 2485–2498, 2020.
[52] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM Trans. on Graphics*, vol. 38, no. 5, pp. 1–12, 2019.
[53] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or, "Meshcnn: a network with an edge," *ACM Trans. on Graphics (Proc. SIGGRAPH)*, vol. 38, no. 4, pp. 1–12, 2019.
[54] L. Li, M. Sung, A. Dubrovina, L. Yi, and L. J. Guibas, "Supervised fitting of geometric primitives to 3d point clouds," in *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2652–2660.
[55] G. Sharma, D. Liu, E. Kalogerakis, S. Maji, S. Chaudhuri, and R. Měch, "Parsenet: A parametric surface fitting network for 3d point clouds," in *European Conference on Computer Vision (ECCV)*, 2020, pp. 261–276.
[56] R. Schnabel, R. Wahl, and R. Klein, "Efficient ransac for point-cloud shape detection." *Comp. Graph. Forum*, vol. 26, no. 2, pp. 214–226, 2007.
[57] M. Attene and G. Patanè, "Hierarchical structure recovery of point-sampled surfaces," *Comp. Graph. Forum*, vol. 29, no. 6, pp. 1905–1920, 2010.
[58] Y. Li, X. Wu, Y. Chrysathou, A. Sharf, D. Cohen-Or, and N. J. Mitra, "Globfit: Consistently fitting primitives by discovering global relations," *ACM Trans. on Graphics*, vol. 30, no. 4, pp. 52:1–52:12, 2011.
[59] T.-T. Tran, V.-T. Cao, and D. Laurendeau, "Extraction of cylinders and estimation of their parameters from point clouds," *Computers & Graphics*, vol. 46, pp. 345–357, 2015.
[60] S. Oesau, F. Lafarge, and P. Alliez, "Planar Shape Detection and Regularization in Tandem," *Comp. Graph. Forum*, vol. 35, no. 1, 2016.
[61] L. Li, M. Sung, A. Dubrovina, L. Yi, and L. Guibas, "Supervised fitting of geometric primitives to 3d point clouds," *arXiv preprint arXiv:1811.08988*, 2018.
[62] S. Petitjean, "A survey of methods for recovering quadrics in triangle meshes," *ACM Computing Surveys (CSUR)*, vol. 34, no. 2, pp. 211–262, 2002.
[63] Y. Chen and C. Liu, "Quadric surface extraction using genetic algorithms," *Computer-Aided Design*, vol. 31, no. 2, pp. 101–110, 1999.
[64] S. J. Ahn, W. Rauh, H. S. Cho, and H.-J. Warnecke, "Orthogonal distance fitting of implicit curves and surfaces," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 5, pp. 620–638, 2002.
[65] T. Kanai, Y. Ohtake, and K. Kase, "Hierarchical error-driven approximation of implicit surfaces from polygonal meshes," in *Proc. of Symp. of Geometry Processing*, vol. 256, 2006, pp. 21–30.
[66] O. K.-C. Au, C.-L. Tai, H.-K. Chu, D. Cohen-Or, and T.-Y. Lee, "Skeleton extraction by mesh contraction," *ACM Trans. on Graphics*, vol. 27, no. 3, p. 44, 2008.
[67] P. Simari, G. Picciau, and L. De Floriani, "Fast and scalable mesh superfacets," *Comp. Graph. Forum*, vol. 33, no. 7, pp. 181–190, 2014.
[68] X. Pan, Y. Zhou, F. Li, and C. Zhang, "Superpixels of rgb-d images for indoor scenes based on weighted geodesic driven metric," *IEEE Trans. on Vis. and Comp. Graphics*, vol. 23, no. 10, pp. 2342–2356, 2016.
[69] F. Cazals and M. Pouget, "Estimating differential quantities using polynomial fitting of osculating jets," *Comp. Aided Geom. Design*, vol. 22, no. 2, pp. 121–146, 2005.
[70] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Y. Zeevi, "The farthest point strategy for progressive image sampling," *IEEE Trans. on Image Processing*, vol. 6, no. 9, pp. 1305–1315, 1997.
[71] Y. Zhuang, H. Dou, N. Carr, and T. Ju, "Feature-aligned segmentation using correlation clustering," *Computational Visual Media*, vol. 3, no. 2, pp. 147–160, 2017.
[72] Q. Zhou and A. Jacobson, "Thingi10k: A dataset of 10,000 3d-printing models," *arXiv preprint arXiv:1605.04797*, 2016.
[73] The CGAL Project, *CGAL User and Reference Manual*, 4.14.1 ed., 2019.
[74] S. Koch, A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin, and D. Panozzo, "Abc: A big cad model dataset for geometric deep learning," in *IEEE Computer Vision and Pattern Recognition (CVPR)*, June 2019.
[75] M. Attene, B. Falcidieno, J. Rossignac, and M. Spagnuolo, "Sharpen&bend: Recovering curved sharp edges in triangle meshes produced by feature-insensitive sampling," *IEEE Trans. on Vis. and Comp. Graphics*, vol. 11, no. 2, pp. 181–192, 2005.
[76] C. Rocchini, P. Cignoni, F. Ganovelli, C. Montani, P. Pingi, and R. Scopigno, "Marching intersections: an efficient resampling algorithm for surface management," in *Proc. International Conference on Shape Modeling and Applications*, 2001, pp. 296–305.

Long Zhang is working toward the Ph.D. degree in the School of Artificial Intelligence at University of Chinese Academy of Sciences, Beijing. He obtained his bachelor degree from Southwest University in 2014. His research interests include computer graphics, geometry processing and 3D reconstruction.

Jianwei Guo is an associate professor in National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences(CASIA). He received his Ph.D. degree in computer science from CASIA in 2016, and bachelor degree from Shandong University in 2011. His research interests include computer graphics, geometry processing, and 3D shape analysis.

Jun Xiao is a professor in University of Chinese Academy of Sciences, Beijing. He obtained his Ph.D. degree in communication and information system from the Graduate University of Chinese Academy of Sciences in 2008. His research interests include computer graphics, computer vision, image processing and 3D reconstruction.

Xiaopeng Zhang is a professor in National Laboratory of Pattern Recognition at Institute of Automation, Chinese Academic of Sciences (CAS). He received his Ph.D. degree in Computer Science from Institute of Software, CAS in 1999. He received the National Scientific and Technological Progress Prize (second class) in 2004. His main research interests include computer graphics and image processing.

Dong-Ming Yan is a professor in National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences(CAS). He received his Ph.D. degree in computer science from Hong Kong University in 2010, and his master and bachelor degrees in computer science and technology from Tsinghua University in 2005 and 2002, respectively. His research interests include computer graphics, geometric processing, and visualization.