Surface Mesh Smoothing, Regularization and Feature Detection

H. Huang * U. Ascher †

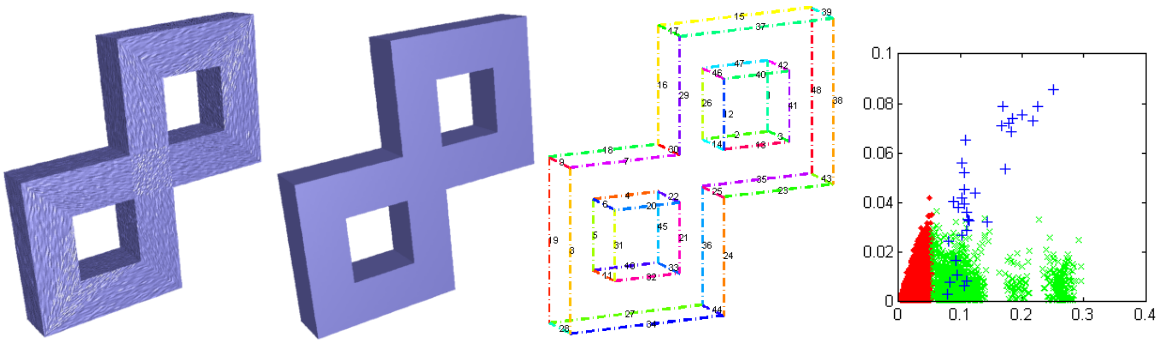
February 21, 2008

Abstract

We describe a hybrid algorithm that is designed to reconstruct a piecewise smooth surface mesh from noisy input. While denoising, our method simultaneously regularizes triangle meshes on flat regions for further mesh processing and preserves crease sharpness for faithful reconstruction. A clustering technique, which combines K-means and geometric a priori information, is first developed and refined. It is then used to implement vertex classification so that we can not only apply different smoothing operators on different vertex groups for different purposes, but also succeed in crease detection, where the tangent plane of the surface is discontinuous, without any significant cost increase. Consequently we are capable of efficiently obtaining different mesh segmentations, depending on user input and thus suitable for various applications.

Key words: Multiscale Anisotropic Laplacian, Umbrella Operator, Mesh Regularization, Crease Detection, Mesh Segmentation, K-means Clustering.

1 Introduction



(a) (b) (c) (d)

Figure 1: Mesh smoothing: (a) noisy Double-Torus model; (b) reconstructed model; (c) crease detection; (d) data classification.

*Institute of Applied Mathematics, University of British Columbia, Vancouver, BC, V6T 1Z2, Canada (hhzhiyan@math.ubc.ca). Supported in part under NSERC Research Grant 84306.

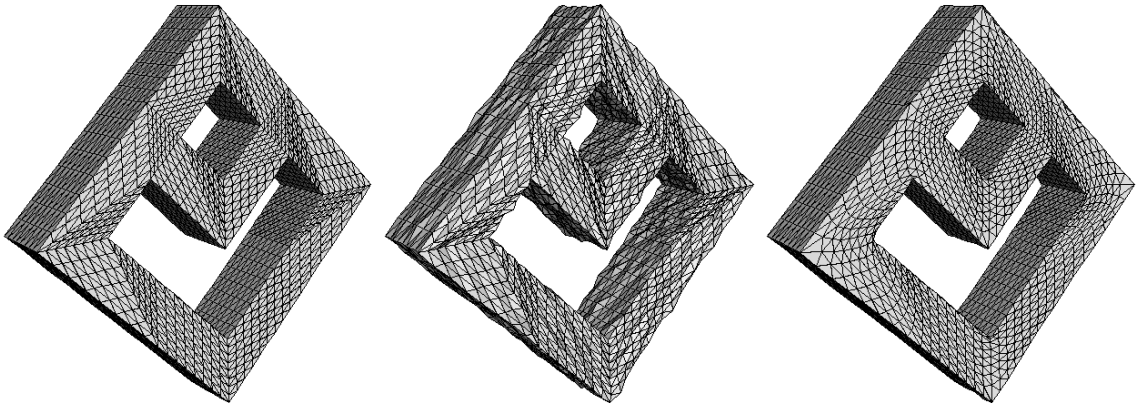
†Department of Computer Science, University of British Columbia, Vancouver, BC, V6T 1Z4, Canada. (ascher@cs.ubc.ca). Supported in part under NSERC Research Grant 84306.

The rapid development of 3D scanning and acquisition technology has necessitated efficient denoising algorithms for triangular surface meshes. Furthermore, simulations in time involving such surfaces often require subsequent smoothing and regularization. Fast and simple 3D mesh smoothing operators based upon geometric isotropic diffusion were proposed in the 1990s [38, 16]. Later, algorithms based on anisotropic diffusion were introduced to avoid smearing out of real important features [11, 8, 3, 37, 4, 17, 9]. These methods are typically expensive, both in terms of cost per iteration and in the number of iterations required to achieve satisfactory results. Moreover, they often require the user to provide unintuitive parameter values, including a threshold value and a time step for the anisotropic diffusion process. The approach of bilateral filtering has given rise to methods that more rapidly yield results of a quality similar to anisotropic diffusion, albeit with less theoretical justification [13, 21, 44]. The latter methods usually require very few, cheap iterations, and cost but a tiny fraction of the computational effort required to carry out an elaborate anisotropic diffusion process, especially on large meshes. However, the results generated by these bilateral filtering variants may strongly depend on vertex neighborhood choices and the manner in which tangent planes are approximated.

Further, sampling irregularities in the given mesh occasionally distort results and significantly slow algorithms down [14, 25, 10, 18]. Several discrete operators were designed to overcome this numerical difficulty and maintain sampling irregularity in the smoothed meshes. To satisfy different application requirements, a smoothing method with simultaneous mesh regularization is proposed in [31].

The methods mentioned above can be thought of as having just one stage or one pass. Another approach for adaptive mesh smoothing is to recognize mesh features first. In [34], a method is proposed that consists of three stages: feature-preserving pre-smoothing, feature and non-feature region partitioning, and feature and non-feature region smoothing using two separate approaches. In [36], the authors first use the eigen analysis of a normal voting tensor to extract sharp edges, then apply bilateral filtering to smooth along the direction of the sharp edge, and finally, employ modified bilateral filtering to the overall mesh to obtain a smoothed mesh model with sharp features. The algorithm proposed here has a similar structure in that edges and corners are recognized at an earlier stage. However, our methods for detecting features and for denoising meshes are totally different. Our methods can be simply implemented, are rather efficient computationally, and can be easily extended to handle a wide variety of applications as described below.

Polygonal surface models with features that are challenging to preserve can be roughly divided into two classes. In the first class objects usually contain many visually meaningful fine scale components or details, and typically require very fine meshes to represent them well. Examples are the Dragon and Igea models used in [18]; see also Figs. 13 and 12. The second class consists of piecewise smooth CAD-like models, which usually have large flat regions, long sharp edges and distinct corners. The required mesh for representation can often be much coarser. Examples are the Star and Fandisk models, here depicted in Figs. 4 and 6. All of these models may be sampled very irregularly, as in Fig. 11. In [18], we have methodically developed a fast adaptive multiscale mesh denoising algorithm that is capable of retaining fine scale texture as well as mesh irregularities. This algorithm performs particularly well for models of the first class. In the present article we focus on denoising, as in Fig. 1, and on mesh regularizing, as in Fig. 2, for the second class of models. Thus, the goal becomes to generate enough smoothing on the flat regions while preserving edge sharpness and corner distinction. In addition, we extend our algorithm to efficiently detect all sharp creases and consequently complete various mesh segmentation options based on user desire, see Figs. 7 and 8.



(a) (b) (c)

Figure 2: Mesh regularization: (a) original clean but nonuniform Torus mesh; (b) corrupted by heavy noise, this is our input data; (c) smoothed and regularized by our hybrid algorithm.

Let us refer to vertices in flat regions as non-feature vertices. Both edge and corner vertices are called feature vertices. Since different vertex groups should be dealt with in different ways, it is advantageous to cluster vertices accurately and then choose the most suitable discrete smoothing operator on each vertex cluster. Data classification on polygonal meshes is not new as such. In [26], Lavou et al designed a constant curvature region decomposition of 3D meshes following vertex classification. Fuzzy clustering was applied on triangular mesh faces for hierarchical mesh decomposition in [24]. Liu and Zhang [27] proposed a 3D mesh segmentation algorithm through spectral clustering of mesh faces. In [7], Chen et al used Bayesian discriminant analysis to determine the decision boundary for separating potential feature and non-feature vertices in curvature space. However, we believe that the present paper is the first to develop a specific clustering technique using the first order height intensity and properly scaled Gaussian curvature, which is good enough to be applied directly on noisy meshes. All related parameters are highly intuitive in our geometric context.

In the following Section 2 we first recall several mesh denoising algorithms. After analyzing the strengths and weaknesses of the corresponding discrete operators we develop our clustering technique in Section 3, and subsequently choose the most suitable operator for smoothing on each vertex cluster with a specific feature. Further, based on the constructed vertex classification we extend our algorithm in Section 4 to detect all sharp creases, and segment meshes with respect to mesh sharp features and user requirements. Implementation details, results and discussion are then presented in Section 5. Summary and conclusions follow.

2 Discrete Laplacian Operators

Before describing different discrete Laplacian operators, we introduce notation. A manifold $\mathcal{M}$ is discretized by a triangular surface mesh S with its sets of vertices $V(S) = \mathbf{x} = \{\mathbf{x}_i;\ i = 1, \ldots, N\}$ and directed edges $E(S)$. If two distinct vertices $\mathbf{x}_i$ and $\mathbf{x}_k$ are linked by an edge $\mathbf{e}_{i,k} = \mathbf{x}_k - \mathbf{x}_i$ then we denote $k \in \mathcal{N}(i)$ and the edge length $l_{i,k} = |\mathbf{e}_{i,k}|$. $\mathcal{F}(i)$ represents the one-ring-face set that is adjacent to the vertex $\mathbf{x}_i$. The given data is a noisy mesh of this sort,

and we denote its vertices $\mathbf{x}(0) = \mathbf{v} = \{\mathbf{v}_i;\ i = 1, \ldots, N\}$.

2.1 Discrete isotropic Laplacian

The simplest discrete isotropic Laplacian operator is the Umbrella operator [38]. It averages the neighboring edges

$$\Delta \mathbf{x}_i = \frac{1}{m_i} \sum_{k \in \mathcal{N}(i)} \mathbf{e}_{i,k}, \tag{1}$$

where $m_i = |\mathcal{N}(i)|$, the number of neighbors of vertex $\mathbf{x}_i$. This is a linear form implying the assumption that all neighboring edge lengths are roughly equal to one. Hence it can serve as an effective smoother if the targeted mesh is close enough to being regular. On the other hand, when the model has different discretization rates significant local mesh regularization, which may or may not be desirable, is introduced by this Umbrella operator (see [31]). To retain mesh sampling rates, a better choice is the scale-dependent version [14]. Further, to solve problems arising from unequal face angles, a better approximation to the mean curvature normal was proposed in [10, 28] which doesn't produce vertex tangential drifting when surfaces are relatively flat and compensates both for unequal edge lengths and for unequal face angles. This approach does not improve the mesh irregularity sampling rate [31]. All these schemes are based on isotropic diffusion, though, which implies that they all easily smear sharp features during the smoothing process.

2.2 Discrete anisotropic Laplacian

To better reconstruct sharp features, consider next designing an anisotropic Laplacian operator [18]. For each vertex $\mathbf{x}_i$ of the given data mesh we estimate the corresponding normal $\mathbf{n}_i$ as the mean of the adjacent face normals and define the local height intensity $h_{i,k} = \mathbf{e}_{i,k}^T \mathbf{n}_i$, the projection of $\mathbf{e}_{i,k}$ along the normal $\mathbf{n}_i$. A corresponding discrete anisotropic Laplacian (AL) operator is then given by

$$\Delta \mathbf{x}_i = \frac{1}{C_i} \left(\sum_{k \in \mathcal{N}(i)} g(h_{i,k}) h_{i,k} \right) \mathbf{n}_i, \tag{2}$$

where $C_i = \sum_{k \in \mathcal{N}(i)} g(h_{i,k})$.

Further, we employ a Gaussian filter in this context, which is simple, robust and clearly reduces the influence of neighbors that contain large discontinuities in the normal space

$$g(h_{i,k}) \quad = \quad exp(-\frac{h_{i,k}^2}{2\sigma_i^2}), \tag{3}$$

where $\mathbf{h}_i = \{h_{i,k}\}_{k \in \mathcal{N}(i)}$. The *robust scale* σ_i may be automatically estimated by either the *mean* absolute deviation scaling

$$\sigma_i \quad = \quad 2 * \mathtt{mean}(|\mathbf{h}_i - \mathtt{mean}(\mathbf{h}_i)|), \tag{4a}$$

or the *median* absolute deviation scaling

$$\sigma_i \quad = \quad \mathtt{median}(|\mathbf{h}_i - \mathtt{median}(\mathbf{h}_i)|). \tag{4b}$$

2.3 Multiscale evolution

The normalization in (2) scales the operator such that, from the point of view of geometric diffusion, a step size $\tau = 1$ can be stably used in an explicit integration method such as forward Euler

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \Delta\mathbf{x}_i, \quad i = 1, \ldots, N. \tag{5}$$

Though AL is very efficient for smoothing some models, it cannot avoid over-smoothing in the presence of significant intrinsic texture, see [18]. A simple first step toward a better solution is then to add a data fidelity term

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \tau\Delta\mathbf{x}_i + \lambda_i(\mathbf{v}_i - \mathbf{x}_i), \quad i = 1, \ldots, N, \quad \lambda > 0, \tag{6}$$

where $\{\mathbf{v}_i;\ i = 1, \ldots, N\}$ is the given vertex set, thus increasing the influence of the given data during the denoising process at all iterates [39]. This also helps to reduce the effect of volume shrinkage over several iterations [38, 10, 18].

Since the surface scale is local and the mesh is generally nonuniform, we choose λ_i at each such iteration depending on the spatial location i. Also, since noise is carried in the position of vertices and changes all the neighboring height intensities $h_{i,k}$, we expect that λ_i should depend on this noise effect. Recall that the Gaussian parameter σ_i, adaptively determined by (4), is a robust estimator on local height intensity, larger over feature regions and smaller over flat regions. This naturally gives rise to the choice

$$\lambda_i = \sigma_i/\bar{\sigma}, \quad \text{where } \bar{\sigma} = \max\{\sigma_i;\ i = 1, \ldots, N\}. \tag{7}$$

For details, see [18]. The importance of the entire function $\lambda(\mathbf{x})$ obtained this way is magnified in the next iteration step by damping out the AL operator (2), to recapture a higher frequency band. Thus, in the jth iteration we calculate λ by (7) and set

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + K^j\Delta\mathbf{x}_i + \lambda_i(\mathbf{v}_i - \mathbf{x}_i), \quad i = 1, \ldots, N, \tag{8}$$

where K is an input parameter, $0 < K < 1$, with the default setting $K = 0.5$. We refer to this denoising scheme as the multiscale anisotropic Laplacian (MSAL) method.

In [18] we have shown that MSAL with the mean absolute deviation scaling (4a) performs rather well for recapturing fine scale texture. However, MSAL with (4a) sometime tends to over-smooth long sharp creases, for instance in the Fandisk model of Fig. 6. We must use another *robust scale* σ in the Gaussian filter, suitable for the current purpose. The image processing literature uses tools from robust statistics to sharpen image edges and corners by automatically estimating σ as the median absolute deviation of the given image intensity gradient, see [5, 33]. Extending this selection to polygonal meshes, our numerical experiments show that for preserving edge sharpness (4b) works much better than (4a). Similar conclusions are reached in [42], where mesh smoothing algorithms via mean and median filters applied to face normals are compared. Unfortunately, median absolute deviation scaling cannot generate enough smoothing on large flat regions and occasionally destroys corners due to its inherent limitations [15, 6]. This suggests that different methods may best be applied for different purposes. Vertex classification is thus a good way to go further. For example, if we classify vertices on a CAD-like model into three non-feature, edge and corner groups, we may apply different smoothing operators with different parameter selections to pursue a better reconstruction.

3 Vertex Classification

3.1 Data set computation for clustering

One way to classify mesh vertices is according to their principal curvatures. The magnitude of both principal curvatures is small for non-feature vertices and large for corner vertices. For an edge vertex, the magnitude of one of its principal curvatures is small and the other is quite large. Thus, if we have curvature information of the mesh, we can partition mesh vertices into three clusters denoted corner, edge and non-feature.

In [28] the discrete mean curvature normal at vertex $\mathbf{x}_i$ is defined by

$$\mathbf{K}(\mathbf{x}_i) = \frac{3}{2A_i} \sum_{k \in \mathcal{N}(i)} (\cot \alpha_{i,k} + \cot \beta_{i,k})(\mathbf{x}_i - \mathbf{x}_k)/2, \tag{9}$$

where $\alpha_{i,k}$ and $\beta_{i,k}$ are the two angles opposite to the edge $\mathbf{e}_{i,k}$ in the two triangles sharing it. Here we use one third of the simple one ring face area A_i at the vertex $\mathbf{x}_i$ to approximate the complicated surface area A_{mixed} defined in [28]. See [41, 29] for approximation convergence estimates. This yields the curvature expression $\kappa_M(\mathbf{x}_i) = \frac{1}{2}||\mathbf{K}(\mathbf{x}_i)||$. The discrete Gauss curvature κ_G at vertex $\mathbf{x}_i$ is approximated as

$$\kappa_G(\mathbf{x}_i) = 3(2\pi - \sum_{k \in \mathcal{F}(i)} \theta_k)/A_i, \tag{10}$$

where the sum is over the faces in the set $\mathcal{F}(i)$ and θ_k is the angle of the k-th face at the vertex $\mathbf{x}_i$. Since $\kappa_M = (\kappa_1 + \kappa_2)/2$ and $\kappa_G = \kappa_1\kappa_2$, discrete principal curvatures at the vertex $\mathbf{x}_i$ can be consequently computed by the quadratic formula

$$\kappa_1(\mathbf{x}_i) = \kappa_M(\mathbf{x}_i) + \sqrt{\Delta(\mathbf{x}_i)}, \qquad \kappa_2(\mathbf{x}_i) = \kappa_M(\mathbf{x}_i) - \sqrt{\Delta(\mathbf{x}_i)}, \tag{11}$$

with $\Delta(\mathbf{x}_i) = \max(\kappa_M^2(\mathbf{x}_i) - \kappa_G(\mathbf{x}_i),\ 0)$. Note that in the continuous case and almost always in the discrete case, $\kappa_M^2(\mathbf{x}_i) > \kappa_G(\mathbf{x}_i)$. The maximum principal curvature κ_1 is alway positive whereas the minimum principal κ_2 follows the sign of the Gaussian curvature κ_G which is negative at hyperbolic vertices. Since it is not necessary to differentiate ellipticity and hyperbolicity in our classification, we just consider the absolute value of both Gaussian curvature κ_G and minimum principal curvature κ_2. The quantitative 2D data matrix $C \in \mathbb{R}^{N\times 2}$ for classification is then generated as

$$C = [\kappa_1(\mathbf{x}_1), \kappa_1(\mathbf{x}_2), \ldots, \kappa_1(\mathbf{x}_N); |\kappa_2(\mathbf{x}_1)|, |\kappa_2(\mathbf{x}_2)|, \ldots, |\kappa_2(\mathbf{x}_N)|]^T. \tag{12}$$

However, since we are directly working on noisy vertices, the second-order curvature information is more sensitive and more easily influenced by noise effects, so that we often cannot obtain the desirable classification, see Figs. 4 and 9. Moreover, computation of principal curvatures of each vertex is obviously not cheap. Fortunately, we already have first-order information to roughly partition noisy meshes. Recall the local height intensity $h_{i,k} = \mathbf{e}_{i,k}^T \mathbf{n}_i$, which is the projection of $\mathbf{e}_{i,k}$ along the vertex normal $\mathbf{n}_i$. Now define

$$h_{max}(\mathbf{x}_i) = \max_{k \in \mathcal{N}(i)} |h_{i,k}|, \qquad h_{min}(\mathbf{x}_i) = \min_{k \in \mathcal{N}(i)} |h_{i,k}|. \tag{13}$$

Similarly to principal curvatures, for an edge vertex, h_{min} is small whereas h_{max} should be relatively quite larger. For a corner vertex, both h_{min} and h_{max} should be large, and for

a non-feature vertex they should both be relatively small. Thus, for a given vertex matrix $V = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^T, \mathbf{x}_i \in \mathbb{R}^{3\times 1}, i = 1, \ldots, N$, we can generate a corresponding quantitative data matrix $H \in \mathbb{R}^{N\times 2}$ with respect to height intensity

$$H = [h_{max}(\mathbf{x}_1), h_{max}(\mathbf{x}_2), \ldots, h_{max}(\mathbf{x}_N); h_{min}(\mathbf{x}_1), h_{min}(\mathbf{x}_2), \ldots, h_{min}(\mathbf{x}_N)]^T. \tag{14}$$

In pattern recognition terminology [12], the rows of C or H are called the patterns or objects and the columns are called the features or attributes. The objective of clustering is to partition the data set into several clusters. Generally, a cluster is a group of objects that are more similar to one another than to members of other clusters. The term *similarity* should be understood as mathematical similarity, measured in some well-defined sense. In our case, considering the first column of data matrices as X-coordinate and the second column as Y-coordinate, the similarity can be measured simply by the Euclidean distance based on the geometric information we already have. The whole data set sits in the first quadrant. Objects that correspond to non-feature vertices should be near the origin; objects that correspond to edge vertices should be close to the X-axis and relatively far from the Y-axis; and objects that correspond to corner vertices should be close to neither X-axis nor Y-axis.

However, a height intensity value depends on the approximation of the vertex normal. For some corner vertices, the value of their h_{min} could be very small where the average of the neighboring face normals might point along one edge such that they would be mistakenly grouped into the edge cluster, see Figs. 4, 5 and 9. To address this, we employ the Gaussian curvature κ_G. There are two reasons. The first is that $|\kappa_G|$ is larger at corner vertices and much smaller at both non-feature and edge vertices. The second reason is that computing κ_G by (10) is inexpensive. We can get θ and A very quickly by applying simple operations to the cross product of two edge vectors in each triangle. These cross products must be performed anyway when computing face normals. In order to generate the data matrix similar to C or H with respect to the Gaussian curvature, we set

$$\kappa_{G1}(\mathbf{x}_i) = \frac{6\pi}{A_i} + \sqrt{\Theta(\mathbf{x}_i)}, \qquad \kappa_{G2}(\mathbf{x}_i) = \frac{6\pi}{A_i} - \sqrt{\Theta(\mathbf{x}_i)}, \tag{15}$$

with $\Theta(\mathbf{x}_i) = (\frac{6\pi}{A_i})^2 - \kappa_G(\mathbf{x}_i)$. The corresponding quantitative data matrix $G \in \mathbb{R}^{N\times 2}$ for corner identification is then generated as

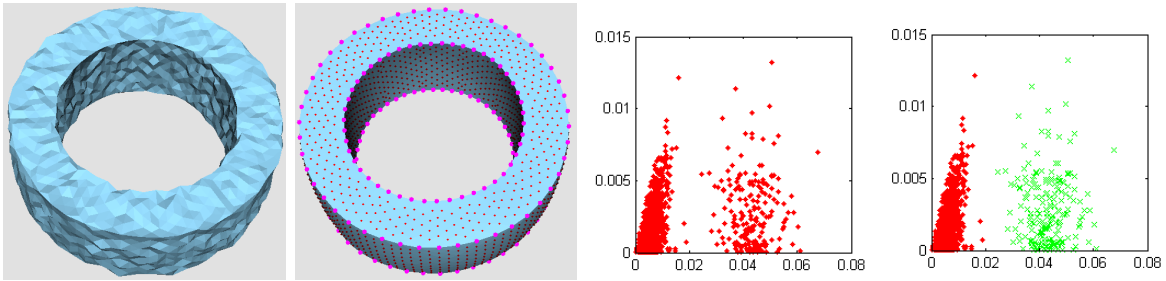
$$G = [\kappa_{G1}(\mathbf{x}_1), \kappa_{G1}(\mathbf{x}_2), \ldots, \kappa_{G1}(\mathbf{x}_N); |\kappa_{G2}(\mathbf{x}_1)|, |\kappa_{G2}(\mathbf{x}_2)|, \ldots, |\kappa_{G2}(\mathbf{x}_N)|]^T. \tag{16}$$

Clearly, $|\kappa_{G2}|$ is larger at the corner vertices than at the rest. Therefore, objects in G corresponding to corner vertices should be relatively far above the X-axis, whereas objects corresponding to non-feature and edge vertices should be close to the X-axis. For a better clustering we linearly rescale the first column of G (X-coordinate in the Euclidean axis) such that all objects are bounded by a square box.

3.2 K-means clustering

The classic unsupervised fast clustering algorithm is K-means [12], which allocates each data point to one of c clusters to minimize the within-cluster sum of squares. Taking the partition data matrix H as an example, we seek a partition $H_1, \ldots, H_c$ to minimize the objective function

$$\sum_{j=1}^{c} \sum_{\hat{h}_i \in H_j} ||\hat{h}_i - \mu_j||_2, \tag{17}$$



(a) (b) (c) (d)

Figure 3: (a) Corrupted Ring model; (b) smoothed model based on partition in (d): non-feature vertices are marked by smaller pink dots and feature (edge) vertices are marked by larger pink dots; (c) data set H computed for the noisy Ring model in (a); (d) two clusters by K-means: red $\rightarrow$ non-feature and green $\rightarrow$ feature.

where $\hat{h}_i = [h_{max}(\mathbf{x}_i), h_{min}(\mathbf{x}_i)]$, H_j is a set of objects (data points) in the j-th cluster and $\mu_j = \texttt{mean}(\sum_{\hat{h}_i \in H_j} \hat{h}_i)$ is the the center point over the j-th cluster. The number of clusters c is usually two in the current context. For example, corner and non-corner clusters are always expected in the data matrix G, see Figs. 4(i) and 5(c). After the corner cluster is identified and temporarily removed from H, nonempty edge and non-feature clusters are expected in the objects that remain, see Figs. 4(p) and 5(f). Or, as in the Ring model in Fig. 3, there is no corner vertex at all, so we only need to classify vertices into the two clusters of feature and non-feature.

One important issue here is the need to supply a good initial guess for the center points μ_j for starting the K-means clustering. According to our analysis of the distribution of data matrices G, H, or C above, we intuitively set

$$\begin{aligned} \mu_{corner} &= [\kappa_{G1}(\mathbf{x}_j),\ |\kappa_{G2}(\mathbf{x}_j)|], \quad j = \texttt{arg} \max_{1\le i\le N} |\kappa_{G2}(\mathbf{x}_i)|, \\ \mu_{noncorner} &= [\kappa_{G1}(\mathbf{x}_k),\ |\kappa_{G2}(\mathbf{x}_k)|], \quad k = \texttt{arg} \min_{1\le i\le N} |\kappa_{G2}(\mathbf{x}_i)|, \end{aligned}$$

for corner identification in the data matrix G, and then

$$\mu_{edge} = [\max_{1\le i\le N} h_{max}(\mathbf{x}_i),\ \min_{1\le i\le N} h_{min}(\mathbf{x}_i)], \quad \mu_{nonedge} = [\min_{1\le i\le N} h_{max}(\mathbf{x}_i),\ \min_{1\le i\le N} h_{min}(\mathbf{x}_i)],$$

for edge detection in the data matrix H. The same initial setting as above has been applied in C for comparison purposes.

K-means clustering is simple and efficient, but it is based on estimating explicit models of the data. When the data distribution is arranged in a complex pattern or corrupted by heavy noise, K-means may not give us a very satisfactory vertex partition. One more advanced clustering approach, which has been shown to handle more complicated structured data, is spectral clustering [1, 22, 32, 40, 35, 30]. It does not require estimating an explicit model of data distribution, but rather employs a spectral analysis of the matrix of point-to-point similarities. However, the real power of spectral clustering is not utilized in the present context. Moreover, in our mesh smoothing application, especially for large meshes, spectral clustering may significantly slow the algorithm down, because it involves the calculation of the c leading

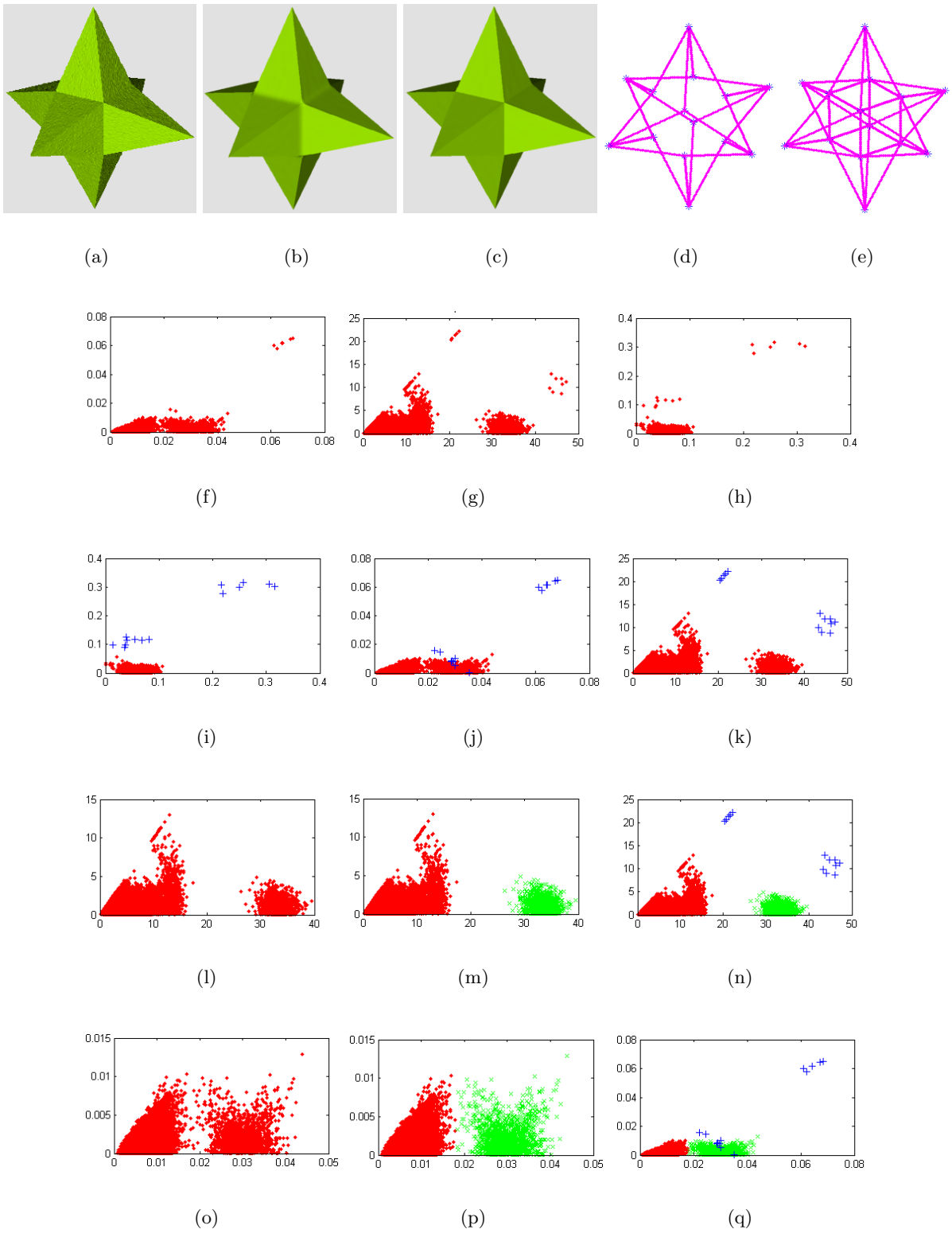


Figure 4: (a) Corrupted Star model; (b) smoothed model based on partition in (n); (c) smoothed model based on partition in (q); (d) 3D plot of feature vertices classified by principal curvatures (blue and green clusters in (n)); (e) 3D plot of feature vertices classified by height intensities (blue and green clusters in (q)); (f) data set H computed for the noisy Star model in (a); (g) data set C; (h) data set G; (i) two clusters by K-means on data set G for corner identification : blue $\rightarrow$ corner; (j) marked corner vertices classified by Gaussian curvature in data set H; (k) marked corner vertices in data set C; (l) temporarily removing the corner cluster from data set C; (m) two clusters by K-means on data that has remained in (l) : red

eigenpairs for the large and full affinity matrix. To maintain high efficiency in our algorithm, we propose another method to improve it in a cheaper way, namely *hierarchical K-means*. Thus, we first implement K-means clustering repeatedly on the unresolved cluster until all sub-clusters consist of vertices with only one type of feature - corner, edge or non-feature. Then we simply combine sub-clusters that contain vertices with the same feature. When this process terminates we have at most three clusters, corner, edge and non-feature, ready for smoothing and further processing. See the column marked Km-N in Table 1 for the number of clustering applications required for different models.

Our numerical results show that, for denoising non-feature vertices, both the umbrella operator and AL with (4a) perform well. The main difference between them is that the umbrella operator regularizes irregular meshes during smoothing, while AL with (4a) keeps the mesh irregularity sampling rate relatively unchanged. Since the umbrella operator is simpler, faster, and produces simultaneous mesh regularization, we apply it by default on the non-feature cluster. Further, we apply MSAL with (4b) on the edge cluster, and keep the corner cluster untouched, see Figs. 3(b) and 4(c). This approach is referred to as our *hybrid denoising algorithm*.

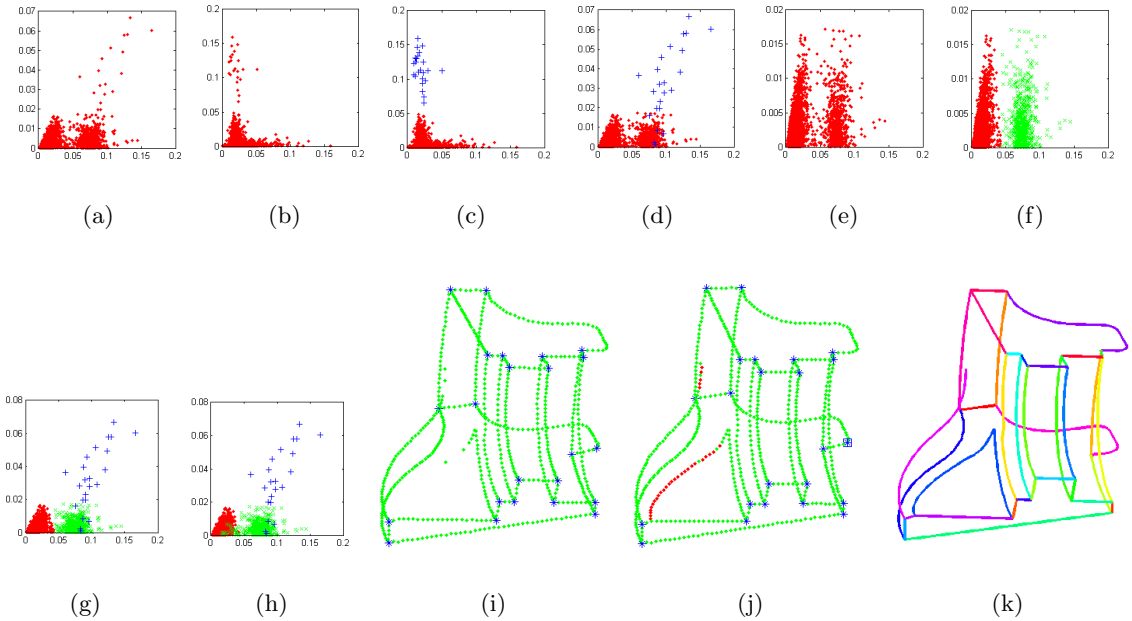
The Star example in Fig. 4 demonstrates a case where using the height intensity data matrix H instead of the principal curvature data matrix C is preferable. Even though in Figs. 4(l) and 4(m) edge and non-feature groups with respect to principal curvatures look well separated, Fig. 4(d) clearly demonstrates that vertices on 12 edges that form a cube are mistakenly grouped into the non-feature cluster with the result that those 12 edges are over-smoothed in the reconstructed model, see Fig. 4(b). Even if we continue to do reclustering on the red non-feature cluster in Fig. 4(m), the result cannot be improved since principal curvature data points corresponding to vertices on the cube are highly mixed with those of vertices on flat regions. In comparison, height intensity data points in Fig. 4(o) provide a better distribution pattern such that only one application of K-means clustering generates a very good partition between edge and non-feature, see Figs. 4(p) and 4(e). Combining with the corner cluster identified in Fig. 4(i), the vertex classification depicted in Fig. 4(q) yields the high quality reconstruction in Fig. 4(c).

3.3 Classification Refinement

Since we are directly working with noisy data, we may not get exact vertex classification even when the more advanced clustering techniques are employed. The approximation of some vertex normals could be badly influenced by heavy noise and likewise for local height intensities.[1] Some non-feature vertices with high frequency noise might be wrongly grouped into the edge or corner clusters; some corner vertices, especially at saddle structures, might be wrongly grouped into the edge or non-feature clusters; some edge vertices on curved creases might be wrongly grouped into the non-feature cluster; and so on. These could result in undesirable bumps in flat regions, defects at edges and corners, or the vanishing of visually meaningful curved ridges, as e.g. in Fig. 6(b). Hence, a post-clustering procedure is proposed to refine the vertex classification.

Take the Fandisk model as an example. Firstly, partition the scaled Gaussian curvature data G into two groups to identify corner vertices; secondly, locate and temporarily remove corners from the data H, see Figs. 5(c) and 5(d); thirdly, employ K-means clustering on the data remaining in H to get the non-feature and edge clusters, see Figs. 5(e) and 5(f); finally,

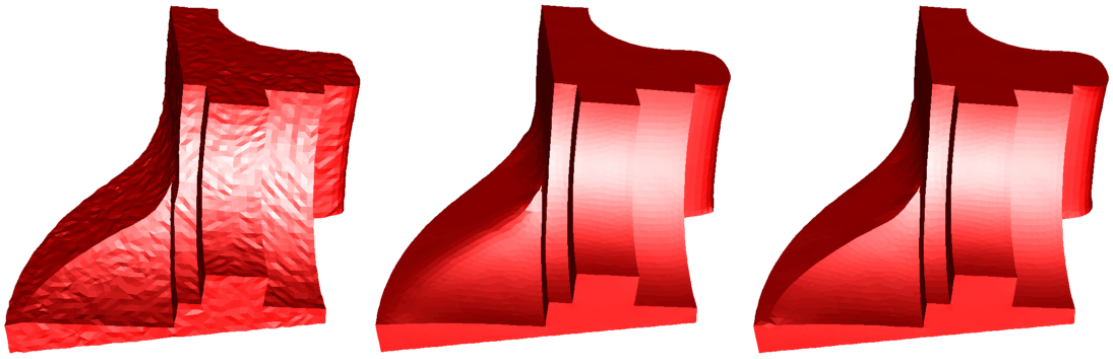
[1] Other methods for estimating the vertex normals [19, 20, 18] have not produced significant improvement in this regard.



(a) (b) (c) (d) (e) (f)

(g) (h) (i) (j) (k)

Figure 5: (a) data set H computed for the noisy Fandisk model depicted in Fig. 6(a); (b) data set G; (c) corner identification in (b); (d) marking corner vertices classified by Gaussian curvature in (a); (e) temporarily removing the corner cluster from (d); (f) two clusters by K-means on the data that has remained in (e) : red $\rightarrow$ non-feature and green $\rightarrow$ edge; (g) adding the corner cluster back into (f); (h) recapturing potential edge vertices back into the edge cluster through the classification refinement procedure ($P = 25$); (i) 3D plot of edge and corner vertices based on clusters in (g), see corresponding model in Fig. 6(b); (j) 3D plot of edge and corner vertices based on refined clusters in (h) where red dots mark the recaptured vertices, see corresponding model in Fig. 6(c); (k) crease detection: each crease is traced by one specific color.



(a) (b) (c)

Figure 6: Recovering curved ridges on the Fandisk model: (a) noisy model; (b) smoothed model based on the vertex partition in Fig. 5(g); (c) smoothed model based on the refined vertex partition in Fig. 5(h).

combine these three clusters and apply the refinement procedure that we design below to improve clustering quality, see Figs. 5(g), 5(h), 5(i) and 5(j).

Refinement Algorithm:

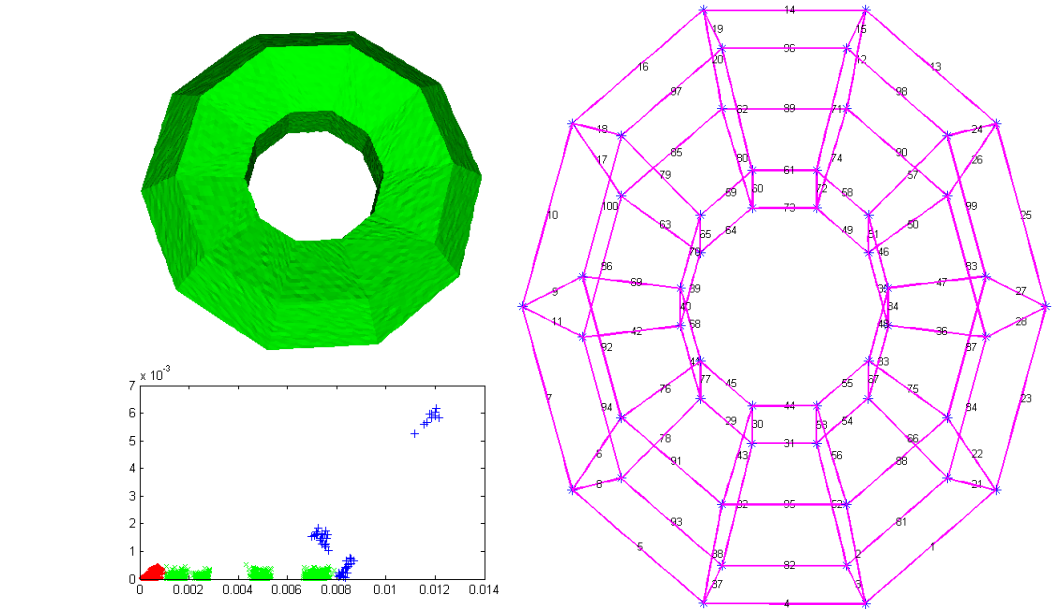
- A corner vertex should have at least three edge vertices in its one-ring neighbor, referred to as edge neighbors. Thus, for each unchecked vertex in the corner cluster:
  1. send it into the non-feature cluster if it does not have any edge neighbor;
  2. send it into the edge cluster if it has fewer than three edge neighbors;
- An edge vertex should have at least two neighboring feature vertices, referred to as feature neighbors, each belonging to one of the edge cluster, corner cluster or potential edge list. Thus, for each unchecked vertex in the edge cluster with the empty potential edge list:
  1. send it into the non-feature cluster if it does not have any feature neighbor;
  2. if it only has one feature neighbor, apply recapturing:
     - 2.1. add the non-feature neighbor with minimal local height intensity to the potential edge list;
     - 2.2. if the total number of vertices on the list exceeds P, send all these vertices back into the non-feature cluster and then move on to the next unchecked edge vertex; otherwise continue;
     - 2.3. begin to check the newest vertex added into the potential edge list: if it has more than one feature neighbor, recapture all vertices on the list from the non-feature cluster into the edge cluster and label them as checked edge vertices and then move on to the next unchecked edge vertex; otherwise do step 2.1 to pick another potential feature neighbor of this edge vertex.

The confidence integer P controls the extent of edge searching. It is the only parameter appearing in our classification refinement algorithm. It should be neither too small nor too large because we want to find as many potential edge vertices as possible whereas we must avoid recapturing by mistake non-feature vertices. Fortunately, it is very intuitive to adjust P up or down according to the 3D plot of feature vertices after running refinement, such as depicted in Fig. 5(j). In addition, the whole refinement procedure is very efficient (see Table 1), so there is no costly trial and error process. In our experience, for most cases the default setting $P = 25$ works very well.

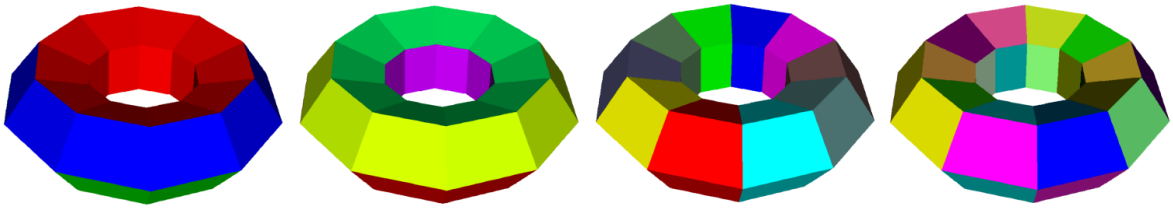
4 Crease Detection and Mesh Segmentation

Since we are able to mark all feature vertices, one quick extension is to detect all sharp creases on CAD-like models. Taking the Donut model in Fig. 7(a) as one example, we consider its feature vertices as a complete, directed and weighted graph. If there is an edge in the original mesh from feature vertex $\mathbf{x}_i$ to $\mathbf{x}_k$, then set the weight $w_{i,k} = h_{i,k}$; otherwise set $w_{i,k} = \infty$. Thus, by implementing the crease detection algorithm described below we are able to locate and label all visually distinct creases on the Donut, see Fig. 7(b).

Subsequently, if some closed piecewise smooth curves have been determined by ordering the labels of creases to form cutting paths, then we are able to carry out the corresponding

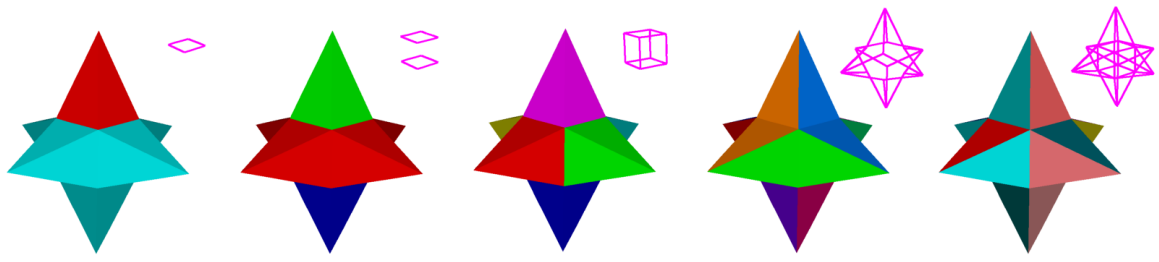


(a) (b)



(c) (d) (e) (f)

Figure 7: (a) Noisy Donut model and vertex classification in the data set H; (b) all creases detected and labeled; (c) segmenting the smoothed mesh into 3 patches; (d) 5 patches; (e) 10 patches; (f) 50 patches.



(a) (b) (c) (d) (e)

Figure 8: Various mesh segmentations for the Star model in Fig. 4 according to input cutting paths sketched in pink: (a) 2 patches; (b) 3 patches; (c) 6 patches; (d) 12 patches; (e) 24 patches.

specific mesh segmentation. This is different from general mesh segmentation problems studied in various articles [27, 43, 2, 23], where the main challenge is to automatically produce segmentation results that are in close agreement with human shape perception. In the current context, the segmentability of a CAD-like shape is quite clear, and all creases that can define different cutting paths are already located. So what we really are concerned with here is how to efficiently make use of available data while respecting user specifications as much as we can. The algorithm is given below. Figs. 7(c) - 7(f) and 8(a) - 8(e) clearly demonstrate the flexibility of our mesh segmentation. Note that we are making one major assumption, namely, that input cutting paths must be closed.

Crease Detection Algorithm:

- Input data: corner vertex set V_c and edge vertex set V_e;
- Initialize: compute weight matrix using mesh connectivity and height intensity;
- Starting point: one random corner vertex in V_c;
    1. from starting point mark edge vertices along path with the smallest weight until arriving at any corner vertex or marked edge vertex;
    2. label the found crease and remove marked edge vertices from V_e;
    3. if the starting corner vertex is not isolated, repeat step 1; otherwise mark it, set another unmarked corner vertex as the starting point and repeat step 1;
- Check convergence if the whole set V_c is marked: exit successfully if V_e is empty, otherwise randomly pick a vertex in V_e as a new starting point and repeat step 1. Note that in this case the found crease may be closed without containing a corner, as in Figs. 3 and 9.

Mesh Segmentation Algorithm:

- Available data: vertex set V and connectivity structure;
- Input data: the list of labels of the creases that form several closed cutting paths;
- Initialize: V_{cut} = vertices on input creases and $V_{interior} = V/V_{cut}$;
- Starting point: one random vertex in $V_{interior}$;
    1. send multiple-ring vertex neighbor of the starting point into one patch-vertex set V_p; stop neighbor chasing at vertices in V_{cut} such that all vertices in the outermost ring are in V_{cut};
    2. paint one-ring face neighbor of all members in $V_p \bigcap V_{interior}$ with the same color and set $V_{interior} = V_{interior}/V_p$;
    3. check if there is any face with all three vertices in $V_p \bigcap V_{cut}$; if yes paint it too;
- Checking convergence: exit successfully if $V_{interior}$ is empty; otherwise start a new patch painting with a different color.

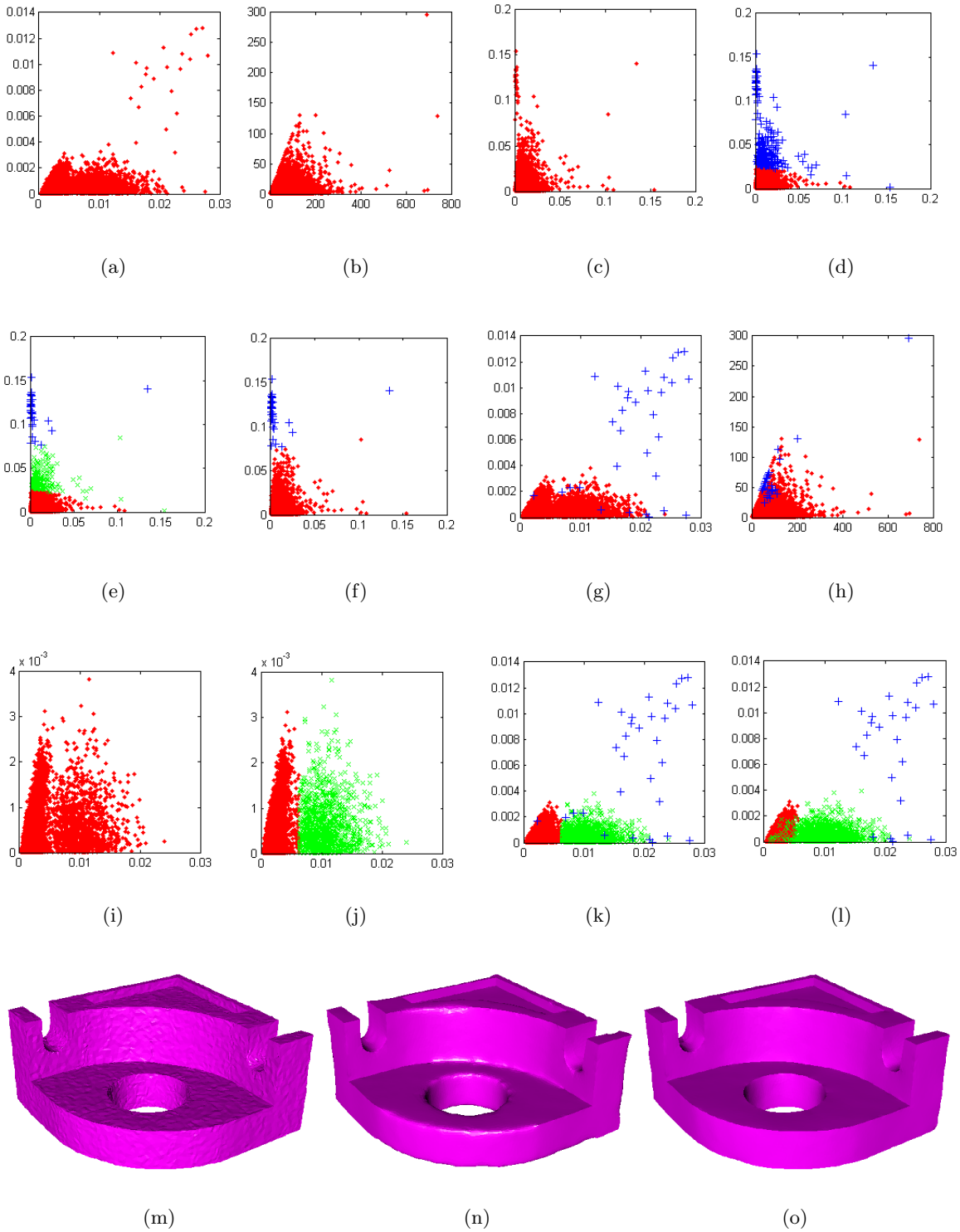


Figure 9: (a) Height intensity data matrix H computed for the noisy Bearing model in (m); (b) principal curvature data matrix C; (c) scaled Gaussian curvature data matrix G; (d) two clusters in G classified by K-means; (e) reclassifying the corner cluster (blue) in (d) by K-means; (f) replacing the corner cluster in (d) with new corner cluster generated in (e) and combining new non-corner cluster (green) with old non-corner cluster in (d); (g) locating corner vertices identified by Gaussian curvature in (a); (h) locating corner vertices in (b); (i) temporarily removing the corner cluster from (g); (j) two clusters by K-means for the data that has remained in (i) : red $\rightarrow$ non-feature and green $\rightarrow$ edge; (k) adding the corner cluster back into (j); (l) classification refinement; (m) noisy Bearing model; (n) smoothed model based on the vertex partition in (k); (o) smoothed model based on the refined vertex partition (l).

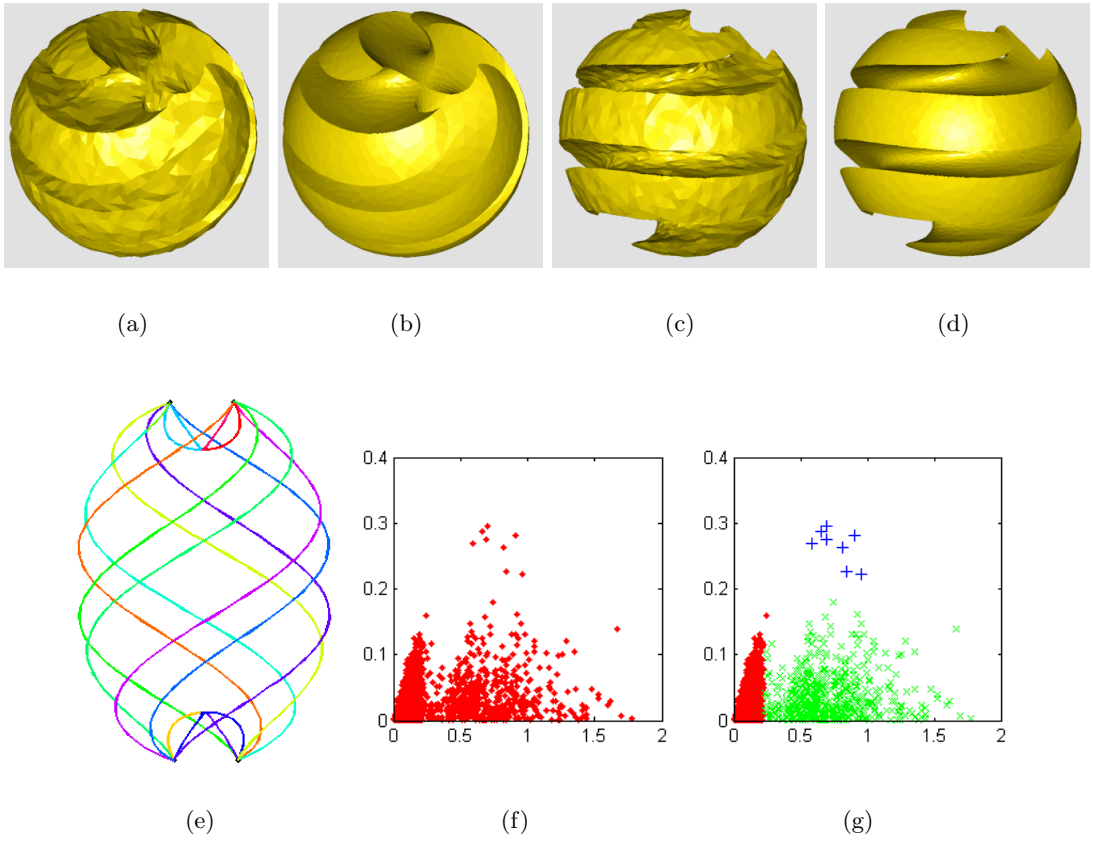


Figure 10: Reconstruction with edge sharpness preservation: (a) noisy Sliced-Sphere model; (b) smoothed model based on the refined vertex classification in (g); (c) another view of (a); (d) corresponding view of (b); (e) detected creases represented in different colors; (f) data set H computed for the noisy model in (a); (g) refined vertex classification.

5 Numerical Results and Discussion

The first set of results presented in Figs. 1 and 2 demonstrates how well edge sharpness is preserved by our reconstructed 3D surface meshes and how efficiently mesh sampling is regularized. The second set of results in Figs. 3, 4, 5 and 6 shows step by step how the data set for the best vertex partition is cheaply built, how K-means clustering efficiently works, how classification quality is optimized and how powerful our corresponding hybrid denoising algorithm can be. Next, in Figs. 7 and 8 we demonstrate the successful extension of our algorithm to sharp crease detection and adaptive mesh segmentation.

A more complicated data pattern is considered in Fig. 9. Just one application of K-means clustering using the data set G is not sufficient to give satisfactory corner identification, see Fig. 9(d). Instead of using the attractive but time-consuming spectral clustering technique, we employ hierarchical K-means developed in Sec. 3.2 to solve this problem in a much cheaper way. After reclustering, combining and refining, we obtain a perfect restoration of the Bearing model in Fig. 9(o). Figs. 9(a) and 9(b) demonstrate that for the noisy Bearing model, the height intensity data set H provides a better pattern distribution for vertex classification than the principal curvature data set C. Figs. 10 and 11 depict more examples where our algorithm

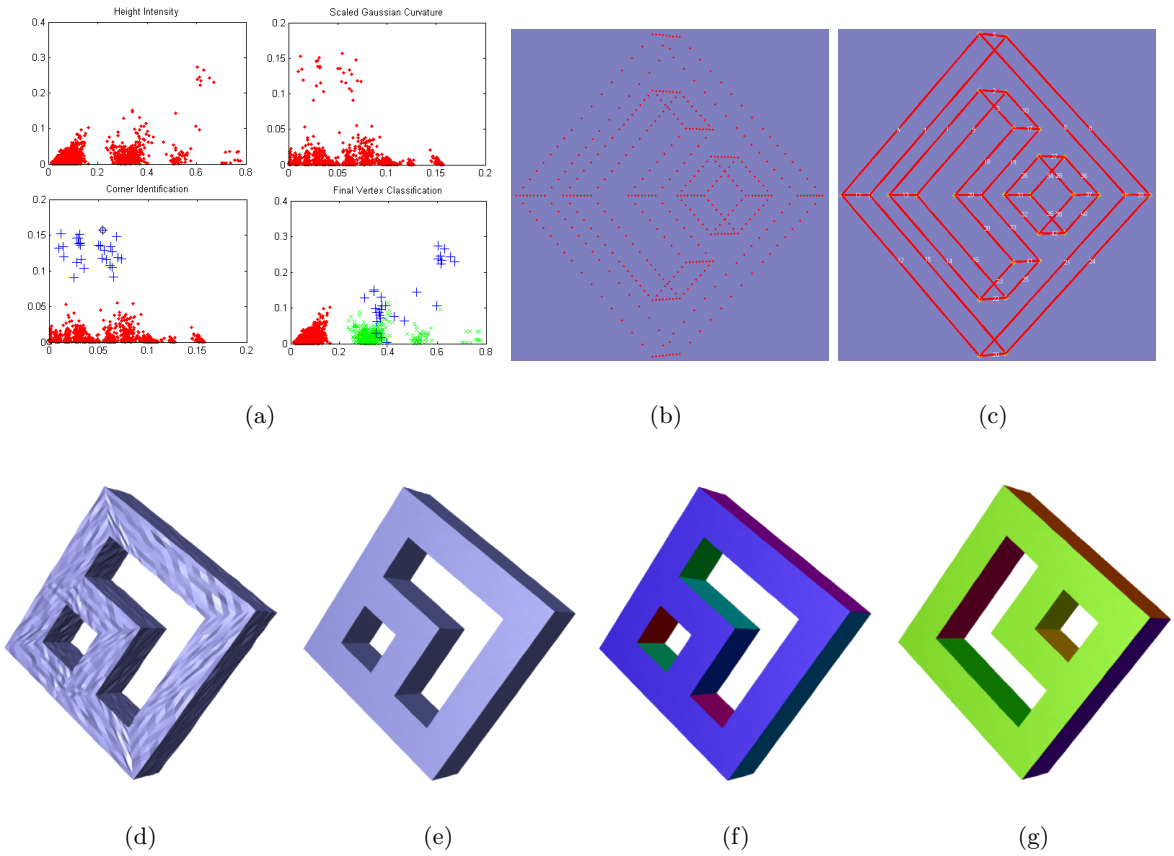
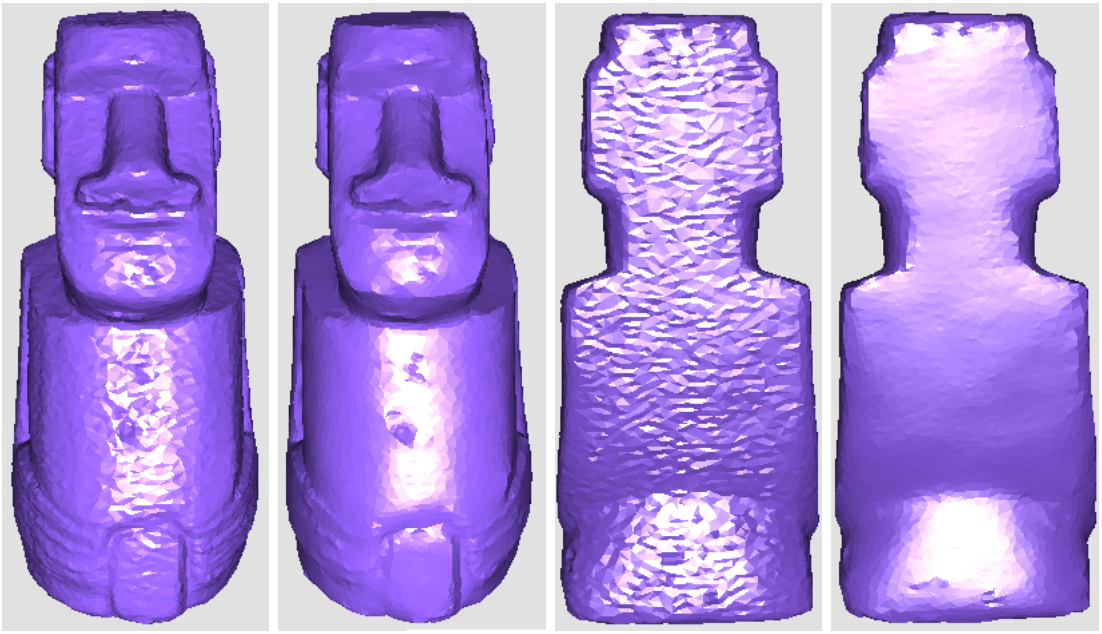


Figure 11: (a) Vertex classification for the noisy Torus model in (d); (b) 3D plot of edge and corner vertices based on clusters in (a); (c) crease detection; (d) noisy Torus model; (e) smoothed model based on the vertex partition in (a); (f) mesh segmentation (16 patches) when all detected creases in (c) are input to construct cutting paths; (g) another view of (f).

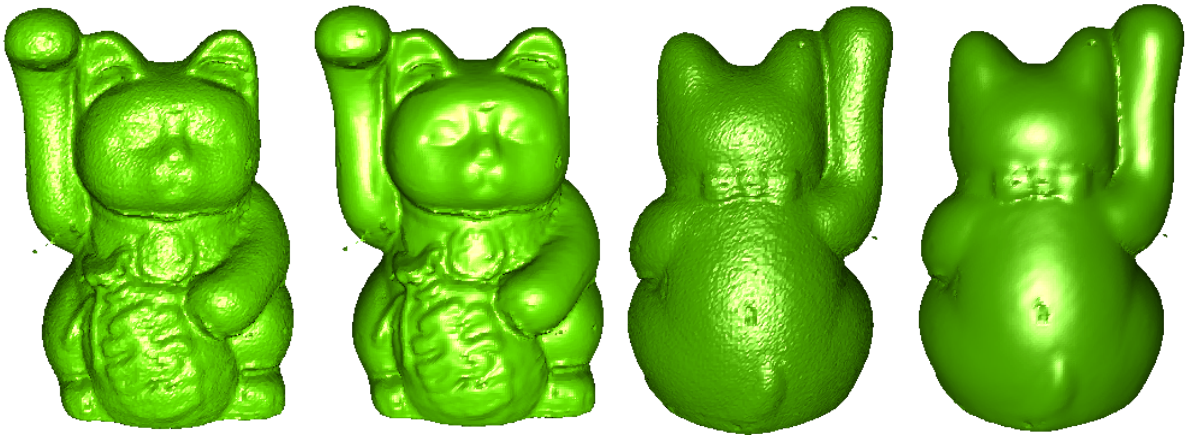
Model	Vertices	Faces	Km-N	Km-T	Re-T	Cr-T	Sm-T	Sm-N	T-T
Ring 3(b)	2.3K	4.6K	1	0.04	0.06	0.06	0.38	4	0.54
Torus 11(e)	2.7K	5.4K	2	0.13	0.09	0.06	0.92	4	1.20
S-Sharp 10(b)	4.3K	8.6K	5	0.29	0.45	0.09	1.05	4	1.88
Fandisk 6(c)	6.5K	13K	2	0.18	0.37	0.11	1.28	4	1.94
Idol 12(b)	10K	20K	1	0.28	1.97	-	0.94	3	3.19
Bearing 9(o)	14K	28K	3	0.48	1.41	0.29	1.67	4	3.85
Star 4(c)	28K	56K	3	0.65	1.06	0.37	3.82	5	5.90
D-Torus 1(b)	35K	70K	4	0.83	1.67	0.32	5.60	5	8.42
M-Neko 13(b)	62K	125K	3	3.76	7.94	-	5.41	3	17.1

Table 1: Runtime data for different models. Km-N: number of applications of K-means; Km-T: total time for K-means clustering; Re-T: time for classification refinement; Cr-T: time for crease detection; Sm-N: number of smoothing iterations; Sm-T: total time for hybrid smoothing; T-T: total execution time for the whole process. Reported times are in seconds; all examples are performed on an Intel Pentium 4 CPU 3.2 GHz machine with 512 RAM.



(a) (b) (c) (d)

Figure 12: Reconstructed results with large featureless regions and intrinsic texture: (a) scanned Idol model containing unknown noise; (b) smoothed model by our hybrid algorithm; (c) back view of (a); (d) back view of (b).



(a) (b) (c) (d)

Figure 13: (a) Scanned Maneki-Neko model with noise; (b) smoothed model with intrinsic texture by our hybrid algorithm; (c) back view of (a) - featureless region with noise; (d) back view of (b) - featureless region without noise.

is employed to yield rather pleasing results in terms of mesh smoothing, segmentation and crease detection.

Figs. 12 and 13 demonstrate that our method not only preserves sharp model features but also retains visually meaningful fine scale components referred to as *intrinsic texture*, even when the model contains large featureless regions as well. In this case we simply divide vertices into feature and non-feature groups, and then use the umbrella operator on the non-feature cluster to obtain enough smoothing while applying MSAL to the feature cluster to capture intrinsic texture. Note that for both these models MSAL can in fact be directly applied without any vertex classification, yielding pleasing results that are comparable to those shown here at somewhat faster total execution times than those listed in Table 1. Our purpose here is to demonstrate that the hybrid method can successfully and efficiently combine different aspects such as mesh regularization on flat regions and careful denoising in the presence of fine scale model features.

In [18] we have demonstrated the power for anti-shrinking of the MSAL scheme as well. Since here we keep corner vertices unchanged and apply MSAL on edges, the hybrid algorithm also helps toward restoring the original volumes, achieving even better results in this regard. Table 1 emphasizes the efficiency of our hybrid algorithm. The number of smoothing iterations Sm-N need not be large: four or five iterations are sufficient for most cases. Although the computation time for vertex classification and crease detection depends on the particular model structure, the total execution time recorded in the last column increases almost linearly with the size of the mesh.

6 Conclusions

We have designed an efficient hybrid algorithm based on specific vertex classification that is capable of denoising 3D surface meshes of models with long edges while preserving edge sharpness, and of generating sufficient smoothing while simultaneously regularizing over featureless regions. Combined with the multiscale method MSAL developed in [18] for models with intrinsic texture we now have a set of algorithms that efficiently handle smoothing and regularization of meshes large and small in a variety of situations. Subsequent crease detection and adaptive mesh segmentation algorithms are developed. These operations can be carried out without any significant cost increase.

Whereas our segmentation algorithm is new as far as we know, we emphasize its restricted nature. The assumption that cutting paths have been detected and are closed is a major one, and in this sense our algorithm is not comparable to the usual, more general mesh segmentation approaches. It is merely an inexpensive, direct extension of our system.

Our method employs hierarchic K-means and a confidence integer P in the classification refinement. This implies that some user intervention in the form of parameter specification must be required. Since a fully automatic mechanism is always desirable, a future challenge is to deduce the optimal parameter P from the mesh structure itself and to introduce some computationally efficient fuzzy clustering techniques that may be more capable of classifying vertices with respect to their features while requiring little or no user intervention.

Acknowledgments

We wish to thank Drs. Nando de Freitas and Alla Sheffer for fruitful discussions. Models presented in this paper have been downloaded off the websites of Klaus Hildebrandt, Alexander

G. Belyaev and AIM@SHAPE Shape Repository.

References

[1] Y. N. Andrew, I. J. Michael, and W. Yair. On spectral clustering: Analysis and an algorithm. In *Proceedings of Neural Information and Processing Systems*, 2002.

[2] M. Attenea, B. Falcidieno, and M. Spagnuolo. Hierarchical segmentation based on fitting primitives. *The Visual Computer*, 22:181–193, 2006.

[3] C. Bajaj and G. Xu. Adaptive surface fairing by geometric diffusion. In *Proceedings of Symposium on Computer Aided Geometric Design*, pages 731–737. IEEE Computer Society, 2001.

[4] C. Bajaj and G. Xu. Anisotropic diffusion on surfaces and functions on surfaces. *ACM Trans. Graphics (SIGGRAPH)*, 22(1):4–32, 2003.

[5] M. J. Black, G. Sapiro, D. H. Marimont, and D. Heeger. Robust anisotropic diffusion. *IEEE trans. image processing*, 7(3):421–432, 1998.

[6] C. Y. Chen and K. Y. Cheng. A sharpness dependent filter for mesh smoothing. *Computer Aided Geometric Design*, 22:376–391, 2005.

[7] C. Y. Chen, K. Y. Cheng, and H. Y. Liao. Fairing of polygon meshes via bayesian discriminant analysis. *J. WSCG*, 12:1–3, 2004.

[8] U. Clarenz, U. Diewald, and M. Rumpf. Anisotropic geometric diffusion in surface processing. In *Proceedings of IEEE Visualization*, pages 397–405, 2000.

[9] U. Clarenz, U. Diewald, and M. Rumpf. Processing textured surfaces via anisotropic geometric diffusion. *IEEE Transactions on Image Processing*, 13(2):248–261, 2004.

[10] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of SIGGRAPH*, pages 317–324, 1999.

[11] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. Anisotropic feature-preserving denoising of height fields and bivariate data. *Graphics Interface*, pages 145–152, 2000.

[12] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* second ed. John Wiley & Sons, INC, 2001.

[13] S. Fleishman, I. Drori, and D. Cohen-Or. Bilateral mesh denoising. *ACM Trans. Graphics SIGGRAPH*, 22(3):950–953, 2003.

[14] K. Fujiwara. Eigenvalues of laplacians on a closed riemannian manifold and its nets. In *Proceedings of AMS*, volume 123, pages 2585–2594, 1995.

[15] R. C. Gonzalez and R. E. Woods. *Digital image processing.* second ed. Prentice-Hall, Engewood Cliffs, NJ, 2002.

[16] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *Proceedings SIGGRAPH*, pages 325–334, 1999.

[17] K. Hildebrandt and K. Polthier. Anisotropic filtering of non-linear surface features. *EUROGRAPHICS*, 23(3):391–400, 2004.

[18] H. Huang and U. Ascher. Fast denoising of surface meshes with intrinsic texture. *Inverse Problems*, 2008. To appear.

[19] S. Jin, R. R. Lewis, and D. West. A comparison of algorithms for vertex normal computation. *The Visual Computer*, 21(1-2):71–82, 2005.

[20] T. Jirka and V. Skala. Gradient vector estimation and vertex normal computation. In *Technical Report No. DCSE/TR-2002-08*. University of West Bohemia in Pilsen, 2002.

[21] T. Jones, F. Durand, and M. Desbrun. Non-iterative, feature preserving mesh smoothing. *ACM Trans. Graphics SIGGRAPH*, 22(3):943–949, 2003.

[22] R. Kannan, S. Vempala, and V. Vetta. On spectral clustering good, bad and spectral. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, 2000.

[23] S. Katz, Leifman, and A. Tal. Mesh segmentation using feature point and core extraction. *The Visual Computer*, 21:649–658, 2005.

[24] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graphics (SIGGRAPH)*, 22(3):954–961, 2003.

[25] L. Kobbelt, S. Campagna, J. Vorsatz, and H. P. Seidel. Interactive multiresolution modeling on arbitrary meshes. In *Proceedings of SIGGRAPH*, pages 105–114, 1998.

[26] G. Lavou, F. Dupont, and A. Baskurt. Constant curvature region decomposition of 3D-mesh by a mixed approach vertex-triangle. *J. WSCG*, pages 245–252, 2004.

[27] R. Liu and H. Zhang. Segmentation of 3D meshes through spectral clustering. In *Proceedings of Computer Graphics and Applications*, pages 298–305. IEEE Computer Society, 2004.

[28] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Proceedings of VisMath*, Berlin, Germany, 2002.

[29] J. M. Morvan and B. Thibert. Smooth surface and triangular mesh: comparison of the area, the normals and the unfolding. In *Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications*, pages 147–158, 2002.

[30] B. Nadler, S. Lafon, R. Coifman, and I. Keverkidis. Diffusion maps, spectral clustering and eigenfunctions of Fokker-Planck operators. In *NIPS*, 2005.

[31] Y. Ohtake, A. Belyaev, and I. A. Bogaevski. Polyhedral surface smoothing with simultaneous mesh regularization. In *Proceedings of Geometric Modeling and Processing*, pages 229–237, 2000.

[32] P. Perona and W. T. Freeman. A factorization approach to grouping. In *Proceedings of the 5th European Conference on Computer Vision*, volume I, pages 655–670, 1998.

[33] G. Sapiro. *Geometric Partial Differential Equations and Image Analysis*. Cambridge, 2001.

[34] J. Shen, B. Maxim, and K. Akingbehin. Accurate correction of surface noises of polygonal meshes. *Int'l J. Numerical Methods in Eng.*, 64(12):1678–1698, 2005.

[35] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[36] T. Shimizu, H. Date, S. Kanai, and T. Kishinami. A new bilateral mesh smoothing method by recognizing features. In *Proceedings of the Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05)*, pages 281–286, 2005.

[37] T. Tasdizen, R. Whitaker, P. Burchard, and S. Osher. Geometric surface smoothing via anisotropic diffusion of normals. In *Proceedings of IEEE Visualization*, pages 125–132, 2002.

[38] G. Taubin. A signal processing approach to fair surface design. *ACM Transactions on Graphics (SIGGRAPH)*, pages 351–358, 1995.

[39] J. Vollmer and R. Mencl. Improved laplacian smoothing of noisy surface meshes. *EUROGRAPHICS*, 18(3):131–139, 1999.

[40] Y. Weiss. Segmentation using eigenvectors: a unifying view. In *International Conference on Computer Vision*, pages 975–982, 1999.

[41] G. L. Xu. Discrete laplace-beltrami operators and their convergence. *Computer Aided Geometric Design*, 21(8):767–784, 2004.

[42] H. Yagou, Y. Ohtake, and A. Belyaev. Mesh smoothing via mean and median filtering applied to face normals. In *Proceedings of Geometric Modeling and Processing*, pages 124–131, 2002.

[43] D. M. Yan, Y. Liu, and W. P. Wang. Quadric surface extraction by variational shape approximation. In *Proceedings of Geometric Modeling and Processing*, pages 73–86, 2006.

[44] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graphics (SIGGRAPH)*, 23(3):641–648, 2004.