

Fast and Effective Feature-Preserving Mesh Denoising

Xianfang Sun, Paul L. Rosin, Ralph R. Martin, and Frank C. Langbein, *Member, IEEE*

Abstract—We present a simple and fast mesh denoising method, which can remove noise effectively, while preserving mesh features such as sharp edges and corners. The method consists of two stages. Firstly, noisy face normals are filtered iteratively by weighted averaging of neighboring face normals. Secondly, vertex positions are iteratively updated to agree with the denoised face normals. The weight function used during normal filtering is much simpler than that used in previous similar approaches, being simply a trimmed quadratic. This makes the algorithm both fast and simple to implement. Vertex position updating is based on the integration of surface normals using a least-squares error criterion. Like previous algorithms, we solve the least-squares problem by gradient descent, but whereas previous methods needed user input to determine the iteration step size, we determine it automatically. In addition, we prove the convergence of the vertex position updating approach. Analysis and experiments show the advantages of our proposed method over various earlier surface denoising methods.

Index Terms—Mesh smoothing, mesh denoising, feature preservation.

I. INTRODUCTION

3D surface mesh models are used in many fields such as CAD, reverse engineering, architectural design, terrain modelling, virtual reality, and computer games. Some 3D surface models are generated via geometric, physical, and solid modelling, whereas others are acquired through 3D measurement technologies such as 3D cameras which produce depth maps, and ship-based sonar. Measured mesh models often contain noise, introduced by the scanning devices and the digitization processes. Such noise can severely impact the usability of mesh models, and it is often desirable to filter it out in a preprocessing step.

In the literature, 3D mesh *denoising* is often confused with surface *smoothing* or *fairing*. Here we wish to emphasise the distinction: in smoothing or fairing, the aim is to generally remove certain high-frequency information in the mesh, whereas in denoising, the aim is to preserve genuine information at all frequencies, while removing any noise or spurious information. In particular, in this paper, we aim to preserve sharp features in the mesh, while removing noise. As Tasdizen et al. [1] point out, much surface smoothing research has been done in the context of surface fairing, with the aim of creating aesthetically pleasing mesh surfaces. However, we follow Shen et al. [2] and clearly distinguish surface fairing

from denoising: the later focuses on removing local errors caused by e.g. 3D sensing technologies, or perhaps by finite element simulation. Mesh *filtering* is another phrase often used to describe similar processes applied to meshes. Filters take one mesh as input and process it to produce another as output. Noise removal is clearly one thing that can be done by such a filter.

In general terms, mesh denoising can be seen as a requirement to adjust vertex positions—vertices affected by noise are not where they should be, and should be moved to the best estimates of their true positions without added noise. (Generally, the vertex positions are the primary measured data, not mesh triangles, which is why we adjust vertex positions). In practice, adjusting vertices directly is not always done, however. New vertex coordinates can be computed in one step or two. *One-step* approaches directly update vertex positions using the original vertex coordinates, and sometimes face normals too, in a neighbourhood around the current vertex. *Two-step* approaches firstly adjust face normals and then update vertex positions using some error minimization criteria based on the adjusted normals. In many cases, a single pass of a one-step or two-step approach does not yield a satisfactory result, and *iterated* operations are performed.

In *iterative two-step* approaches, the iterations can be performed in two distinct ways: the two steps can be coupled as a pair to give an overall iteration procedure informally described as $(\text{Step 1} + \text{Step 2})^n$, or two separate iteration phases can be performed: informally $(\text{Step 1})^{n_1} + (\text{Step 2})^{n_2}$; here n , n_1 and n_2 are numbers of iterations. To distinguish these two schemes, we call these *one-stage* and *two-stage* iteration schemes respectively.

The relative advantages of these two schemes merits detailed investigation. One seeming advantage of one-stage iteration over two-stage iteration is that the former only involves one iteration parameter n , while the latter has two iteration parameters, n_1 and n_2 . However, the two-stage scheme is generally likely to require fewer iterations to obtain a given degree of denoising, i.e., $\max(n_1, n_2) < n$, for two reasons. Firstly, note that normal updating in a two-stage scheme is based on the previously updated normals, while the normal updating in a one-stage scheme is based on the normals computed from the previously updated vertices. Generally, these normals will not be the same; if the normal updating algorithm is optimal in each iteration, then the normals used in a two-stage scheme will be better than those used in a one-stage scheme. Hence, to obtain the same degree of denoising, n is likely to be greater than n_1 . Secondly, a similar consideration applies to vertex updates. Vertex updating is

Manuscript received

The authors are with the School of Computer Science, Cardiff University, 5 The Parade, Roath, Cardiff CF24 3AA, UK. X. Sun is also with the School of Automation Science and Electrical Engineering, Beihang University, Beijing 100083, P.R. China.

based on previously obtained normals. Vertex updating in a two-stage scheme is based on the final (optimal) normals, while vertex updating in a one-stage scheme is based on the most recently updated normals. If the vertex updating algorithm is optimal in each iteration, then n is likely to be greater than n_2 to obtain the same degree of denoising.

The approach proposed in this paper is an *iterative, two-step* method. Our first step adjusts the face normals and our second step updates the vertex positions. Because for an iterative two-step method, the *two-stage* iteration scheme has the advantage of fewer iterations, we adopt a two-stage scheme for our two-step method.

II. PREVIOUS WORK

Many surface *smoothing* and *denoising* methods have been proposed, and to a large degree their differing aims have been conflated—smoothing algorithms have often been suggested for noise removal. The classical Laplacian smoothing method [3], [4] is the fastest and simplest surface smoothing method. However, when applied to a noisy 3D surface, significant shape distortion and surface shrinkage may result in addition to noise removal. To overcome the shrinkage problem, Taubin [5] proposed a filtering method with positive and negative damping factors. A first-order filter with positive damping shrinks and smooths the mesh surface, while a first-order filter with negative damping expands the surface, to compensate for shrinkage. This method is fast and simple, but still has the problem of distortion of prominent mesh features. In addition, if the parameters of the two filters are not chosen carefully, the algorithm can be numerically unstable.

Desbrun et al. [6] introduced diffusion and curvature flow into surface fairing and proposed a simple and numerically stable implicit filtering method, which can deal with irregular meshes. They overcome the problem of shrinkage by re-scaling the mesh to preserve its volume. Again, however, distortion of prominent mesh features occurs.

Taubin’s [5] and Desbrun et al.’s [6] methods can both be considered as filtering methods from the viewpoint of signal processing. The former can be considered as a moving average, or finite impulse response filtering, while the latter can be seen as autoregressive, or infinite impulse response filtering. Combining the above two approaches, Kim and Rossignac [7] developed a general autoregressive moving average filter approach. Through suitable choice of parameters, the filter can act as a lowpass, bandpass, highpass, notch, band amplification or band attenuation filter, thus enabling it to filter out e.g. high-frequency noise and, at the same time, enhance or suppress certain features. However, it is difficult to design a suitable filter that can preserve sharp, high frequency features at the same time as removing noise.

The above methods are all isotropic filtering methods, in which the filter acts independently of direction. This makes it hard for such filters to preserve prominent directional mesh features, particularly edges. Thus, various *anisotropic* filtering schemes have been proposed which smooth surfaces yet preserve edge features simultaneously.

Anisotropic filtering schemes can be divided into four classes. The first class is based on anisotropic geometric diffu-

sion [1], [8]–[11]. Such methods have been used for smoothing height fields and bivariate data [9], level set surfaces [1], and general discretized surfaces [8], [10], [11].

The second class is based on bilateral filters [12], [13]. Fleishman et al. [13] use an iterative one-step approach, in which new vertex coordinates are computed directly from the vertex’s neighbourhood. This approach is relatively fast because a one-step computation is needed for each iteration of vertex updating. However, our experiments show that this method does not always accurately preserve fine features of a mesh. Jones et al.’s [12] robust estimation smoothing is a non-iterative two-step approach. Although non-iterative, this approach is slow because it treats normal smoothing and vertex updating as *global* problems.

The third class is based on combining normal filtering and vertex position updating [14]–[18]. Yagou et al.’s [15], [16] mean, median, and alpha-trimming methods, and Chen and Cheng’s [18] sharpness dependent method are one-stage iterative two-step approaches. Taubin’s [14] anisotropic filtering algorithm and Shen and Barner’s [17] fuzzy vector median filtering approach are two-stage iterative two-step approaches.

The final class of approach comprises an assortment of other methods [2], [19], [20]. Shen et al.’s [2] method consists of three steps: feature-preserving pre-smoothing, feature and non-feature region partitioning, and feature and non-feature region smoothing using two separate approaches. Nehab et al. [19] presented an energy minimization method with the energy being the sum of the position error and the normal error. Diebel et al. [20] used a Bayesian technique for reconstruction and decimation of noisy 3D surface models, converting surface smoothing into an energy minimization problem where the energy is again a sum of the position error and the normal error. The differences between the latter two methods are that the former uses additional information about the measured normals that the latter does not, and the former yields a linear solution unlike the latter.

All the above surface smoothing or denoising methods are based on surface mesh models. In recent years, the point-based surface model has received increasing attention as an alternative surface representation [21]–[23]. Surface smoothing or denoising methods corresponding to the point-based models were also investigated [24], [25]. Although this is a promising research field, this paper only focuses on surface mesh denoising.

The above anisotropic filtering approaches either do not preserve features effectively, or are complex and computationally expensive. From an analysis of earlier anisotropic filtering approaches, we introduce a novel, simple and fast approach which can effectively preserve features. Our approach uses the same two steps as [14] and [17], but it is significantly simpler, and yields results comparable to those of [17].

III. NOTATION

A triangular mesh is denoted by $T = (V, E, F, X)$, where $V = \{i : i = 1, \dots, n\}$ is the vertex set, $E = \{(i, j) : (i, j) \in V \times V\}$ is the edge set, $F = \{(i, j, k) : (i, j), (i, k), (j, k) \in E\}$ is the triangular face set, and $X = \{\mathbf{x}_i : \mathbf{x}_i \in \mathbb{R}^3, i \in V\}$

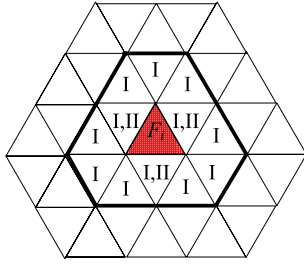


Fig. 1. Face neighbourhoods. Faces labeled I belong to $N_{FI}(i)$; faces labeled II belong to $N_{FII}(i)$

is the vertex coordinate set. We use $|\cdot|$ to denote the cardinality of a set. A vertex, edge, or face is sometimes loosely represented by its corresponding index, i.e. a number i may be used to denote the i^{th} vertex V_i , edge E_i , or face F_i , where this is not ambiguous. The area of face F_i is denoted by A_i ; the normal of F_i is denoted by \mathbf{n}_i . ∂F_i denotes the set of edges that constitute the boundary of face F_i .

In algorithms, various quantities are iteratively updated. We use $'$ to represent the updated value, relative to the current value: e.g. \mathbf{n}'_i denotes the updated value of \mathbf{n}_i .

The 1-ring vertex neighbourhood of a vertex V_i , denoted by $N_V(i)$, is the set of vertices that are connected to V_i by an edge. The set of faces that share a common vertex V_i is denoted by $F_V(i)$. The faces in the 1-ring face neighbourhood of a face F_i can be divided into two types. The first type, denoted by $N_{FI}(i)$, is the set of faces that have a common vertex or edge with the face F_i . The second type, denoted by $N_{FII}(i)$, is the set of faces that share an edge with the face F_i . Fig. 1 shows the two types of face neighbourhoods. Clearly, $N_{FI}(i) \supset N_{FII}(i)$. To refer to the union of F_i and its neighbourhood, we define $N_{FI}^*(i) = N_{FI}(i) \cup \{F_i\}$ and $N_{FII}^*(i) = N_{FII}(i) \cup \{F_i\}$.

IV. NORMAL FILTERING

This Section now considers how we *filter normals* in our two-step approach; the next Section considers how we perform *vertex updating*. In each case, we start by analysing existing approaches, using the results to justify our approach.

A. Previous approaches to normal filtering

Several filtering techniques have been proposed to adjust face normals. An *indirect* approach to normal filtering is given by Jones et al. [12], where the face normals are updated indirectly via vertex updating. A virtual vertex update is first performed using weighted Gaussian filtering. The real vertex positions remain unchanged, but the virtual vertex coordinates are used to compute the new face normals. Although this approach to normal filtering removes noise, it does not consider the requirement to preserve fine features.

Most other normal filtering approaches update the normals directly from the original face normals. Yagou et al. [15], [16] use mean, median, and alpha-trimming filters. Shen and Barner [17] use a fuzzy vector median filter to compute the new normals. We now consider the properties of these filters.

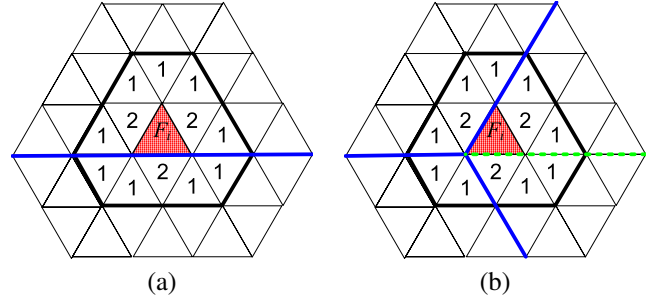


Fig. 2. Faces F_i is adjacent to (a) a ridge feature and (b) a corner feature. Blue lines are ridge lines; the dotted green line is also a ridge line for a different case. The areas between the ridge lines are approximately flat.

The *mean filtering* approach [15] computes the updated normal of a face using area-weighted averaging of the normals of its neighbours:

$$\mathbf{n}'_i = \text{normalise} \left(\frac{1}{\sum_{j \in N_{FI}(i)} A_j} \sum_{j \in N_{FI}(i)} A_j \mathbf{n}_j \right), \quad (1)$$

where $\text{normalise}(\cdot)$ scales a vector length to 1. Note that the scaling coefficient $1/\sum_{j \in N_{FI}(i)} A_j$ in the above formula is actually unnecessary, and only adds computational cost because of the subsequent normalisation. Surprisingly, several other papers also needlessly use a scaling coefficient followed by normalisation: e.g. see [17], [26].

Although mean filtering smooths face normals, it also has the same limitation as Jones et al.'s [12] approach: it destroys fine features of the mesh, and is not feature-preserving.

The *median filtering* approach [15] determines the updated normal \mathbf{n}'_i of a face F_i according to the angles $\angle(\mathbf{n}_i, \mathbf{n}_j)$ between the normals of face F_i and its neighbouring faces $F_j, j \in N_{FI}(i)$:

$$\mathbf{n}'_i = \arg \text{median}_{\mathbf{n}_j} \{w_j \odot \angle(\mathbf{n}_i, \mathbf{n}_j) : j \in N_{FI}(i)\}, \quad (2)$$

where $w_j \odot \angle(\mathbf{n}_i, \mathbf{n}_j)$ means that w_j copies of $\angle(\mathbf{n}_i, \mathbf{n}_j)$ should be included when performing the median operation. Different choices may be used for w_j . A simple choice is to set all $w_j = 1$, i.e. to use an unweighted median. Another choice proposed in [15] is to set $w_j = 2$ when $F_j \in N_{FII}$ and $w_j = 1$ when $F_j \in (N_{FI} \setminus N_{FII})$ (weights are shown in Fig. (2)). Yagou et al. [15] claim that this weighted median filter better preserves features than the unweighted one.

Such median filters preserve ridge and even corner features when there is little or no noise on the mesh. However, when there is a high level of noise, the median filter may yield poor results, as we now explain.

First, consider the surfaces shown in Figs. 2(a) and 2(b) separated by the blue ridge lines (ignore the dotted green line). When there is little or no noise, Eqn. (2) selects as median normal the normal of a face sharing the same flat area as F_i , leading to an updated normal for F_i with little error. However, if the noise level is high, the median normal may come from a different flat area to F_i , and the updated normal of F_i will then have a large error.

Next, consider a corner also including an additional ridge line, shown as a dotted green line in Fig. 2(b). It is clear that

in this case, the median normal will come from a flat area different from the one to which F_i belongs, even if there is no noise on the mesh. Thus, there will be a significant error in the updated normal for F_i .

When using the median filter, because the updated normal \mathbf{n}'_i is selected from the normals \mathbf{n}_j ($j \in N_{FI}(i)$), noise present in \mathbf{n}_j also exists in the updated normal \mathbf{n}'_i . It is hard to eliminate noise solely by using a median filter.

The *alpha-trimming* filtering approach [16] is a compromise between mean and median filters. It updates the normals in the following way:

$$\mathbf{n}'_i = \text{normalise} \left(\sum_{j \in N_{FI}(i)} I_\alpha(j) A_j \mathbf{n}_j \right), \quad (3)$$

where $I_\alpha(j)$ is an indicator function, which equals 0 when $\angle(\mathbf{n}_i, \mathbf{n}_j)$ is in the top or bottom proportion α of all angle values $\angle(\mathbf{n}_i, \mathbf{n}_j)$, $j \in N_{FI}(i)$, and 1 otherwise. When $\alpha = 0$, this filter is the same as the mean filter, and when $\alpha = 0.5$ it is the same as the median filter. Yagou et al. [16] suggested that $\alpha = 0.2$ is a good choice for smoothing models with sharp features. However, it also unavoidably weakens sharp features to some extent. For example, suppose little or no noise exists on the mesh in Fig. 2(a). If we choose $\alpha = 0.2$, we will ignore two faces above the blue line and two below. The remaining faces include faces belonging to a different flat area from F_i , and so the averaging operation will inappropriately include normals from the wrong side of the ridge.

The *fuzzy vector median* filter [17] computes normals using two steps. Firstly, a vector median of the normals is computed using:

$$\mathbf{n}_m = \arg \min_{\mathbf{n}_j} \left\{ \sum_{k \in N_F^*(i)} d(\mathbf{n}_j, \mathbf{n}_k) : j \in N_F^*(i) \right\}, \quad (4)$$

where $N_F^*(i)$ is the union of face F_i and its face neighbourhood, either $N_{FI}^*(i)$ or $N_{FII}^*(i)$ as desired, and $d(\mathbf{n}_j, \mathbf{n}_k)$ is a distance function between \mathbf{n}_j and \mathbf{n}_k , using either the L_p norm $\|\mathbf{n}_j - \mathbf{n}_k\|_p$ or the angle $\angle(\mathbf{n}_j, \mathbf{n}_k)$. Secondly, the updated normal \mathbf{n}'_i is computed as

$$\mathbf{n}'_i = \text{normalise} \left(\sum_{j \in N_F^*(i)} \mathbf{n}_j \tilde{R}_{j,m} \right), \quad (5)$$

where $\tilde{R}_{j,m}$ is a fuzzy relation between \mathbf{n}_j and \mathbf{n}_m , which takes the form of a Gaussian function in [17],

$$\tilde{R}_{j,m} = \exp(-d(\mathbf{n}_j, \mathbf{n}_m)^2 / 2\sigma^2); \quad (6)$$

σ is a parameter that can be either determined by the user, or adaptively computed according to a cost function [17]. (Eqn. 5 differs from the version given in [17] as we have eliminated an unnecessary normalisation, as explained earlier).

As in the case of Yagou et al.'s [15] median filter, again taking the ridge face example shown in Fig. 2, when there is little or no noise, the vector median \mathbf{n}_m given by Eqn. (4) chooses the normal of a face that shares the same flat area with F_i ; when the noise level is high, \mathbf{n}_m can come from a different flat area to F_i . However, because both F_i and its

neighbourhood are used in the computation in Eqn. (4), \mathbf{n}_m is more likely to come from a face sharing the same flat area as F_i than the \mathbf{n}'_i computed using Eqn. (2). Furthermore, if we use the neighbourhood $N_{FII}^*(i)$, it is most likely that \mathbf{n}_m comes from a face sharing the same flat area as F_i even in cases like the one including the additional green dotted ridge line in Fig. 2(b).

The above analysis shows that Eqn. (4) is expected to produce better results than Eqn. (2). However, Eqn. (4) is more time consuming to compute than Eqn. (2). Furthermore, if we replace $N_{FI}(i)$ in Eqn. (2) by $N_{FI}^*(i)$ or $N_{FII}^*(i)$, the result has similar properties to those produced by Eqn. (4).

Because \mathbf{n}_m is selected from the normals \mathbf{n}_j ($j \in N_F^*(i)$), it is subject to noise. As Eqn. (5) is a weighted average of the normals in $N_F^*(i)$, the resulting \mathbf{n}'_i can denoise \mathbf{n}_i . Because a Gaussian weight function gives very low weights to those normals widely different from \mathbf{n}_m , Eqn. (5) can effectively average normals that do not come from the same flat area as \mathbf{n}_m . Compared to the alpha-trimming filter algorithm Eqn. (3), which uses the face areas as weights, the fuzzy vector median filter algorithm can yield normals with lower error, but has the disadvantage of using Gaussian weights which increase the computational costs.

B. Our approach to normal filtering

Taking the above analysis into account, we can now give a new algorithm for normal filtering with low computational cost, while still yielding a face normal with low error even for faces adjacent to a ridge or a corner. We perform normal updates using:

$$\mathbf{n}'_i = \text{normalise} \left(\sum_{j \in N_F^*(i)} h_j \mathbf{n}_j \right), \quad (7)$$

where h_j is a weight function defined as

$$h_j = \begin{cases} f(\mathbf{n}_i \cdot \mathbf{n}_j - T) & \text{if } \mathbf{n}_i \cdot \mathbf{n}_j > T \\ 0 & \text{if } \mathbf{n}_i \cdot \mathbf{n}_j \leq T \end{cases}. \quad (8)$$

Here, $0 \leq T \leq 1$ is a threshold determined by the user, used to control averaging, and $f(x)$ can be any suitably chosen monotonically increasing function for $x \geq 0$. Our experiments showed that $f(x) = x^2$ is a good choice.

The motivation behind the weight function Eqn. (8) is to give high weights to those face normals close to \mathbf{n}_i and low weights to those far from \mathbf{n}_i . In addition, when a face i is adjacent to a ridge or a corner, we want to give null weight to the normal \mathbf{n}_j of any face j which does not share the same surface with face i , and thus having \mathbf{n}_j far from \mathbf{n}_i . Note that $\mathbf{n}_i \cdot \mathbf{n}_j = \cos \angle(\mathbf{n}_i, \mathbf{n}_j)$, so $f(x)$ being a monotonically increasing function implies that the weight function defined by Eqn. (8) satisfies the above requirements.

The choice of T determines how many normals will be used in the weighted averaging operation. $T = 1$ means that only normals with \mathbf{n}_j equal to \mathbf{n}_i are included in the weighted average, while $T = 0$ means that all normals with an angle less than $\pi/2$ from \mathbf{n}_i are used. When the mesh surface has sharp edges, a larger value of T is appropriate, whereas when

the surface is relatively smooth, T should be smaller. We will further discuss the choice of T in Section VI.

Compared to the alpha-trimming filtering algorithm, our algorithm is an improvement since it gives greater weight to normals close to \mathbf{n}_i and eliminates normals that are widely different from \mathbf{n}_i : the alpha-trimming algorithm can get rid of normals close to \mathbf{n}_i as well as those varying widely from it.

Compared to the fuzzy vector median filtering algorithm, our algorithm is less time-consuming because we do not need to compute the vector median, and our weight function is simpler than its Gaussian weights. While that algorithm uses the vector median operation to find a normal whose corresponding face shares the same surface with F_i , we simplify this operation and use \mathbf{n}_i as \mathbf{n}_m in Eqn. (4).

V. VERTEX POSITION UPDATING

A. Previous approaches to vertex updating

After adjusting the face normals, the vertex positions are updated based on these new normals. Several algorithms exist to do this. Taubin [14] used orthogonality between the normal and the three edges of each face on the mesh to give the following family of simultaneous linear equations for vertex position updating:

$$\begin{cases} \mathbf{n}_f \cdot (\mathbf{x}_j - \mathbf{x}_i) = 0 \\ \mathbf{n}_f \cdot (\mathbf{x}_k - \mathbf{x}_j) = 0 \\ \mathbf{n}_f \cdot (\mathbf{x}_i - \mathbf{x}_k) = 0 \end{cases}, \quad \forall f = (i, j, k) \quad (9)$$

Since in general this system of equations has no non-trivial solution, Taubin [14] proposed to solve it in a least squares sense, i.e. to minimize the following error function:

$$e_1(X) = \sum_{k \in F} \sum_{(i,j) \in \partial F_k} (\mathbf{n}'_k \cdot (\mathbf{x}_i - \mathbf{x}_j))^2. \quad (10)$$

Because this least squares problem is linear and its normal equations have a symmetric sparse matrix, it can be solved using various efficient linear system solvers, such as LU, QR, or Cholesky factorization [27]–[29], gradient descent [14], [17], the conjugate gradient method [19], [28], or an multigrid iterative solver [28], [29]. The respective advantages and disadvantages of these solvers were investigated in [28].

In Taubin's paper [14] (see also [17]), the gradient descent method is used to minimize $e_1(X)$, so vertex position updating is implemented as:

$$\mathbf{x}'_i = \mathbf{x}_i + \lambda \sum_{j \in N_V(i)} \sum_{(i,j) \in \partial F_k} \mathbf{n}'_k (\mathbf{n}'_k \cdot (\mathbf{x}_j - \mathbf{x}_i)), \quad (11)$$

where $\lambda > 0$ is the iteration step size. Careful choice of λ is crucial: λ too large results in an unstable algorithm, while λ too small increases the time taken to achieve convergence.

Ohtake et al. [30] proposed the minimization of a different error function defined by

$$e_2(X) = \frac{1}{3} \sum_{i \in F} A_i \|\mathbf{n}_i - \mathbf{n}'_i\|^2, \quad (12)$$

using the L_2 norm. Gradient descent is again used to solve this problem. The solution involves computation of the gradients of A_i , and $A_i(\mathbf{n}'_i \cdot \mathbf{n}_i)$. This algorithm is computationally

more expensive than Taubin's, and again has the problem of choosing a suitable step size λ .

Ohtake et al. [31] also gave another vertex updating algorithm:

$$\mathbf{x}'_i = \mathbf{x}_i + \frac{1}{\sum_{k \in F_V(i)} A_k} \sum_{k \in F_V(i)} A_k \mathbf{n}'_k (\mathbf{n}'_k \cdot (\mathbf{c}_k - \mathbf{x}_i)), \quad (13)$$

where \mathbf{c}_k is the centre of the triangle F_k . This equation can be rewritten as:

$$\mathbf{x}'_i = \mathbf{x}_i + \frac{1}{3 \sum_{k \in F_V(i)} A_k} \sum_{j \in N_V(i)} \sum_{(i,j) \in \partial F_k} A_k \mathbf{n}'_k (\mathbf{n}'_k \cdot (\mathbf{x}_j - \mathbf{x}_i)), \quad (14)$$

Although the reasoning behind this algorithm was not presented in [31], it can be simply explained as the minimization by gradient descent of the error function:

$$e_3(X) = \sum_{k \in F} \sum_{(i,j) \in \partial F_k} A_k (\mathbf{n}'_k \cdot (\mathbf{x}_i - \mathbf{x}_j))^2 \quad (15)$$

with step size $1/6 \sum_{k \in F_V(i)} A_k$.

Unlike Taubin's algorithm, this method does not have the problem of choosing a step size, but it is computationally more expensive since it needs to compute triangle areas.

Next, we explain the relationship between Ohtake et al.'s algorithm [31] and Jones et al.'s algorithm [12]. Rearranging Eqn. (13), we may write:

$$\mathbf{x}'_i = \frac{1}{\sum_{k \in F_V(i)} A_k} \sum_{k \in F_V(i)} A_k \mathbf{p}_k(i), \quad (16)$$

where $\mathbf{p}_k(i) = \mathbf{x}_i + \mathbf{n}'_k (\mathbf{n}'_k \cdot (\mathbf{c}_k - \mathbf{x}_i))$ is the projection of vertex V_i onto the plane Π'_k defined by $\mathbf{n}'_k \cdot (\mathbf{x} - \mathbf{c}_k) = 0$, and Π'_k can be taken as a modification of the plane Π_k containing the original triangular face F_k since its normal has been changed from \mathbf{n}_k to \mathbf{n}'_k .

Eqn. (16) implies that the updated vertex coordinate is the area-weighted average of the projections of the vertex V_i onto the modified planes whose original triangular faces have the common vertex V_i . The vertex update algorithm proposed by Jones et al. [12] also computes a weighted average of the projections of the vertex onto the modified planes, as follows:

$$\mathbf{x}'_i = \frac{1}{s_i} \sum_{k \in F} A_k f(\|\mathbf{c}_k - \mathbf{x}_i\|) g(\|\mathbf{p}_k(i) - \mathbf{x}_i\|) \mathbf{p}_k(i), \quad (17)$$

where both $f(\cdot)$ and $g(\cdot)$ are Gaussian functions, and s_i is a normalizing factor:

$$s_i = \sum_{k \in F} A_k f(\|\mathbf{c}_k - \mathbf{x}_i\|) g(\|\mathbf{p}_k(i) - \mathbf{x}_i\|). \quad (18)$$

Two differences exist between Ohtake et al.'s algorithm and Jones et al.'s algorithm. Firstly, the weight computation in Eqn. (17) is more complicated than that in Eqn. (16). Secondly, the faces used in the computation of Eqn. (17) include all faces of the mesh, while Eqn. (16) includes only the faces of $F_V(i)$. Clearly the former is much more costly to compute. However, the additional complexity of Jones et al.'s algorithm means that it preserves features better, and does not need to be iterated.

The above methods directly use the face normals to update vertex position. Recently, Yu et al. [27] proposed an implicit method to update vertex position through gradient field manipulation. A new gradient field is computed using the local rotation matrix derived from \mathbf{n}_i and \mathbf{n}'_i , and the new gradient field is used in a Poisson equation to compute the updated vertex position. The Poisson equation is linear, and can be solved by various linear system solvers mentioned above. This method is promising but needs to be further developed because of the need for extra computations of the gradient field. Unfortunately, in our attempts to implement the gradient field-based vertex updating method, we were unable to obtain satisfactory results. Perhaps, as Yu et al. [27] suggest, it might be appropriate to use the solution of this algorithm as the initialization for further nonlinear optimization by Ohtake et al.'s algorithm [30].

B. Our approach to vertex updating

In summary, existing vertex updating algorithms either need the user to determine a suitable step size, or are computationally expensive. We now propose a modification to Ohtake et al.'s algorithm [31] to make it more efficient. We simply replace all the area weights in Eqn. (13) by 1, for reasons explained shortly. Thus we have:

$$\mathbf{x}'_i = \mathbf{x}_i + \frac{1}{|F_V(i)|} \sum_{k \in F_V(i)} \mathbf{n}'_k (\mathbf{n}'_k \cdot (\mathbf{c}_k - \mathbf{x}_i)), \quad (19)$$

or, equivalently,

$$\mathbf{x}'_i = \frac{1}{|F_V(i)|} \sum_{k \in F_V(i)} \mathbf{p}_k(i). \quad (20)$$

We now justify our modified algorithm. If a triangle F_k has a large area, there is generally a large distance between the vertices of F_k . Vertices with larger distances from vertex V_i should have a smaller influence on the position update for V_i . Thus, we should give a lower weight to the projection $\mathbf{p}_k(i)$ of V_i onto Π'_k when A_k is larger. Our modified algorithm, which gives the same weight for $\mathbf{p}_k(i)$ whether A_k is large or small, seems more reasonable than Yagou et al.'s original algorithm, which gives large weight to $\mathbf{p}_k(i)$ when A_k is large. Our method is also simpler.

Note that Eqn. (19) can also be written as:

$$\mathbf{x}'_i = \mathbf{x}_i + \frac{1}{3|F_V(i)|} \sum_{j \in N_V(i)} \sum_{(i,j) \in \partial F_k} \mathbf{n}'_k (\mathbf{n}'_k \cdot (\mathbf{x}_j - \mathbf{x}_i)). \quad (21)$$

Comparing Eqn. (21) with Eqn. (11), it can be seen that our algorithm is the same as Taubin's algorithm with the choice $\lambda = 1/3|F_V(i)|$. This avoids the need for the user to choose λ as in Taubin's original algorithm. Going further, using Euler's formula, the average valence of a vertex on a triangular mesh is six [32], and for speed, we can further replace $|F_V(i)|$ by 6. Hence, we can simply choose $\lambda = 1/18$ in Taubin's algorithm.

C. Convergence of vertex updating algorithms

We now prove the convergence of our vertex updating algorithm. Because it is a special case of Ohtake et al.'s

algorithm [31], we first analyze the convergence of Ohtake et al.'s algorithm (15). We start by rewriting the error function $e_3(X)$ defined by Eqn. (15) as:

$$e_3(X) = X^T M X, \quad (22)$$

where $X \in \mathbb{R}^{3|V|}$ is a vector formed by concatenating $\{\mathbf{x}_i, i \in V\}$, and $M \in \mathbb{R}^{3|V| \times 3|V|}$ is a block matrix with each 3×3 block defined by

$$M_{ij} = - \sum_{k:(i,j) \in \partial F_k} A_k \mathbf{n}'_k \mathbf{n}'_k{}^T, \quad M_{ii} = - \sum_{j \in N_V(i)} M_{ij}. \quad (23)$$

In the following, we use capital and small letters, respectively, to distinguish a block and an element in a block matrix, i.e., we use m_{ij} to denote the $\{i, j\}$ element of matrix M .

Obviously, M is a sparse symmetric positive semidefinite matrix. Define a block diagonal matrix D whose 3×3 diagonal block is given by:

$$D_{ii} = 3 \sum_{k \in F_V(i)} A_k I_3. \quad (24)$$

We now have the following lemma:

Lemma 1: Matrix $H = 2D - M$ is a symmetric positive semidefinite matrix.

Proof: Matrix H is obviously symmetric. So, we only need to prove that all its eigenvalues are nonnegative.

Let λ be one of its eigenvalues and Z its corresponding eigenvector. Thus,

$$(2D - M)Z = \lambda Z. \quad (25)$$

Without loss of generality, let the p^{th} element of Z be the largest, i.e. $z_p \geq |z_q|, \forall q \neq p$. Furthermore, suppose z_p is in the i^{th} block when Z is divided into blocks corresponding to D and M . Then we have:

$$\begin{aligned} \lambda &= \sum_q (2d_{pq} - m_{pq}) \frac{z_q}{z_p} \\ &= 6 \sum_{k \in F_V(i)} A_k - \sum_q m_{pq} \frac{z_q}{z_p} \\ &\geq 6 \sum_{k \in F_V(i)} A_k - \sum_q |m_{pq}|. \end{aligned} \quad (26)$$

Consider $\|\mathbf{n}'_k\| = 1$. It can easily be shown that the sum of the absolute values of any row of the matrix $\mathbf{n}'_k \mathbf{n}'_k{}^T$ is not greater than $(1 + \sqrt{3})/2$. Further, by considering Eqn. (23), we can show that

$$\sum_q |m_{pq}| \leq 2(1 + \sqrt{3}) \sum_{k \in F_V(i)} A_k. \quad (27)$$

Substituting (27) into (26), we have $\lambda \geq 0$. ■

Using Lemma 1, we can now prove the convergence of Ohtake et al.'s algorithm [31].

Proposition 1: The vertex updating algorithm (15) is convergent in the sense that the error $e_3(X)$ defined by Eqn. (15) does not increase in each iteration, i.e., $e_3(X') \leq e_3(X)$.

Proof: From (15), the new vertex position vector X' is given by $X' = (I - D^{-1}M)X$. Substituting this into $e_3(X')$, we can compute:

$$\begin{aligned} e_3(X) - e_3(X') &= X^T (2MD^{-1}M - MD^{-1}MD^{-1}M)X \\ &= X^T MD^{-1}(2D - M)D^{-1}MX \end{aligned} \quad (28)$$

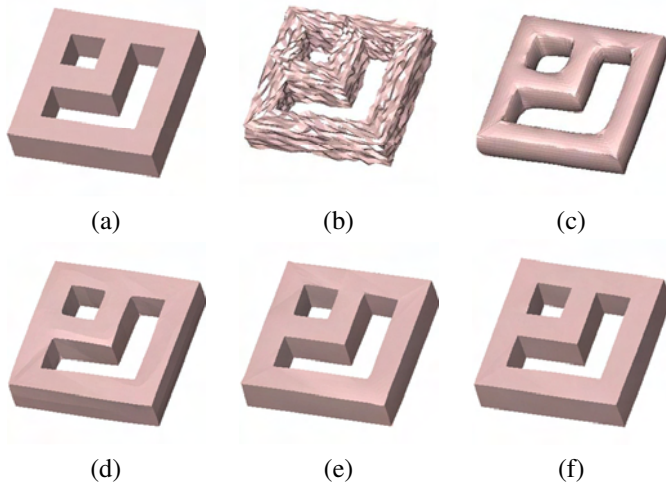


Fig. 3. Denoising of a double-torus model ($|V| = 2,686$, $|F| = 5,376$). (a) Original model, courtesy of MPII, (b) noisy model (Gaussian noise, standard deviation = 0.2 mean edge length), (c) bilateral filtering result ($n = 15$), (d) median filtering ($n = 50$), (e) fuzzy vector median filtering result ($n_1 = 30$, $n_2 = 20$, $\sigma = 0.3$), (f) our result ($n_1 = 30$, $n_2 = 20$, $T = 0.45$).

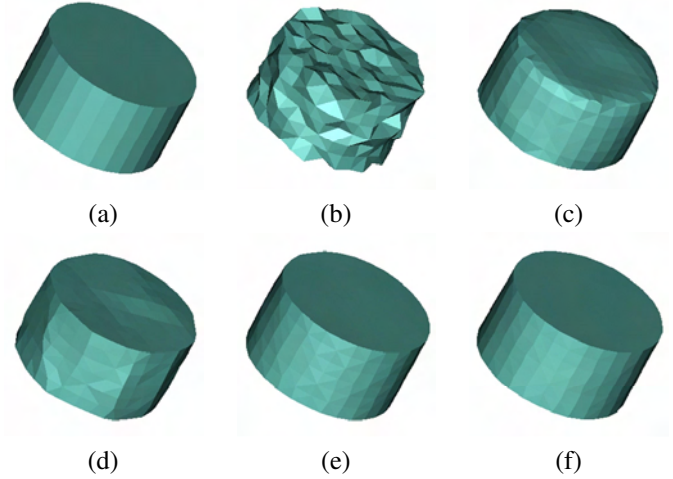


Fig. 4. Denoising of a cylinder model ($|V| = 404$, $|F| = 804$). (a) Original model, (b) noisy model (Gaussian noise, standard deviation = 0.2 mean edge length), (c) bilateral filtering result ($n = 5$), (d) median filtering ($n = 20$), (e) fuzzy vector median filtering result ($n_1 = 25$, $n_2 = 20$, $\sigma = 0.4$), (f) our result ($n_1 = 25$, $n_2 = 20$, $T = 0.4$).

From Lemma 1, we have $e_3(X) - e_3(X') \geq 0$. ■

Proposition 1 directly proves convergence of our vertex updating algorithm.

Corollary 1: The vertex updating algorithm in (21) is convergent in the sense that the error $e_1(X)$ defined by Eqn. (10) does not increase in each iteration, i.e., $e_1(X') \leq e_1(X)$.

Following a similar proof to that for Proposition 1, we can also easily prove the convergence of Taubin’s algorithm.

Corollary 2: If $\lambda \leq 1/3|F_V(i)|_{max}$, then the vertex updating algorithm (11) is convergent in the sense that the error $e_1(X)$ defined by Eqn. (10) does not increase in each iteration, i.e., $e_1(X') \leq e_1(X)$.

VI. RESULTS AND DISCUSSION

This Section demonstrates results of tests carried out on our approach, and discusses optimal choice of parameters.

A. Denoising results

We now compare denoising results of several algorithms including our new method. The next Section considers efficiency of various methods.

The algorithms described in this paper have been implemented in VC++.net. Several experiments were performed on a PC with a 3.2GHz Intel Xeon CPU and 2.0GB of RAM. Both synthetic and measured models were used in our experiments.

We first visually compare results obtained using our algorithm with those from several other algorithms. We then employ two numerical measures to compare the results in more detail. In these comparisons, we show the best results obtained for each approach after fine tuning the parameters. All models have been rendered using flat shading.

Fig. 3 shows denoising results for a model with sharp edges—the double-torus model. It can be seen that bilateral filtering [13] tends to blur sharp edges. Median filtering [15] preserves sharp edges, but makes flat areas uneven. In contrast, fuzzy vector median filtering [17] and our approach

both preserve sharp edges and flat areas. Nevertheless, close examination shows that our approach generates a smoother final surface for this model. We also tested mean filtering [15] and alpha-trimming filtering [16], but both methods blur sharp edges, so we have not illustrated the corresponding results.

Fig. 4 shows the denoising results from a cylinder (first faceted, then triangulated), which has both flat and curved surfaces, and sharp edges. It can be seen that bilateral filtering does not preserve sharp edges. Median filtering preserves the sharp edges, but also introduces spurious additional sharp edges. Fuzzy vector median filtering and our approach preserve both sharp edges and the surface characteristics. There is little visible difference between the latter two results, although our method again seems to lead to a result more faithful to the original surface.

Fig. 5 shows denoising results for the fandisk model. All four approaches preserve most of the sharp edges. Bilateral filtering and median filtering even preserve those sharp edges with small angles between neighboring surfaces, but on the other hand the surfaces in other areas are not particularly smooth. Fuzzy vector median filtering and our approach produce smooth surfaces and preserve most sharp edges, but blur those sharp edges with small angles. Bilateral filtering preserves sharp edges better on this model than on the previous models. The reason seems to be due to the relatively small noise level in this model, and few iterations of filtering are required. We also performed a test on the fandisk model after adding Gaussian noise with standard deviation = 0.2 mean edge length. In this case, bilateral filtering blurs the sharp edges if we try to achieve a reasonably smooth final surface.

Fig. 6 shows denoising results on a mesh model with details at various sizes—the “iH” embossed Stanford Bunny Model. All approaches do well apart from median filtering. Median filtering has a tendency to enhance features in the noisy model, and the resulting surface is not smooth. For this model, perhaps bilateral filtering provides the best overall result, with the

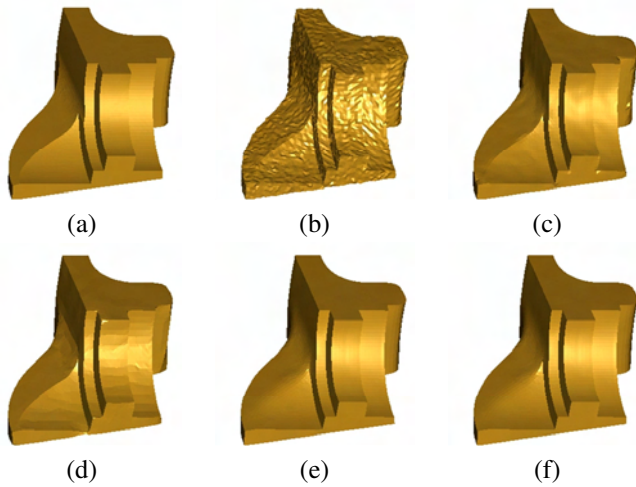


Fig. 5. Denoising of the fandisk model ($|V| = 6,475, |F| = 12,946$). (a) Original model, courtesy of H. Hoppe. (b) noisy model (Gaussian noise, standard deviation = 0.1 mean edge length), (c) bilateral filtering result ($n = 5$), (d) median filtering result ($n = 10$), (e) fuzzy vector median filtering result ($n_1 = 10, n_2 = 10, \sigma = 0.3$), (f) our result ($n_1 = 10, n_2 = 10, T = 0.55$).

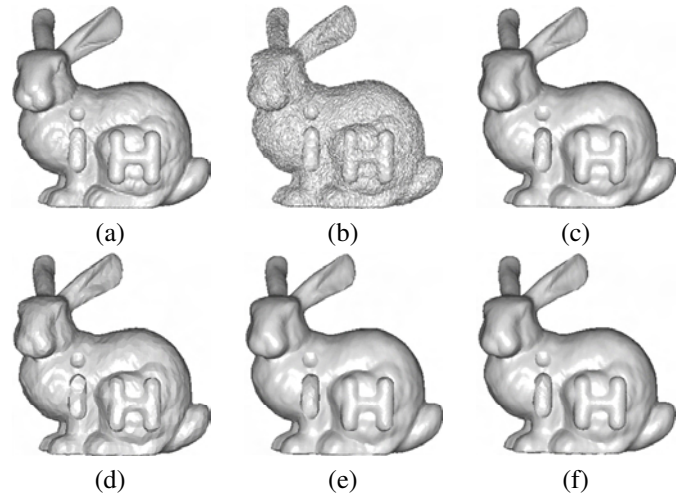


Fig. 6. Denoising of the “iH” embossed Stanford Bunny model ($|V| = 34,834, |F| = 69,451$). (a) Original model, courtesy of A. Belyaev, (b) noisy model (Gaussian noise, standard deviation = 0.2 mean edge length), (c) bilateral filtering result ($n = 5$), (d) median filtering result ($n = 15$), (e) fuzzy vector median filtering result ($n_1 = 5, n_2 = 20, \sigma = 0.3$), (f) our result ($n_1 = 5, n_2 = 20, T = 0.5$).

fuzzy vector median, and to a lesser extent our approach, losing a little of the finer detail.

Figs. 7 and 8 show results of denoising two real, scanned models. All approaches preserve the surface details to some extent. Fuzzy vector median filtering and our approach seem to produce smoother final surfaces than bilateral filtering and median filtering. Moreover, Fig. 7 shows that our method preserve surface details better than the other methods. However, for the face model shown in Fig. 8, areas near to the hole boundary are not well denoised by our algorithm, or the fuzzy vector median algorithm, when compared to the bilateral filtering method. We do not give any special treatment to holes in our algorithm, and future research should consider this problem. A possible solution is to create virtual vertices as suggested by Desbrun et al. [6].

Figs. 9 and 10 give further comparisons of our method to bilateral filtering method when applied to two scanned models. Comparing the forehead and the ears of the lion-dog shown in Fig. 9(b) and (c), it can be seen that our method produces a smoother surface than bilateral filtering, while comparing the hair, and the pits on the tip of the nose and on the right-hand side of the forehead in Figs. 9(d) and (e), one can see that our method preserves detail better than bilateral filtering. Comparing the ripples on the collar in Figs. 10(b) and (c), and the wrinkles on the face in Figs. 10(d) and (e), further verifies that our method preserves detail better. Considering the lips, jaws and throat in Figs. 10(d) and (e) shows that our method produces a similar smoothness to bilateral filtering in this example.

Fig. 11 shows denoising results for the fandisk model under addition of impulse noise (with similar properties to salt and pepper noise used in image processing). Similar results are obtained as in the case of Gaussian added noise.

The above comparisons show that the results from fuzzy vector median filtering and our approach are generally visually similar, and in most cases both produce better results than

TABLE I
 L^2 ERROR COMPARISON ($\times 10^{-3}$):

Model	double-torus	Cylinder	Fandisk	“iH” Bunny
Bilateral	8.1082	17.7088	1.0778	1.6328
Median	1.7264	8.7001	0.9076	1.0707
Fuzzy	1.5017	4.5274	0.9329	1.2385
our Method	1.2774	4.4076	1.0395	1.1481

either bilateral or median filtering. We next use two numerical measures to compare our approach with other approaches, particularly fuzzy vector median filtering.

Many metrics have been proposed in the literature to compare the similarity (or difference) of two mesh surfaces. These include 3D distance metrics [15], [33]–[36], tangential error metrics [34], curvature error metrics [34], normal error metrics [15], [17], [35], and combined methods [34].

We first use the L^2 vertex-based mesh-to-mesh error metric [35] defined by Eqn. (29) to make a numerical comparison.

$$E_v = \sqrt{\frac{1}{3 \sum_{k \in F} A_k} \sum_{i \in V} \sum_{j \in F_V(i)} A_j \text{dist}(\mathbf{x}'_i, T)^2}, \quad (29)$$

where $\text{dist}(\mathbf{x}'_i, T)$ is the L^2 distance between the new vertex \mathbf{x}'_i and a triangle of the reference mesh T which is closest to \mathbf{x}'_i . Table I shows the E_v errors for the various algorithms considered. This numerical comparison generally, but not invariably, is in agreement with the visual comparison results. In general, our approach produces results almost as good as, or better than, than those from other approaches, whether compared visually or numerically.

Because the difference in the results between our approach and the fuzzy vector median filtering is small and their main difference is in the resulting face normals, we also use a normal error metric to further compare them. The error metric used is the mean square angular error (MSAE) [37], which is

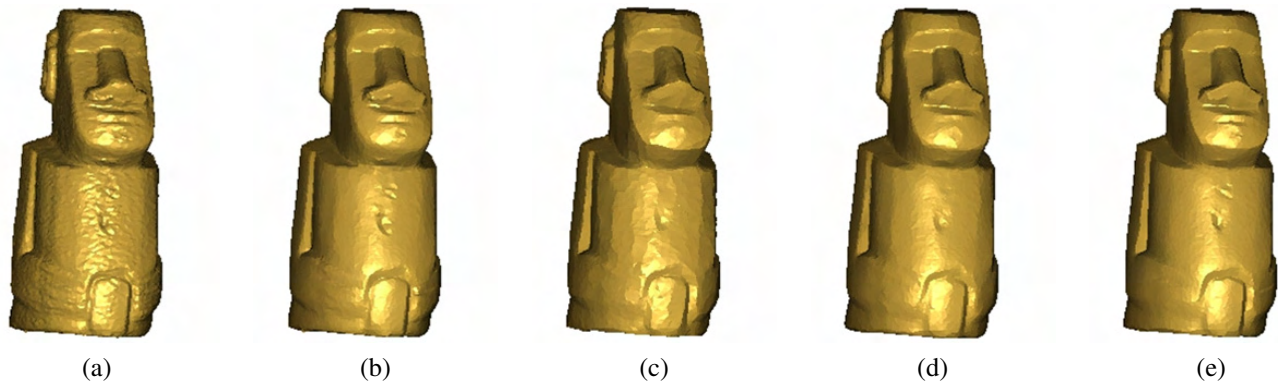


Fig. 7. Denoising of the Moai model ($|V| = 10,002, |F| = 20,000$). (a) Original model, courtesy of Y. Ohtake, (b) bilateral filtering result ($n = 5$), (c) median filtering result ($n = 10$), (d) fuzzy vector median filtering result ($n_1 = 5, n_2 = 10, \sigma = 0.1$), (e) our result ($n_1 = 5, n_2 = 10, T = 0.9$).

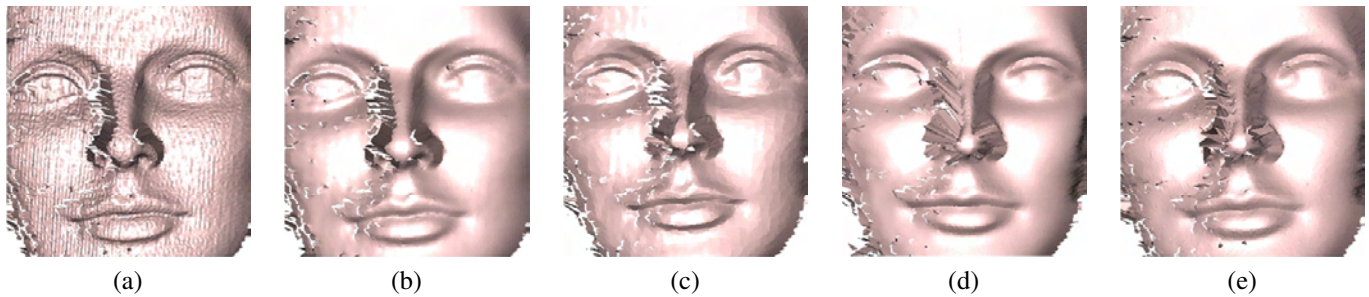


Fig. 8. Denoising of a face model ($|V| = 40,544, |F| = 76,522$). (a) Original model, courtesy of J.-Y. Bouguet, (b) bilateral filtering result ($n = 5$), (c) median filtering result ($n = 10$), (d) fuzzy vector median filtering result ($n_1 = 10, n_2 = 20, \sigma = 0.3$), (e) our result ($n_1 = 10, n_2 = 20, T = 0.65$).

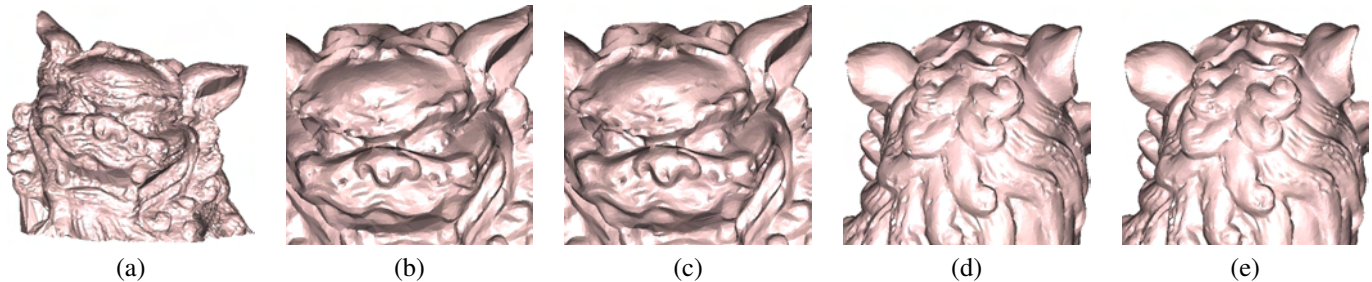


Fig. 9. Denoising of a lion-dog head model ($|V| = 24,930, |F| = 50,000$). (a) Original model, courtesy of Y. Ohtake, (b) front view of the bilateral filtering result ($n = 5$), (c) front view of our result ($n_1 = 5, n_2 = 10, T = 0.6$), (d) back view of the bilateral filtering result, (e) back view of our result.

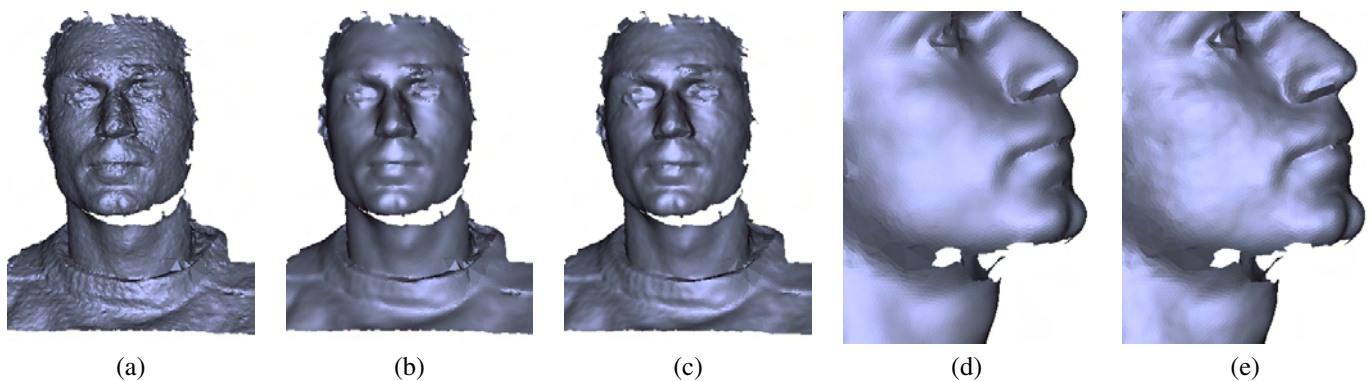


Fig. 10. Denoising of a scanned head model ($|V| = 16,527, |F| = 32,369$). (a) Original model (b) front view of the bilateral filtering result ($n = 5$), (c) front view of our result ($n_1 = 5, n_2 = 10, T = 0.4$), (d) zoom-in side view of the bilateral filtering result, (e) zoom-in side view of our result.

also used in [17]. The definition of MSAE is

$$\text{MSAE} = E[\angle(n_d, n)], \quad (30)$$

where E is the expectation operator and $\angle(n_d, n)$ is the angle

between the denoised normal n_d and the original normal n . Note that we compare the normals produced by each approach with those of the original model (before addition of noise).

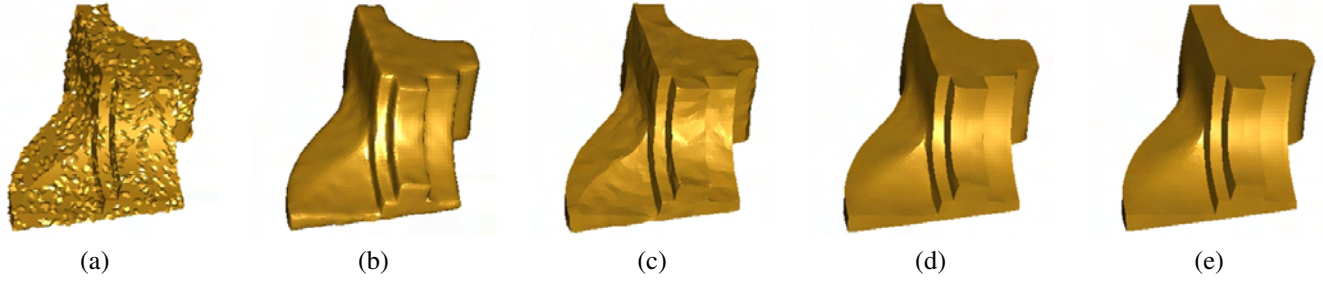


Fig. 11. Denoising a fan disk model with added impulse noise (noise added to 20% of the points, amplitude = 0.5 mean edge length). (a) Noisy model, (b) bilateral filtering result ($n = 5$), (c) median filtering result ($n = 10$), (d) fuzzy vector median filtering result ($n_1 = 20, n_2 = 30, \sigma = 0.3$), (e) our result ($n_1 = 20, n_2 = 30, T = 0.55$).

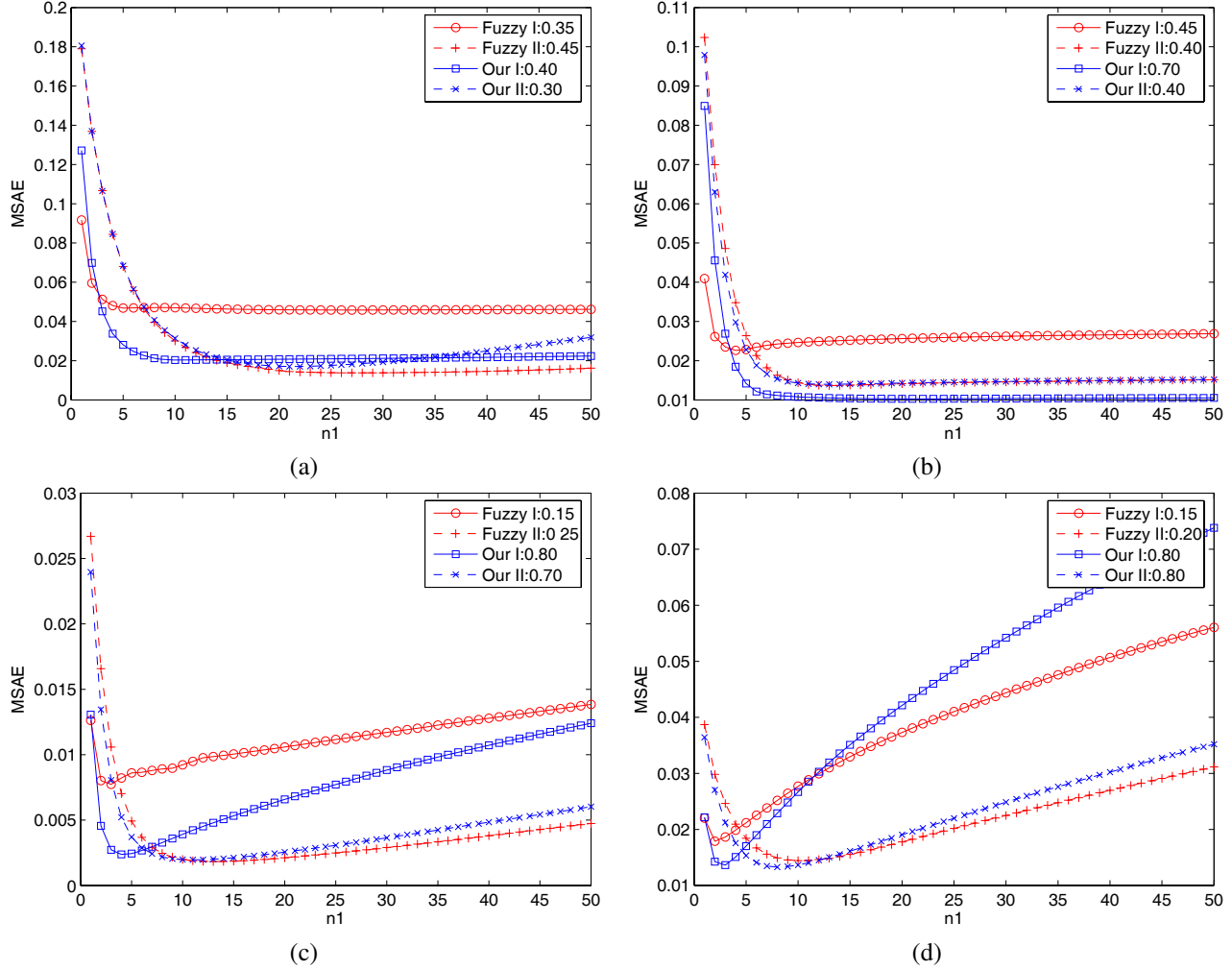


Fig. 12. Normal errors resulting from our approach and fuzzy vector median filtering, after n_1 iterations. I and II indicate the type of face neighbourhood used. Parameter values shown are σ or T , as appropriate. Graphs correspond to: (a) double-torus, (b) cylinder, (c) fan disk, and (d) "iH" Bunny.

Fig. 12 shows the normal errors resulting from our approach and fuzzy vector median filtering for several different models, and how they vary with number of iterations of denoising. Here, we use the distance function based on normal angle in the fuzzy vector median filtering approach (see Eqn. (4)), as it seems to generally yields better results. Overall, there is little difference in normal errors produced by our approach and fuzzy vector median filtering. However, closer analysis shows that for models with many sharp edges our approach

usually results in lower normal error than the fuzzy vector median filtering approach, but for models with mainly smooth surfaces, the opposite is true. The minimal error produced by our approach is usually smaller than that produced by the fuzzy vector median: if the number of iterations is chosen suitably, our approach tends to perform better than fuzzy vector median filtering. We note that the threshold parameters shown in Fig. 12 are different from those used previously in the visual comparisons. There, we chose parameter values which resulted

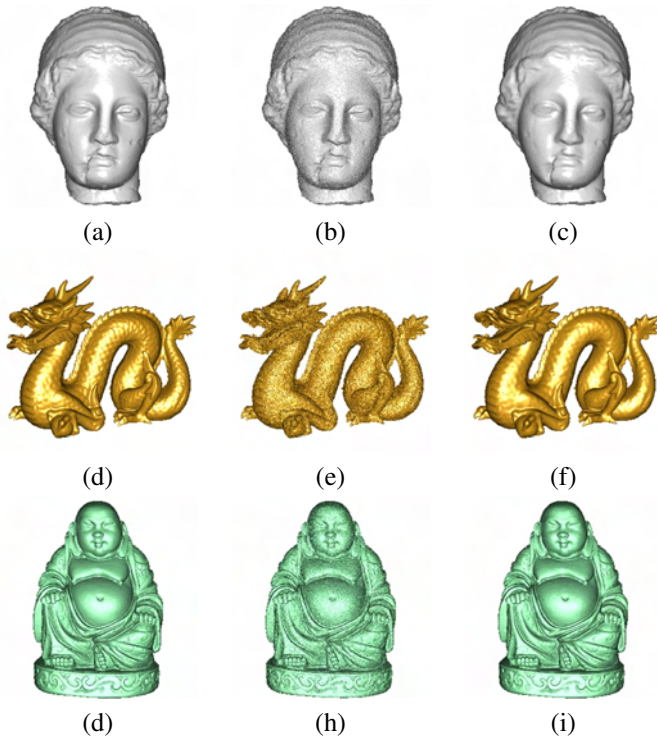


Fig. 13. Denoising results for large models. (a) Original Igea model, data courtesy of Cyberware ($|V| = 134,345, |F| = 268,686$), (b) noisy model (Gaussian noise, standard deviation = 0.1 mean edge length), (c) denoised result ($n_1 = 3, n_2 = 10, T = 0.1$). (d) Original dragon model, data from Stanford University Computer Graphics Laboratory 3D scanning repository ($|V| = 437,645, |F| = 871,414$), (e) noisy model (Gaussian noise, standard deviation = 0.2 mean edge length), (f) denoised result ($n_1 = 10, n_2 = 15, T = 0$). (g) Original Buddha model, courtesy of VCG-ISTI via the AIM@SHAPE Shape Repository ($|V| = 757,490, |F| = 1,514,962$), (h) noisy model (Gaussian noise, standard deviation = 0.1 mean edge length), (i) denoised result ($n_1 = 3, n_2 = 10, T = 0.1$).

in best visual impression for a given fixed number of iterations, while for Fig. 12 we have chosen the parameters which result in the minimum MSAE value over all values of iterations ($n_1 = 1, \dots, 50$).

B. Computational cost

Firstly, we compare the time taken by the normal update step of our algorithm with that by the fuzzy vector median algorithm (again using a distance function based on angle). Table II shows the CPU times recorded in our experiments. For comparative purposes, we performed 50 iterations of normal update for each algorithm, although it is not necessary in practice to use so many iterations. Our approach is much faster than the fuzzy vector median filtering method. When Type I face neighbourhoods are used in both algorithms, our approach is about 40 times faster than the fuzzy vector median filtering method; for Type II face neighbourhoods the advantage is about 12 times. Table II also shows a comparison of the vertex update time taken by Yagou et al.’s method [15] and our modified method, again for 50 iterations. Our modified method is somewhat faster than the former.

Secondly, we compare in Table III the *overall* time taken by our approach with that required by other approaches. The values in parentheses are the numbers of iterations n

or n_1, n_2 we found necessary to satisfactorily denoise the models. Overall, bilateral filtering is generally fastest, as it requires less iterations. However, our approach requires a time similar to that of the bilateral filtering approach, even though requiring more iterations; sometimes, our approach is even faster than bilateral filtering. The other approaches take significantly longer.

Fig. 13 shows denoising results using our approach for large models. The time taken for denoising these models are shown in Table III. Our approach can rapidly and effectively denoise large models.

Overall, our method can provide denoising results of a quality comparable to the slowest of these methods, with nearly the speed of the fastest.

C. Choice of parameters

We now briefly discuss the choice of parameters in our algorithm. These are the threshold T , the number of normal update iterations n_1 , and the number of vertex position update iterations n_2 . We first give a heuristic rule for choice of parameters, then provide an automatic method of choosing parameters T and n_2 .

Fig. 14 shows MSAE for different choices of parameter T . While smaller T usually yields lower normal errors for models with smooth surfaces, for models with sharp edges (e.g. the double-torus model), too small a value of T results in a large error, especially as the number of iterations n_1 increases. Our experiments indicate that $T \in [0.4, 0.6]$ is a good choice for surfaces with sharp edges, and $T \in [0, 0.3]$ is a good choice for smooth surfaces.

Fig. 14 also shows that models with larger noise levels require more iterations of denoising, and models with sharp edges require more iterations than those without. We can also see that too many iterations can cause oversmoothing and result in large normal errors. Our experiments used $n_1 \in [15, 30]$ for sharp-edged models and $n_1 \in [3, 10]$ for smooth models.

In general, for models with higher noise level, the vertices deviate further from their true positions, so n_2 should be larger. Our experiments show that $n_2 = 10$ is generally enough for models with Gaussian noise with 0.1 mean edge length of standard deviation, and $n_2 = 20$ enough for 0.2 mean edge length of standard deviation.

The above suggestions usually allow suitable parameters to be chosen interactively. However, we may require an automatic algorithm. Fortunately, we have proved convergence of the vertex updating algorithm in Section V-C, and we can use this to automatically determine n_2 . We only need to set a maximum number of iterations n_2 and a precision. The algorithm can terminate the vertex updating process before the maximum n_2 iterations when the absolute error of $e_1(X)$ or the relative error between two iterations reaches the required precision.

To determine T automatically, we follow the suggestion of Fleishmann et al. [13]. The user first selects a small region on the surface that is expected to be smooth, and then the minimum value of the dot product of normals of adjacent faces in this region can be computed and used as the threshold T .

TABLE II
 TWO-STAGE TIME COMPARISON (SECONDS):

	Model	Fandisk	“iH” Bunny	Igea	Dragon	Buddha
	Vertices	6,475	34,834	134,345	437,645	757,490
	Triangles	12,946	69,451	268,686	871,414	1,514,962
Normal Update	Our method	0.235	1.438	5.609	19.25	32.375
Type I neighbourhoods	Fuzzy	9.391	48.406	196.531	677.719	1263.06
Normal Update	Our method	0.11	0.609	2.516	8.593	14.313
Type II neighbourhoods	Fuzzy	1.375	7.297	28.421	91.797	177.906
Vertex Position Update	Our method	0.078	0.922	5.047	9.359	21.969
	Yagou	0.125	1.14	6.375	11.031	29.094

TABLE III
 OVERALL TIME COMPARISON (SECONDS, FOR GIVEN NUMBERS OF ITERATIONS):

	Fandisk	“iH” Bunny	Igea	Dragon	Buddha
Bilateral	0.046 (5)	0.313 (5)	1.313 (5)	3.75 (5)	7.094 (5)
Median	0.281 (10)	2.594 (15)	7.25 (10)	33.219 (15)	39.047 (10)
Fuzzy	1.891 (10, 10)	5.141 (5, 20)	12.641 (3, 10)	140.438 (10, 15)	70.093 (3, 10)
Our method	0.078 (10, 10)	0.515 (5, 20)	1.407 (3, 10)	6.672 (10,15)	6.406 (3,10)

VII. CONCLUSION

Many mesh denoising methods have been proposed. Some can both remove noise and preserve mesh features effectively. However, most of these methods are computationally expensive, especially if real-time interactive mesh processing is desired. Through analysis of existing algorithms, we have been able to propose a fast and effective feature-preserving denoising approach.

Experiments show that our approach is as fast as the bilateral filtering approach [13]: e.g. it can denoise the Buddha model of 1.5 million triangles within 7s. However, our approach can preserve sharp edges better than the bilateral filtering approach, providing a final surface quality comparable to that achieved by the fuzzy vector median filtering approach [17], while being significantly faster.

We have conducted many further experiments than space permits to demonstrate in this paper. Most of the models shown here have uniform triangle size; we have found that our algorithm is also suitable for models with nonuniform triangles, even though we do not take the triangle area into account in our algorithm. As our analysis here suggests, area-weighted algorithms sometimes give worse results than algorithms which ignore triangle areas.

We have also experimented with a one-stage iteration scheme (Step 1+Step 2)ⁿ using our algorithm. Experimental results agreed with our analysis in Section I, i.e., that to obtain a given degree of denoising, we usually need $\max(n_1, n_2) < n$, and so the two-stage iteration scheme is faster than the one-stage scheme.

Although our algorithm is simple and efficient for feature-preserving mesh denoising, it also has some problems in common with other algorithms. For example, it cannot effectively deal with meshes with lots of holes. If the normal denoising results are not very good due to undersmoothing or oversmoothing, the vertex updating step may result in non-optimal positions, causing some geometric inconsistencies,

such as mesh folding, self intersections and poorly-shaped triangles. Future research is needed to resolve these problems. Also, convergence analysis is needed on the normal filtering algorithms.

ACKNOWLEDGMENT

We are grateful to the anonymous reviewers for their valuable comments. This work was supported by EPSRC Grant EP/C007972 and NSFC Grant 60674030.

REFERENCES

- [1] T. Tasdizen, R. Whitaker, P. Burchard, and S. Osher, “Geometric surface smoothing via anisotropic diffusion of normals,” in *Proceedings of the Conference on Visualization 2002*. IEEE Computer Society, 2002, pp. 125–132.
- [2] J. Shen, B. Maxim, and K. Akingbehin, “Accurate correction of surface noises of polygonal meshes,” *International Journal for Numerical Methods in Engineering*, vol. 64, no. 12, pp. 1678–1698, 2005.
- [3] D. A. Field, “Laplacian smoothing and delaunay triangulations,” *Communications in Numerical Methods in Engineering*, vol. 4, pp. 709–712, 1988.
- [4] J. Vollmer, R. Mencl, and H. Müller, “Improved laplacian smoothing of noisy surface meshes,” *Computer Graphics Forum*, vol. 18, no. 3, pp. 131–138, 1999.
- [5] G. Taubin, “A signal processing approach to fair surface design,” in *SIGGRAPH’95 Conference Proceedings*, 1995, pp. 351–358.
- [6] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, “Implicit fairing of irregular meshes using diffusion and curvature flow,” in *Proceedings of SIGGRAPH’99*, 1999, pp. 317–324.
- [7] B. Kim and J. Rossignac, “Geofilter: Geometric selection of mesh filter parameters,” *Computer Graphics Forum*, vol. 24, no. 3, pp. 295–302, 2005.
- [8] U. Clarenz, U. Diewald, and M. Rumpf, “Anisotropic geometric diffusion in surface processing,” in *Proceedings of the Conference on Visualization 2000*. IEEE Computer Society, 2000, pp. 397–405.
- [9] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, “Anisotropic Feature-Preserving denoising of height fields and bivariate data,” in *Graphics Interface’2000 Proceedings*, 2000, pp. 145–152.
- [10] C. L. Bajaj and G. Xu, “Anisotropic diffusion of surfaces and functions on surfaces,” *ACM Trans. Graphics*, vol. 22, no. 1, pp. 4–32, 2003.
- [11] K. Hildebrandt and K. Polthier, “Anisotropic filtering of non-linear surface features,” *Computer Graphics Forum*, vol. 23, no. 3, pp. 391–400, 2004.

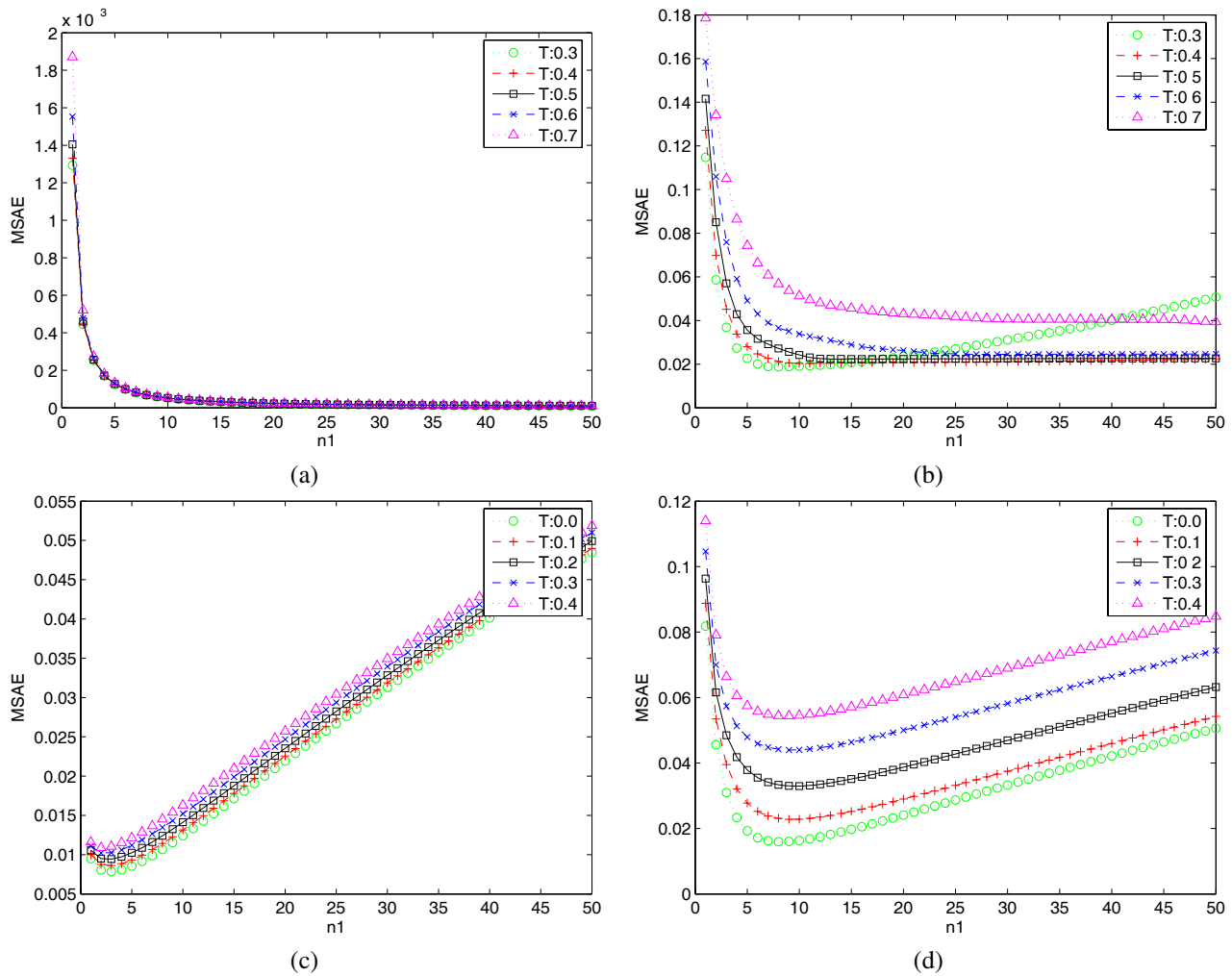


Fig. 14. Normal error for different choice of parameter T . (a,b): double-torus model, (c,d) dragon model; (a,c) noise standard deviation = 0.05 mean edge length, (b,d) noise standard deviation = 0.2 mean edge length.

[12] T. R. Jones, F. Durand, and M. Desbrun, "Non-iterative, feature-preserving mesh smoothing," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 943–949, 2003.

[13] S. Fleishman, I. Drori, and D. Cohen-Or, "Bilateral mesh denoising," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 950–953, 2003.

[14] G. Taubin, "Linear anisotropic mesh filtering," IBM Research Report RC22213(W0110-051), IBM T.J. Watson Research Center, October 2001.

[15] H. Yagou, Y. Ohtake, and A. G. Belyaev, "Mesh smoothing via mean and median filtering applied to face normals," in *Proceedings of Geometric Modeling and Processing*, 2002, pp. 124–131.

[16] H. Yagou, Y. Ohtake, and A. G. Belyaev, "Mesh denoising via iterative alpha-trimming and nonlinear diffusion of normals with automatic thresholding," in *Computer Graphics International 2003 (CGI'03)*, 2003, pp. 28–34.

[17] Y. Shen and K. E. Barner, "Fuzzy vector median-based surface smoothing," *IEEE Trans. Visualization and Computer Graphics*, vol. 10, no. 3, pp. 252–265, 2004.

[18] C.-Y. Chen and H.-Y. Cheng, "A sharp dependent filter for mesh smoothing," *Computer Aided Geometric Design*, vol. 22, pp. 376–391, 2005.

[19] D. Nehab, S. Rusinkiewicz, J. Davis, and R. Ramamoorthi, "Efficiently combining positions and normals for precise 3D geometry," *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 24, no. 3, pp. 536–543, 2005.

[20] J. R. Diebel, S. Thrun, and M. Brünig, "A bayesian method for probable surface reconstruction and decimation," *ACM Trans. Graphics*, vol. 25, no. 1, pp. 39–59, 2006.

[21] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, "Point set surfaces," in *Proceedings of the conference on Visualization '01*. IEEE Computer Society, 2001, pp. 21–28.

[22] L. Kobbelt and M. Botsch, "A survey of point-based techniques in computer graphics," *Computers & Graphics*, vol. 28, no. 6, pp. 801–814, 2004.

[23] A. Adamson and M. Alexa, "Anisotropic point set surfaces," in *Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*. ACM Press, 2006, pp. 7–13.

[24] S. Fleishman, D. Cohen-Or, and C. T. Silva, "Robust moving least-squares fitting with sharp features," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 544–552, 2005.

[25] Y. Ohtake, A. Belyaev, and M. Alex, "Sparse low-degree implicit surfaces with applications to high quality rendering, feature extraction, and smoothing," in *Third Eurographics Symposium on Geometry Processing*, 2005, pp. 149–158.

[26] T. Mashiko, H. Yagou, D. Wei, Y. Ding, and G. Wu, "3D triangle mesh smoothing via adaptive MMSE filtering," in *Proceedings of the Fourth International Conference on Computer and Information Technology (CIT'04)*. IEEE Computer Society, 2004, pp. 734–740.

[27] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H. Shum, "Mesh editing with poisson-based gradient field manipulation," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 644–651, 2004.

[28] M. Botsch, D. Bommes, and L. Kobbelt, "Efficient linear system solvers for mesh processing," vol. 3604, 2005, pp. 62–83.

[29] L. Shi, Y. Yu, N. Bell, and W.-W. Feng, "A fast multigrid algorithm for mesh deformation," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 1108–1117, 2006.

[30] Y. Ohtake, A. Belyaev, and H.-P. Seidel, "Mesh smoothing by adaptive

and anisotropic gaussian filter applied to mesh normals,” in *Vision, Modeling, and Visualization 2002*, 2002, pp. 203–210.

- [31] Y. Ohtake, A. Belyaev, and I. Bogaevski, “Mesh regularization and adaptive smoothing,” *Computer-Aided Design*, vol. 33, no. 11, pp. 789–800, 2001.
- [32] O. Sorkine, D. Cohen-Or, R. Goldenthal, and D. Lischinski, “Bounded-distortion piecewise mesh parameterization,” in *Proceedings of the Conference on Visualization 2002*. IEEE Computer Society, 2002, pp. 355–362.
- [33] P. Cignoni, C. Rocchini, and R. Scopigno, “Metro: Measuring error on simplified surfaces,” *Computer Graphics Forum*, vol. 17, no. 2, pp. 167–174, 1998.
- [34] S.-J. Kim, S.-K. Kim, and C.-H. Kim, “Discrete differential error metric for surface simplification,” in *Proceedings of the 10th Pacific Conference on Computer Graphics and Application (PG’02)*. IEEE Computer Society, 2002, pp. 276–283.
- [35] A. Belyaev and Y. Ohtake, “A comparison of mesh smoothing methods,” in *Proceedings of Israel-Korea Bi-National Conference on Geometric Modeling and Computer Graphics*, 2003, pp. 83–87.
- [36] P. Rondao Alface, M. De Craene, and B. Macq, “Three-dimensional image quality measurement for the benchmarking of 3d watermarking schemes,” *Proceedings of SPIE*, vol. 5681, pp. 230–240, 2005.
- [37] A. Nehorai and M. Hawkes, “Performance bounds for estimation vector systems,” *IEEE Trans. Signal Processing*, vol. 48, no. 6, pp. 1737–1749, 2000.



Xianfang Sun received his BSc degree in Electrical Automation from Hubei University of Technology in 1984. He received his MSc and PhD degrees in Control Theory and its Applications from Tsinghua University in 1991, and the Institute of Automation, Chinese Academy of Sciences in 1994, respectively. He joined Beihang University (BUAA) as a Post-Doctorial Fellow, where he became an Associate Professor in 1997 and a Professor in 2001. Currently, he is on leave from Beihang University and is a Research Associate at Cardiff University. His

research interests include computer vision and graphics, pattern recognition and artificial intelligence, system identification and filtering, fault diagnosis and fault-tolerant control. He has completed many research projects and published over 50 papers. He is on the editorial board of “Acta Aeronautica et Astronautica Sinica”. He is also a member of the Committee of Technical Process Failure Diagnosis and Safety, Chinese Association of Automation.



Paul L. Rosin received the B.Sc. degree in computer science and microprocessor systems in 1984 from Strathclyde University, Glasgow, U.K., and the Ph.D. degree in information engineering from City University, London, U.K., in 1988. He was a Research Fellow at City University, developing a prototype system for the Home Office to detect and classify intruders in image sequences. He worked on the Alvey project “Model-Based Interpretation of Radiological Images” at Guy’s Hospital, London, before becoming a Lecturer at Curtin University of Technology, Perth, Australia, and later a Research Scientist at the Institute for Remote Sensing Applications, Joint Research Centre, Ispra, Italy. He then returned to the U.K., becoming a Lecturer at the Department of Information Systems and Computing, Brunel University, London. Currently, he is a Senior Lecturer at the School of Computer Science, Cardiff University, Cardiff, U.K. His research interests include the representation, segmentation, and grouping of curves, knowledge-based vision systems, early image representations, low level image processing, machine vision approaches to remote sensing, medical and biological image analysis, and the analysis of shape in art and architecture.



Ralph R. Martin has been working in the field of CAD/CAM since 1979. He obtained his PhD in 1983 from Cambridge University for a dissertation on “Principal Patches”, and since then has worked his way up from Lecturer to Professor at Cardiff University. He has published over 170 papers and 10 books covering such topics as solid modelling, surface modelling, intelligent sketch input, vision based geometric inspection, geometric reasoning and reverse engineering. He is a Fellow of the Institute of Mathematics and its Applications, and a Member

of the British Computer Society. He is on the editorial boards of “Computer Aided Design”, the “International Journal of Shape Modelling”, the “International Journal of CAD/CAM”, and “Computer-Aided Design and Applications”. He has also been active in the organisation of many conferences.



Frank C. Langbein (M’00) received a Diploma in Mathematics from the University of Stuttgart in 1998. He obtained his PhD in 2003 from Cardiff University for a dissertation on “Beautification of Reverse Engineered Geometric Models”. Since then he has been a Lecturer in Computer Science at Cardiff University. His research interests in geometric modelling include detection and representation of design intent in geometric models, geometric constraints, mesh processing, free-form surfaces and reverse engineering. He is also interested in quantum

modelling and quantum information and works on simulation, identification and control of quantum systems. He is a member of the American Mathematical Society and the IEEE.