Random walks for feature-preserving mesh denoising

Xianfang Sun [a,b,*], Paul L. Rosin [a], Ralph R. Martin [a], Frank C. Langbein [a]

[a] *School of Computer Science, Cardiff University, UK*

[b] *School of Automation Science and Electrical Engineering, Beihang University, China*

Abstract

An approach to mesh denoising based on the concept of random walks is examined. The proposed method consists of two stages: face normal filtering, followed by vertex position updating to integrate the denoised face normals in a least-squares manner. Face normal filtering is performed by weighted averaging of normals in a neighbourhood. A novel approach to determining weights is to compute the probability of arriving at each neighbour following a fixed-length random walk of a virtual particle starting at a given face of the mesh. The probability of the particle stepping from its current face to some neighbouring face is a function of the angle between the two face normals, based on a Gaussian distribution whose variance is adaptively adjusted to enhance the feature-preserving property of the algorithm. The vertex position updating procedure uses the conjugate gradient algorithm for speed of convergence. Analysis and experiments show that random walks of different step lengths yield similar denoising results. Our experiments show that, in fact, iterative application of a one-step random walk in a progressive manner effectively preserves detailed features while denoising the mesh very well. This approach is faster than many other feature-preserving mesh denoising algorithms.

Key words: Mesh denoising, Mesh smoothing, Random walk, Feature preservation

* Corresponding author. Tel.: +44 (0) 29 2087 9355; fax: +44 (0) 29 2087 4598.
Email addresses: `Xianfang.Sun@cs.cardiff.ac.uk` (Xianfang Sun), `Paul.Rosin@cs.cardiff.ac.uk` (Paul L. Rosin), `Ralph.Martin@cs.cardiff.ac.uk` (Ralph R. Martin), `F.C.Langbein@cs.cardiff.ac.uk` (Frank C. Langbein).

1 Introduction

3D surface *mesh denoising* has been an active research field for several years. Although much progress has been made, mesh denoising technology is still not mature. The presence of intrinsic fine details and sharp features in a noisy mesh makes it hard to simultaneously denoise the mesh and preserve the features. In this paper, we present a new approach to feature-preserving mesh denoising based on a random walk model.

Mesh denoising is usually posed as a problem of adjusting vertex positions while keeping the connectivity of the mesh unchanged. In the literature, mesh denoising is often confused with surface *smoothing* or *fairing* (Sun et al., 2007), because all of them use vertex adjustment to make the mesh surface smooth. However, they have different purposes, and different algorithms are needed to meet their specific requirements, and we should keep in mind the distinctions. Tasdizen et al. (2002), Shen et al. (2005) and Sun et al. (2007) already distinguish the different concepts well. Here, we only emphasise that the main goal of mesh fairing is related to aesthetics, while the goal of mesh denoising has more to do with fidelity, and mesh smoothing generally attempts to remove small scale details. Another commonly used term—*mesh filtering*—is also often used in place of mesh fairing, smoothing or denoising. Filtering, however, is a rather general term which simply refers to some black box which processes a signal to produce a new signal, and could, in principle, perform some quite different function such as feature enhancement.

In the early literature, mesh denoising was mostly done in the context of surface smoothing or fairing. Because the main objective of surface smoothing is the smoothness of the mesh surface, together with removal of the noise, prominent features in the resulting mesh surface may also be removed. In recent years, accompanied by increasing requirements on model fidelity, more and more attention has been given to the *feature-preserving* noise removal problem. Our work considers the feature-preserving problem, and in particular, the preservation of sharp edges.

There are two different approaches used for adjusting *vertex positions*: one is to directly adjust vertex positions in one step, and the other is to adjust vertex positions in two steps, by first computing appropriate new surface normals, and then computing the vertex positions from them. For a discussion of these two approaches, the reader is referred to Sun et al. (2007). This work adopts a two-step scheme. We first filter face normals iteratively, and then update vertex positions based on the filtered normals, also iteratively. Simply, our algorithm can be described as (Step 1)^{n_1} + (Step 2)^{n_2}, where Step 1 performs one pass of normal filtering, Step 2 performs one pass of vertex position updating, and n_1 and n_2 are numbers of iterations. We choose to separately update face normals and vertex positions because small normal errors, but not vertex position errors, may cause significant aliasing problems (Botsch and Kobbelt, 2001). In addition, it is also face normals, but not vertex positions, that greatly affect the appearance of rendered surfaces (Jones et al., 2004). We thus need to pay careful attention to the normal updating step. The novelty of our work is that the normal filtering is based on a *random walk* process.

Random walks are used as models in many areas of mathematics and physics. Use of random walk models in computer vision and image processing first appeared in Wechsler and Kidode (1979), where they were used for texture discrimination. More recently, random walks have been applied to other problems such as image enhancement (Smolka and Wojciechowski, 2001; Azzabou et al., 2006), image filtering (Smolka and Wojciechowski, 2001; Szczepanski et al., 2003), and image segmentation (Grady, 2006). Although image processing is closely linked to mesh processing, random walks on an image only have to deal with a 2D domain, which furthermore has a regular neighbourhood structure. It is not straightforward to extend such 2D methods to 3D meshes with irregular connectivity. The application of random walks to mesh processing appears to be new. In this paper, we consider a method of applying random walks to 3D mesh denoising, motivated by the random walk based image denoising approach proposed by Smolka and Wojciechowski (2001).

The rest of the paper is organised as follows. Section 2 reviews previous work on mesh denoising. Section 3 describes the notation we use, while Section 4 presents a simple introduction to random walk models. Section 5 is the core of the paper, discussing mesh normal filtering using random walks. We also present an adaptive parameter adjustment method, and analyse the feature-preserving properties of our approach. Section 6 explains how we perform vertex position updating using the conjugate gradient method. Section 7 presents experimental results and compares our method to other recent feature-preserving mesh denoising methods. Finally, Section 8 concludes the paper and gives some directions for future work.

2 Related work

In recent years, many mesh surface smoothing, fairing and denoising methods have been proposed. Many of the surface fairing and smoothing methods have also been used for noise removal, so we also consider them along with mesh denoising methods.

Most of the earlier surface fairing algorithms are based on minimisation of certain *energy functionals*. The most commonly used functionals are membrane energy, thin plate energy (Kobbelt et al., 1998) and total curvature (Welch and

Witkin, 1994). In general, minimising these continuous functionals is computationally complicated. However, after discretisation and parameterisation, the minimisation of the continuous functionals can be reduced to a linear or non-linear system with respect to vertex positions (Shen et al., 2005). For example, the variational derivatives of the membrane energy and thin plate energy lead to the Laplacian operator and the second Laplacian operator (Kobbelt et al., 1998; Desbrun et al., 1999), which with special discretisation and parametrisation turn out to be a simple umbrella operator (the *discrete Laplacian operator*) and a second-order umbrella operator, respectively (Kobbelt et al., 1998).

Classical *Laplacian smoothing* or fairing (Field, 1988) is the fastest and simplest surface smoothing method. However, when applied to a noisy 3D surface, significant shape distortion and surface shrinkage can result in addition to noise removal. To overcome these problems, Vollmer et al. (1999) proposed various improvements to classical Laplacian smoothing, but they still did not solve these problems completely. Starting from the viewpoint of signal processing, Taubin (1995) improved upon Laplacian fairing and proposed a second-order moving averaging filter to overcome shrinkage, but his approach still suffers from distortion of prominent mesh features. Moreover, if the two parameters of the filter are not chosen suitably, the algorithm can be numerically unstable. Desbrun et al. (1999) proposed a first-order autoregressive filter to tackle numerical stability. They overcome the problem of shrinkage by rescaling the mesh to preserve its volume. They also discussed diffusion and introduced curvature flow into surface fairing. Again, however, distortion of prominent mesh features occurs in their algorithm. Kim and Rossignac (2005) developed a general autoregressive moving average filter approach. Through suitable choice of parameters, the filter can act as a low-pass, band-pass, high-pass, notch, or band amplification or attenuation filter. Thus it can filter out, for example, high-frequency noise, and at the same time, enhance or suppress certain features. However, it is difficult to design a suitable filter that does both well.

All of the above methods were developed in the context of fairing and hence paid little or no attention to feature preservation. In fact, the above 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, especially edges. Recently various anisotropic filtering schemes have been proposed which smooth surfaces while simultaneously preserving prominent features. Sun et al. (2007) divided feature-preserving mesh denoising methods into four classes: one class is based on anisotropic geometric diffusion (Clarenz et al., 2000; Desbrun et al., 2000; Tasdizen et al., 2002; Bajaj and Xu, 2003; Hildebrandt and Polthier, 2004); the second class is based on bilateral filters and their variants (Jones et al., 2003; Fleishman et al., 2003; Choudhury and Tumblin, 2003; Yu et al., 2004; Lee and Wang, 2005; Shimizu et al., 2005; Li et al., 2005); among them Yu et al. (2004), Lee and Wang (2005) and Li et al. (2005) can also be assigned to a third class, which is based on combining normal filtering and vertex position updating (Ohtake et al., 2001; Taubin, 2001; Yagou et al., 2002, 2003; Shen and Barner, 2004; Chen and Cheng, 2005; Sun et al., 2007). The last class consists of remaining methods that do not simply fit into the above three classes (Shen et al., 2005; Nehab et al., 2005; Diebel et al., 2006; Yoshizawa et al., 2006; Schall et al., 2007). A fuller discussions of these methods is given in Sun et al. (2007).

The denoising approach proposed in this paper belongs to the third class. Compared to the aforementioned approaches, our approach either preserves features more efficiently or is computationally less expensive.

3 Notation

Here we briefly state the notation used in the rest of the paper.

We use $T = (V, E, F, X)$ to represent a triangle mesh, where $V = \{i : i = 1, \ldots, n\}$ are the *vertices*, $E = \{(i, j) : (i, j) \in V \times V\}$ are the *edges*, $F = \{(i, j, k) : (i, j), (i, k), (j, k) \in E\}$ are the *faces*, and $X = \{\mathbf{x}_i : \mathbf{x}_i \in \mathbb{R}^3, \ i \in V\}$ are the *vertex coordinates*. We use $|\cdot|$ to denote the *cardinality* of a set, so e.g. $|V|$ denotes the number of vertices. 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 *normal* to face F_i is denoted by $\mathbf{n}_i$. ∂F_i denotes the set of edges bounding 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 $N_V(i)$, comprises those 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 $N_{FI}(i)$, comprises those faces that have a *vertex or edge* in common with face F_i, and the second type, denoted by $N_{FII}(i)$, comprises those faces that share an *edge* with face F_i. Fig. 1 shows the two types of face neighbourhoods. Note that the $N_{FI}(i) \supset N_{FII}(i)$. We refer to the union of F_i and its neighbourhood by $N^*_{FI}(i) = N_{FI}(i) \bigcup \{F_i\}$ and $N^*_{FII}(i) = N_{FII}(i) \bigcup \{F_i\}$.

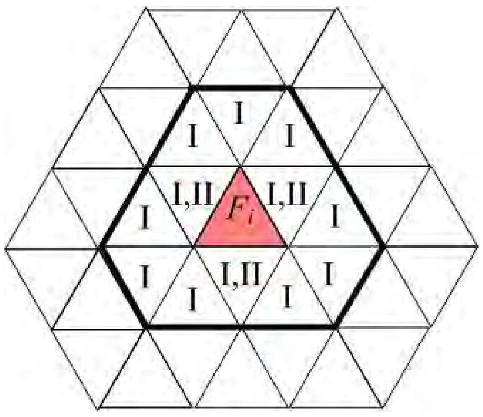


Fig. 1. Face neighbourhoods: faces labelled I belong to $N_{FI}(i)$; faces labelled II belong to $N_{FII}(i)$.

4 Markov chains and random walks

In this section, we introduce *random walks* from a practical point of view to help readers unfamiliar with this topic understand the rest of the paper. For a more mathematical treatment, the reader is referred to a standard textbook, e.g. Spitzer (2001). Because random walks are closely related to *Markov chains*, we begin with an explanation of Markov chains.

A *Markov chain* is a sequence of random variables $\{X_t : t = 0, 1, 2, \ldots\}$ with the property that, the future state depends only on the current state, and is conditionally independent of states before the present state, i.e.

$$\begin{aligned} &P(X_{t+1} = x_{t+1}|X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_0 = x_0) \\ &= P(X_{t+1} = x_{t+1}|X_t = x_t). \end{aligned} \tag{1}$$

The possible values X_t may take form a countable set called the *state space*; this may be finite or infinite. Here we simply use the index set $\mathcal{I} = \{1, \ldots, N\}$ to represent the state space, as we only consider finite state spaces.

In general, $P(X_{t+1} = x|X_t = y)$ need not equal $P(X_t = x|X_{t-1} = y)$. However, if $P(X_{t+1} = x|X_t = y) = P(X_t = x|X_{t-1} = y)$ for all t, we have a *stationary Markov chain*, a stochastic process in which the transition probabilities do not depend on t.

The transition probability from state i to state j at the t^{th} time step is denoted by $p_{i,j}(t) = P(X_t = j|X_{t-1} = i)$. We may construct an $N \times N$ matrix $\Pi(t)$, the *transition probability matrix*, whose $(i, j)^{th}$ entry is $p_{i,j}(t)$, where $i, j \in \mathcal{I}$. Thus, $\Pi(t)$ is a stochastic matrix in which each row sums to 1. We denote the probability that the Markov chain reaches the state i at time step t by $p_i(t) = P(X_t = i)$. We can now use a vector $P(t) = [p_1(t), \ldots, p_N(t)]$ to represent the probability distribution of the Markov chain over all states at time t. Note that $\sum_{i\in\mathcal{I}} p_i(t) = 1$.

The transition probability matrices together with the initial probability distribution completely determine the Markov chain. Let the initial distribution be denoted by $P(0)$. Then the distribution of the Markov chain is $P(1) = P(0)\Pi(1)$ after one step, and is $P(t) = P(0)\Pi^t$ after t steps, where $\Pi^t = \Pi(1)\cdots\Pi(t)$. We call Π^t the *t-step transition probability matrix.* The $(i,j)^{th}$ element of Π^t is denoted by $p^t_{i,j}$, and is the probability of going from state i to j after t steps.

Finally, a *random walk* is a discrete stochastic process consisting of a sequence of steps, each in a random direction. A random walk is a restricted type of Markov chain, where, in general, a single step can only move to a small set of states in the state space: the neighbouring states of the current state. Thus, its transition matrix $\Pi(t)$ is sparse for small t. However, in the limit after many steps, a random walk can reach any state, and as t grows, Π^t becomes non-sparse.

5 Normal filtering

We now discuss how a random walk model can be used to denoise face normals. There are two basic concepts. Firstly, the probability of stepping from one triangle to another should be greater, the more similar their normals are. Secondly, after walking has finished, new normals are found by averaging surrounding normals; the final probabilities of reaching surrounding triangles are used as weights in this averaging process. We thus give greater weight to similar triangles (belonging to similar parts of the surface) and less weight to ones that are, for example, on the other side of an edge feature. Related ideas have been used in Smolka and Wojciechowski (2001) for image denoising.

5.1 Random walk for normal filtering

We now give further details. We assume that initially a single virtual particle is placed on each face of the mesh, and this particle remembers the normal of its original face. At each step, the virtual particle may move to a neighbour of its current face, or stay in its current position; the probabilities depend on the face normals. After t steps of such a random walk, the particles will have been redistributed on the mesh surface according to the t-step transition probability matrix Π^t, which we then use to compute the new face normals in our normal filtering algorithm. We perform normal updating as weighted averaging of the normals of all faces on the mesh,

$$\mathbf{n}'_i = \frac{\sum_{j\in F} p^t_{i,j}\mathbf{n}_j}{\left|\sum_{j\in F} p^t_{i,j}\mathbf{n}_j\right|}. \tag{2}$$

Note that weighted averaging is a frequently used approach to updating normals (Taubin, 2001; Ohtake et al., 2001, 2002; Yagou et al., 2002, 2003; Shen and Barner, 2004; Sun et al., 2007).

To implement Equation (2), we need to know how to compute $p^t_{i,j}$. This depends on two elements: the choice of t, the number of steps, and $p_{i,j}(t)$, the transition probabilities. We first discuss the single-step transition probability.

Intuitively, the larger the difference between the normals of two neighbouring faces, the less similar they are, and hence the less appropriate it is that they should be included in the same average. Thus, the larger the difference, the smaller the probability should be of the virtual particle on one face visiting the other face. Hence, the single-step transition probability $p_{i,j}(t)$ should be a decreasing function of the normal difference $\|\mathbf{n}_i - \mathbf{n}_j\|$. Moreover, it is reasonable to require that this function should also be convex in $[0,\infty)$, and should tend to zero as its argument goes towards infinity (of course, the normal difference is in the range $[0,2]$, rather than $[0,\infty)$, but this makes little difference). Numerous functions satisfy the above conditions. Typical functions found in the literature (Szczepanski et al., 2003) are

$$f_1(x) = Ce^{-\beta x^2}, \tag{3}$$
$$f_2(x) = Ce^{-\beta x}, \tag{4}$$
$$f_3(x) = C\frac{1}{1+\beta x}, \tag{5}$$
$$f_4(x) = C\frac{1}{(1+x)^\beta}, \tag{6}$$
$$f_5(x) = C\left(1-\frac{2}{\pi}\arctan(\beta x)\right), \tag{7}$$
$$f_6(x) = C\frac{2}{(1+e^{\beta x})}, \tag{8}$$
$$f_7(x) = C\frac{1}{1+x^\beta}, \tag{9}$$
$$f_8(x) = \begin{cases} C(1-\beta x) & \text{if } x < 1/\beta, \\ 0 & \text{if } x \geq 1/\beta, \end{cases} \tag{10}$$

where $\beta \in (0,\infty)$ is a parameter altering in detail how the function varies with respect to its argument, and C is a normalisation constant which ensures that probabilities sum to one.

Clearly, many functions satisfy our basic requirements. *A priori* knowledge and experiments help us to choose a suitable function. In our experiments, we have found that functions f_3, f_5 and f_7 do not preserve sharp edges effectively whilst denoising meshes, but all the other functions perform well, as we show later in the paper. Ultimately, we chose to use the Gaussian function $f_1(x)$ since a Gaussian noise distribution occurs commonly in the real world, and our experiments show that it yields very good results (see Section 7.1.1). A further benefit is that the Gaussian function can be easily used in conjunction with the adaptive parameter adjustment procedure described in the next section.

We note that Ohtake et al. (2001, 2002) also use a Gaussian function as a weighting function in their weighted averaging of normals. The difference between our approach and that of Ohtake et al. (2001, 2002) is that we have chosen different arguments to the weighting function, and that our argument—the normal difference—is simpler than their variable—the directional curvature.

Because

$$\|\mathbf{n}_i - \mathbf{n}_j\|^2 = 2(1 - \mathbf{n}_i \cdot \mathbf{n}_j), \tag{11}$$

we may write (after combining constant factors into the coefficient C and the parameter β of the Gaussian function $f_1(x)$)

$$p_{i,j}(t) = \begin{cases} Ce^{\beta \mathbf{n}_i \cdot \mathbf{n}_j} & \text{if } j \in N_F(i), \\ 0 & \text{otherwise,} \end{cases} \tag{12}$$

where the normalisation coefficient C is given by

$$C = 1 / \sum_{k \in N_F(i)} e^{\beta \mathbf{n}_i \cdot \mathbf{n}_k}. \tag{13}$$

Here, $N_F(i)$ is the 1-ring face neighbourhood of the face F_i, which we may take to be either $N_{FI}(i)$ or $N_{FII}(i)$.

Next, we consider how to choose the number of steps t for the walk. As discussed in Section 4, as t becomes larger, more non-zero elements appear in the matrix Π^t, which means that more face normals are used in the computation of the new normal in Equation (2), and better results can be expected. However, because the number of non-zero elements increases, the computational cost of Equation (2) also becomes larger. We must seek a tradeoff between computational cost and quality of results.

If we adopt a non-iterative scheme to update face normals, t must be large enough to obtain a result satisfying the qualitative requirements of denoising. However, if we adopt an iterative scheme, then using small t can also produce good results if we use several iterations of normal updating. Because a non-iterative scheme only needs to update face normals once, it might appear that a non-iterative scheme would be computationally more efficient than an iterative scheme. However, just as was found in a comparison of computational cost between two bilateral filtering schemes (Fleishman et al., 2003; Jones et al., 2003), the non-iterative scheme is in practice more time-consuming. Thus, we adopt an iterative scheme to update face normals.

To investigate the effect of t on the final quality and computational cost, we first consider the face normal updating formulae for different t. In the simplest case $t = 1$, we get $p_{i,j}^1 = p_{i,j}(1)$. Thus, the face normal updating formula in Equation (2) becomes

$$\mathbf{n}'_i = \frac{\sum_{j \in N_F^*(i)} e^{\beta \mathbf{n}_i \cdot \mathbf{n}_j} \mathbf{n}_j}{\left| \sum_{j \in N_F^*(i)} e^{\beta \mathbf{n}_i \cdot \mathbf{n}_j} \mathbf{n}_j \right|}, \tag{14}$$

or,

$$\mathbf{n}'_i = \frac{\sum_{j \in N_F(i)} e^{\beta \mathbf{n}_i \cdot \mathbf{n}_j} \mathbf{n}_j}{\left| \sum_{j \in N_F(i)} e^{\beta \mathbf{n}_i \cdot \mathbf{n}_j} \mathbf{n}_j \right|}, \tag{15}$$

where $N_F^*(i)$ is the union of F_i and $N_F(i)$. These alternatives correspond to the cases in which the virtual particle on the current face F_i is or is not allowed to stay on F_i in the next step, respectively. The normalisation coefficient C does not need to explicitly appear in the above formulae because the last step of computing $\mathbf{n}'_i$ is to normalise it.

Note that in each iteration, $\mathbf{n}'_i$ is computed sequentially from $i = 1$ to $i = |F|$. Thus when we compute $\mathbf{n}'_i$ using Equation (14) or (15), some right-hand-side normals $\mathbf{n}_j$ may have a new value $\mathbf{n}'_j$ available. We can either use their old values $\mathbf{n}_j$ obtained in the last iteration, or the new values $\mathbf{n}'_j$ already obtained in this iteration when computing $\mathbf{n}'_i$. We call the former scheme the *batch* scheme and the latter the *progressive* scheme. The progressive scheme is expected to more quickly give a result of the same quality than the batch scheme because $\mathbf{n}'_j$ used in the progressive scheme is closer to the required denoised normal than $\mathbf{n}_j$ used in the batch scheme. Our experiments justify this conclusion.

Now consider the case $t > 1$. If we directly use Equation (2) to update normals, we need to compute Π^t. Because Π^t will become non-sparse as t grows, the computational cost will grow quickly, and additional memory will be required to store the whole matrix Π^t. To save memory and computation time, we suggest that normals should be updated sequentially:

$$\mathbf{n}_i(k) = \sum_{j \in N_F^*(i)} p_{i,j}(k)\mathbf{n}_j(k-1), \quad k = \{1, \ldots, t\}, \tag{16}$$

or, using the alternative neighbourhood,

$$\mathbf{n}_i(k) = \sum_{j \in N_F(i)} p_{i,j}(k)\mathbf{n}_j(k-1), \quad k = \{1, \ldots, t\}, \tag{17}$$

and finally,

$$\mathbf{n}'_i = \frac{\mathbf{n}_i(t)}{|\mathbf{n}_i(t)|}. \tag{18}$$

The algorithm given by Equation (16) or (17) updates normals $\{\mathbf{n}_i(k), i \in V\}$ starting from $k = 1$ with known $\{\mathbf{n}_j(0) : j \in V\}$ which take the values $\{\mathbf{n}_j\}$ of the last iteration (or the initial normals during the first iteration). This is repeated until the normal values for $k = t$ are computed. Then Equation (18) is used to give $\mathbf{n}'_i$. It can easily be shown that Equation (18) together with Equation (16) or (17) is equivalent to Equation (2). However, because for a given k, only a sparse matrix $\Pi(k)$ is required in the computation of Equations (16) or (17), the memory requirement and the computational cost are greatly reduced.

Note that the above implementation is a *batch* scheme for the case $t > 1$. If we adopt a *progressive* scheme, i.e. once we obtain a $\mathbf{n}_i(k)$, we immediately normalise it, take it as $\mathbf{n}_i(k-1)$, and use it in the computation of $p_{i,j}(k)$, then one iteration of the t-step algorithm in Equation (16) or (17) is equivalent to t iterations of the 1-step algorithm in Equation (14) or (15), respectively. Thus, when considering the progressive scheme, it makes no difference whether we talk about it as 1-step or t-step algorithm. While the progressive scheme has the advantage of generally converging more quickly than the batch scheme, due to its use of more up-to-date information, the need to normalise normals immediately in the case of $t > 1$ incurs an extra cost which counterbalances this advantage.

Another explanation of the progressive scheme can be given. We can consider it as random walks in which different virtual particles on the surface start walking at different times—the particle on face 1 begins walking first (which causes the first normal and corresponding probabilities to be changed), and then the one on face 2, and so on. After all the particles have taken one step, the first particle begins its second step, then the second particle, and so on.

This process of updating normals and probabilities continues until all particles have finished their t-step walks. We can consider this procedure as t iterations of 1-step random walks. We can also consider it as one iteration of t-step random walks, but with different probabilities for different steps.

5.2 Adaptive parameter adjustment

In the computation of $\mathbf{n}'_i$, the only parameter involved is β. Choosing a suitable parameter value affects the quality of the result. Since we have chosen a Gaussian function for the probability distribution function, and β is inversely proportional to the variance, we can immediately see *qualitatively* how to choose β: if the model noise is high, β should be small, and vice versa. However, if preservation of surface features such as sharp edges and corners is important, we should make β large so that neighbouring normals which deviate far from the current normal $\mathbf{n}_i$ make a very small contribution to the computation of $\mathbf{n}'_i$. Further investigation is needed to *quantitatively* determine β. One method of doing so is through experiments: our experiments show that $\beta \in [8, 12]$ generally works well for most models we have tested.

If an iterative approach is used to denoising, normal noise should reduce after each iteration. The qualitative analysis suggests that we should dynamically adjust β so that it becomes larger after each iteration. Smolka and Wojciechowski (2001) suggest in their application that β should be adjusted using $\beta' = \delta\beta$, where $\delta > 1$; see also (Szczepanski et al., 2003). We have performed experiments using such a parameter adjustment scheme, but found that it only provides a minor improvement.

Instead, we introduce an alternative adaptive method for parameter adjustment, using a similar idea to one introduced by Shen and Barner (2004). We adjust β to give us minimum overall change in normals between the original, noisy, model, and the final denoised model, the goal being to preserve as much available information as possible about the features of the model. Let $\mathbf{n}_{i0}$ be the initial noisy normal of face F_i, and let $\mathbf{n}'_i(\beta)$ be the updated normal using a parameter value of β. Then, we wish to find the value of β which minimises the cost function $J(\beta) = E(|\mathbf{n}_{i0} - \mathbf{n}'_i(\beta)|)$, where E is the expectation operator. This is equivalent to minimising

$$J(\beta) = E\left(-\mathbf{n}_{i0} \cdot \mathbf{n}'_i(\beta)\right). \tag{19}$$

We use a stochastic gradient-based algorithm to solve the problem of minimising $J(\beta)$. The parameter is updated using

$$\beta' = \beta - \mu \frac{\partial J}{\partial \beta}, \tag{20}$$

where μ is a damping factor, and J is the current value of $|\mathbf{n}_{i0} - \mathbf{n}'_i(\beta)|$. In the following derivation, we use Equation (14) as a specific example definition for $\mathbf{n}'_i(\beta)$; using Equation (15) gives similar results. When Equations (16)–(18) are used to update normals, it is not as easy to derive similar results. However, fortunately, when we adopt the progressive scheme, this is not a problem because, as we have pointed out, one iteration of the t-step scheme in Equation (16) or (17) is equivalent to t iterations of the 1-step scheme in Equation (14) or (15), respectively. Let us now define

$$\tilde{\mathbf{n}}_i = \sum_{j \in N_F^*(i)} e^{\beta \mathbf{n}_i \cdot \mathbf{n}_j} \mathbf{n}_j. \tag{21}$$

The derivative of $\tilde{\mathbf{n}}_i$ with respect to β is given by

$$\dot{\tilde{\mathbf{n}}}_i = \frac{\partial \tilde{\mathbf{n}}_i}{\partial \beta} = \sum_{j \in N_F^*(i)} (\mathbf{n}_i \cdot \mathbf{n}_j) e^{\beta \mathbf{n}_i \cdot \mathbf{n}_j} \mathbf{n}_j. \tag{22}$$

As $\mathbf{n}'_i(\beta) = \tilde{\mathbf{n}}_i / \|\tilde{\mathbf{n}}_i\|$, the gradient can be written as

$$\frac{\partial J}{\partial \beta} = \frac{1}{\|\tilde{\mathbf{n}}_i\|^2} \left(\mathbf{n}_{i0} \cdot \tilde{\mathbf{n}}_i \frac{\partial \|\tilde{\mathbf{n}}_i\|}{\partial \beta} - \mathbf{n}_{i0} \cdot \dot{\tilde{\mathbf{n}}}_i \|\tilde{\mathbf{n}}_i\| \right). \tag{23}$$

By further noting that $\|\tilde{\mathbf{n}}_i\| \partial \|\tilde{\mathbf{n}}_i\| / \partial \beta = \tilde{\mathbf{n}}_i \cdot \dot{\tilde{\mathbf{n}}}_i$, Equation (23) becomes

$$\frac{\partial J}{\partial \beta} = \frac{1}{\|\tilde{\mathbf{n}}_i\|^3} \left((\mathbf{n}_{i0} \cdot \tilde{\mathbf{n}}_i)(\tilde{\mathbf{n}}_i \cdot \dot{\tilde{\mathbf{n}}}_i) - (\mathbf{n}_{i0} \cdot \dot{\tilde{\mathbf{n}}}_i) \|\tilde{\mathbf{n}}_i\|^2 \right). \tag{24}$$

Combining (20) and (24), we obtain the parameter updating formula

$$\beta' = \beta - \frac{\mu}{\|\tilde{\mathbf{n}}_i\|^3} \left((\mathbf{n}_{i0} \cdot \tilde{\mathbf{n}}_i)(\tilde{\mathbf{n}}_i \cdot \dot{\tilde{\mathbf{n}}}_i) - (\mathbf{n}_{i0} \cdot \dot{\tilde{\mathbf{n}}}_i) \|\tilde{\mathbf{n}}_i\|^2 \right). \tag{25}$$

This gives a method of updating β which is only based on one face of the mesh. Because the optimal update depends on all faces, we keep the parameter unchanged during each iteration, and update the parameter only after a whole iteration step is finished. The magnitude of the parameter update from one iteration to the next is the accumulated update magnitude for all faces (while an average might be more intuitively correct, we allow for this by scaling μ appropriately, as described shortly), i.e.,

$$\beta' = \beta - \sum_{i \in F} \frac{\mu}{\|\tilde{\mathbf{n}}_i\|^3} \left((\mathbf{n}_{i0} \cdot \tilde{\mathbf{n}}_i)(\tilde{\mathbf{n}}_i \cdot \dot{\tilde{\mathbf{n}}}_i) - (\mathbf{n}_{i0} \cdot \dot{\tilde{\mathbf{n}}}_i) \|\tilde{\mathbf{n}}_i\|^2 \right). \tag{26}$$

To implement the idea in Equation (26), μ and an initial value of β need to be given. We have chosen $\mu = 500/|F|$ in all of our experiments, which seems to work well. Our experiments show that β can vary over a large range with little difference in experimental results. As initial value for β we have used 8 in most of our experiments.

5.3 Feature-preserving property

Equations (14)–(18) show that the updated normal of each face is a weighted average of the normals of its neighbouring faces. Because the weight function is a decreasing function of the difference between the normal of the central face and that of a neighbouring face, the further these normals deviate, the less influence this neighbouring face has on the central face normal. Where part of a mesh lies on a feature, it is required that neighbouring faces have only a small influence on each other during updating. This is satisfied as normals of neighbouring faces usually substantially deviate from each other. Hence, our algorithm has an inbuilt feature-preserving property. In addition, because the parameter β is adaptively adjusted to minimise the cost of the difference between the initial normal and the updated normal, the feature-preserving property is further improved.

Various other anisotropic mesh filtering algorithms also compute weighted averages of neighbouring face normals. Mean filtering algorithms (Taubin, 2001; Yagou et al., 2002) treat all neighbouring faces the same, and so do not have a feature-preserving property. Median filtering algorithms (Yagou et al., 2002) use the median of neighbouring face normals for the updated normal. This can preserve features but cannot satisfactorily smooth the mesh surface. The alpha-trimming filtering algorithm (Yagou et al., 2003) is a simple compromise between mean and median filtering algorithms. However, it is not a good compromise because the feature-preserving property can easily be ruined. The fuzzy vector median filtering algorithm (Shen and Barner, 2004) can effectively preserve features and smooth the mesh. However, it is more time-consuming than our algorithm given here.

6 Vertex position updating

After adjusting the face normals, the vertex positions are updated based on the new normals. Several algorithms exist for this purpose (Taubin, 2001; Ohtake et al., 2001, 2002; Yu et al., 2004; Sun et al., 2007). Taubin (2001) uses orthogonality between the face normal and the face plane on the mesh to give a system of linear equations for vertex position updating. In general, this system of equations has no non-trivial solution, so he solves it in a least-squares sense using the gradient descent method (see also Shen and Barner (2004)). Unfortunately, this method converges slowly, and if the step size is not suitably chosen, it may be unstable. Ohtake et al. (2001) also gives a vertex updating algorithm similar to Taubin's with a particular step size and additional area weights. Again, as it a gradient descent based method, it converges slowly. Sun et al. (2007) further improves the vertex position updating algorithm of Ohtake et al. (2001) by removing its area weights; they also prove the convergence of the vertex updating algorithms of Ohtake et al. (2001) and Sun et al. (2007). Still, their algorithm also converges slowly. Ohtake et al. (2002) propose another vertex position updating algorithm based on the minimisation of the area-weighted sum of the squared differences between the original and the new face normals. The solution to this minimisation problem is also performed using gradient descent. However, since the gradient computation involved is more complicated, it is computationally more expensive than Taubin's algorithm. It also has the problem of choosing a suitable step size. The method of Yu et al. (2004) is an implicit method which updates vertex positions through gradient field manipulation. A gradient field is first computed using a local rotation matrix derived from $\mathbf{n}_i$ and $\mathbf{n}'_i$, which is then used in a Poisson equation to compute the updated vertex positions. Because the Poisson equation is linear, a linear system solver can be used. Compared to Taubin's method, Yu et al.'s method is computationally more complex, however, because of its extra requirement to compute the gradient field.

The least-squares problem for vertex position updating is linear, and its normal equations are given by a symmetric sparse matrix, so there are various efficient linear solvers available which could be used. Botsch et al. (2005) discuss various linear solvers and their respective advantages and disadvantages. Here, we adopt a simple and yet relatively efficient approach, the conjugate gradient method. We start with the face orthogonality conditions which yield the following family of simultaneous linear equations (Taubin, 2001):

$$\begin{cases} \mathbf{n}'_k \cdot (\mathbf{x}_{k_1} - \mathbf{x}_{k_2}) = 0 \\ \mathbf{n}'_k \cdot (\mathbf{x}_{k_2} - \mathbf{x}_{k_3}) = 0 \\ \mathbf{n}'_k \cdot (\mathbf{x}_{k_3} - \mathbf{x}_{k_1}) = 0 \end{cases}, \quad \forall k \in F, \tag{27}$$

where k_1, k_2, and k_3 are the vertices of face k. The least-squares cost function corresponding to the above system is

$$e(X) = \sum_{k \in F} \sum_{(i,j) \in \partial F_k} \left(\mathbf{n}'_k \cdot (\mathbf{x}_i - \mathbf{x}_j)\right)^2. \tag{28}$$

We could generalise the right hand side of Equation (28) to add weights related to the triangle areas, edge lengths, or shapes. Suitably chosen weight functions might produce better quality meshes according to particular criteria. However, introducing weight functions also requires additional computational effort. Because many meshes in practice have fairly uniform triangle sizes, we only consider Equation (28) itself in this paper. Indeed, our experiments show that we can obtain satisfactory results by simply using Equation (28) even for nonuniform meshes.

General formulae for solving least-squares problems using the conjugate gradient method can be found in many standard textbooks, e.g. (Press et al., 1992). However, as the problem here is reduced to solving a sparse system, we give detailed formulae, allowing our paper to be self-contained.

We introduce vectors $\{\mathbf{g}_i \in \mathbb{R}^3,\ i \in V\}$, $\{\mathbf{p}_i \in \mathbb{R}^3,\ i \in V\}$, and $\{\mathbf{q}_k \in \mathbb{R}^3,\ k \in F\}$, and separately concatenate $\{\mathbf{g}_i\}$, $\{\mathbf{p}_i\}$, and $\{\mathbf{q}_k\}$, respectively, to form three long vectors $G \in \mathbb{R}^{3|V|}$, $P \in \mathbb{R}^{3|V|}$, and $Q \in \mathbb{R}^{3|F|}$. The initial values of $\mathbf{g}_i$ and $\mathbf{p}_i$ are computed by

$$\mathbf{g}_i = \mathbf{p}_i = 3 \sum_{k \in F_V(i)} \mathbf{n}'_k(\mathbf{n}'_k \cdot (\bar{\mathbf{x}}_k - \mathbf{x}_i)), \tag{29}$$

where $\bar{\mathbf{x}}_k = \frac{1}{3}\sum_{j=1}^{3} \mathbf{x}_{k_j}$ is the mid-point of face k. The conjugate gradient method then updates the vertex positions $\mathbf{x}_i$ together with $\mathbf{g}_i$, $\mathbf{p}_i$, and $\mathbf{q}_k$ in the following way:

$$\mathbf{q}_k = \begin{bmatrix} \mathbf{n}_k \cdot (\mathbf{p}_{k_1} - \mathbf{p}_{k_2}) \\ \mathbf{n}_k \cdot (\mathbf{p}_{k_2} - \mathbf{p}_{k_3}) \\ \mathbf{n}_k \cdot (\mathbf{p}_{k_3} - \mathbf{p}_{k_1}) \end{bmatrix}, \quad \forall k \in F, \tag{30}$$

$$\alpha = \|G\|^2 / \|Q\|^2, \tag{31}$$

$$\mathbf{x}'_i = \mathbf{x}_i + \alpha \mathbf{p}_i,\ \forall i \in V, \tag{32}$$

$$\mathbf{g}'_i = \mathbf{g}_i + 3\alpha \sum_{k \in F_V(i)} \mathbf{n}'_k(\mathbf{n}'_k \cdot (\bar{\mathbf{p}}_k - \mathbf{p}_i)), \quad \forall i \in V, \tag{33}$$

$$\gamma = \|G'\|^2 / \|G\|^2, \tag{34}$$

$$\mathbf{p}'_i = \mathbf{g}'_i + \gamma \mathbf{p}_i,\ \forall i \in V, \tag{35}$$

where $\bar{\mathbf{p}}_k = \frac{1}{3}\sum_{j=1}^{3} \mathbf{p}_{k_j}$, and G' is formed by concatenating $\{\mathbf{g}'_i\}$.

The conjugate gradient algorithm in Equations (29)–(35) is iterated until it reaches a given maximum number of iterations, n_2, or meets a given tolerance

ϵ such that $\|X' - X\| < \epsilon$, or $\|G\| < \epsilon$. In all of our experiments, we have set a fixed maximum $n_2 = 50$ and the tolerance $\|G\|^2 < 10^{-6}|V|$.

7 Results and discussion

This Section presents results of tests carried out on our random walk filtering (RF) approach, which we also compare to several other approaches: median filtering (MF, Yagou et al., 2002), bilateral filtering (BF, Fleishman et al., 2003), and fuzzy vector median filtering (FF, Shen and Barner, 2004). The algorithms were implemented in VC++.net, and our experiments were performed on a PC with a 3.2GHz Intel Xeon CPU with 2GB of RAM. Both synthetic and scanned models were used.

7.1 Experiments on random walk filtering

In Section 5 we presented several alternative schemes of implementing the random walk-based normal filtering algorithm. Here, we compare these schemes experimentally to determine which is best. In some cases, it is easy to distinguish the quality of different schemes by means of a visual comparison. However, when only small visual differences are apparent, we need a numerical criterion to distinguish them. The criterion used here is the mean square angular error (MSAE) between the ideal and the denoised normals. This criterion was also used in Nehorai and Hawkes (2000) and Shen and Barner (2004). It is defined as

$$\mathrm{MSAE} = E\left(\angle\left(\mathbf{n}_d, \mathbf{n}\right)\right), \tag{36}$$

where $\angle(\mathbf{n}_d, \mathbf{n})$ is the angle between the denoised normal $\mathbf{n}_d$ and the original normal $\mathbf{n}$: we compare the normals produced by each scheme with those of the original synthetic model (before addition of noise). To implement the expectation operator, we take a simple average over all face normals in the mesh. We conducted experiments with various meshes; all the experiments result in the same conclusions.

7.1.1 Choice of transition probability function

This section presents experimental results concerning the use of different transition probability functions in our approach. The eight functions used in our experiments were f_1–f_8 given in Equations (3)-(10).

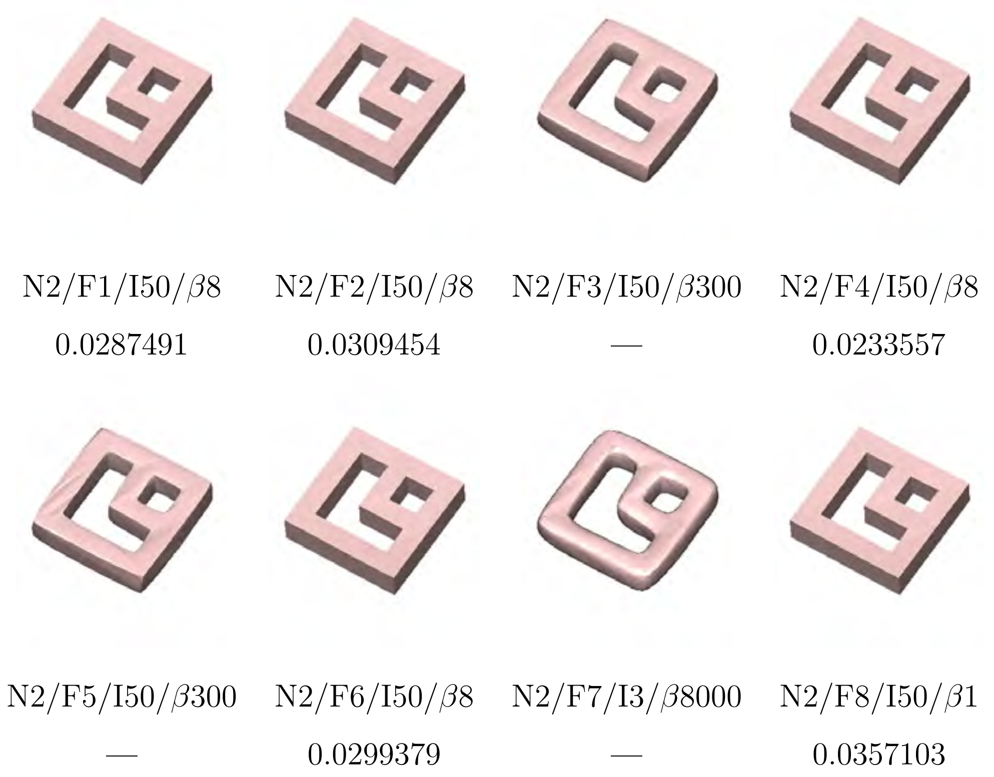


Fig. 2. Experimental results for a double-torus model for different transition probability functions, original model courtesy of Y. Ohtake.

Experimental results are shown in Figs. 2–5, where the notation Nx/Fy/Iz/βw below each figure means that the model noise is $x/10$ times the mean edge length of the mesh, the function used is function number y, the number of iterations of normal updating is z, and the parameter β has the value w. The lower number beneath each figure is the value of MSAE. Note that in some cases we have not provided an MSAE value (denoted by '—') since a visual comparison is sufficient by itself to determine that they are unsatisfactory results.

The results presented show that functions $f_3(x)$, $f_5(x)$ and $f_7(x)$ do not effectively preserve features for models with sharp edges, but can effectively denoise relatively smooth models, such as the bunny model. Function $f_4(x)$ generally results in the smallest MSAE for models with sharp edges, except for the cylinder model. The other functions are also able to preserve sharp edges. Generally, functions $f_2(x)$ and $f_6(x)$ produce better results than $f_1(x)$ and $f_8(x)$.

Table 1 shows the time taken for 50 iterations of normal updating for each function, using several models. From the table we can see that using $f_1(x)$ is fastest while using $f_4(x)$ is slowest. Thus if we want the fastest denoising process, we can use $f_1(x)$, while if we want to obtain the best qualitative results

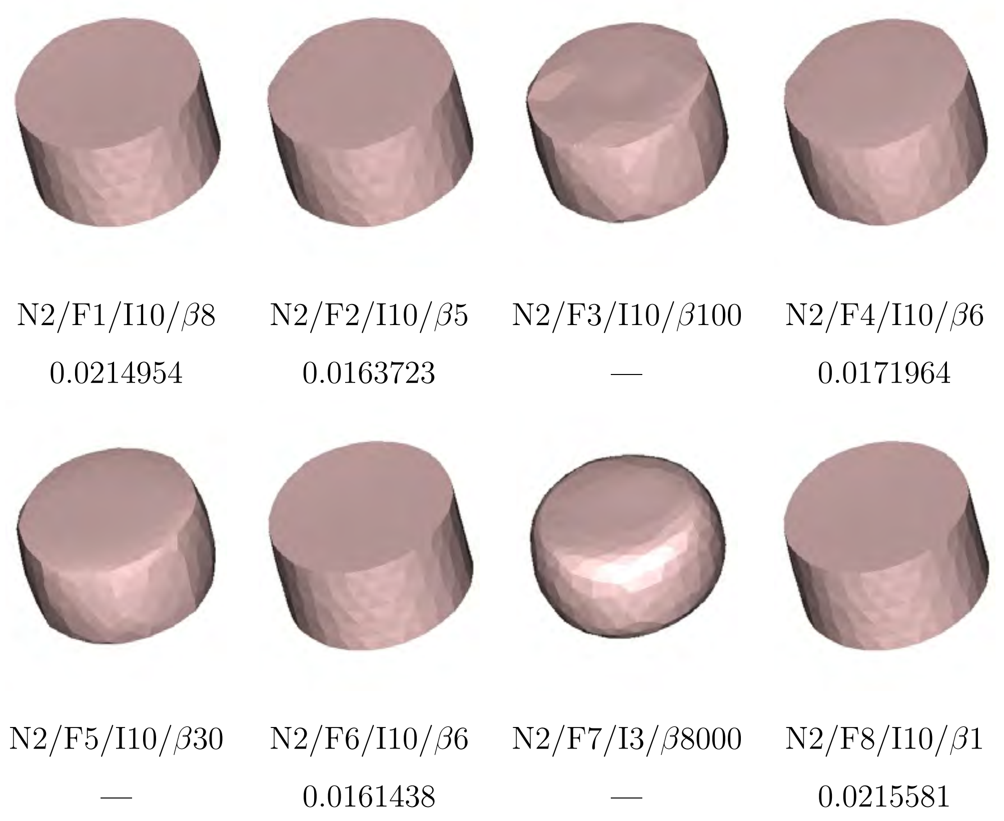


Fig. 3. Experimental results for a cylinder model for different transition probability functions.

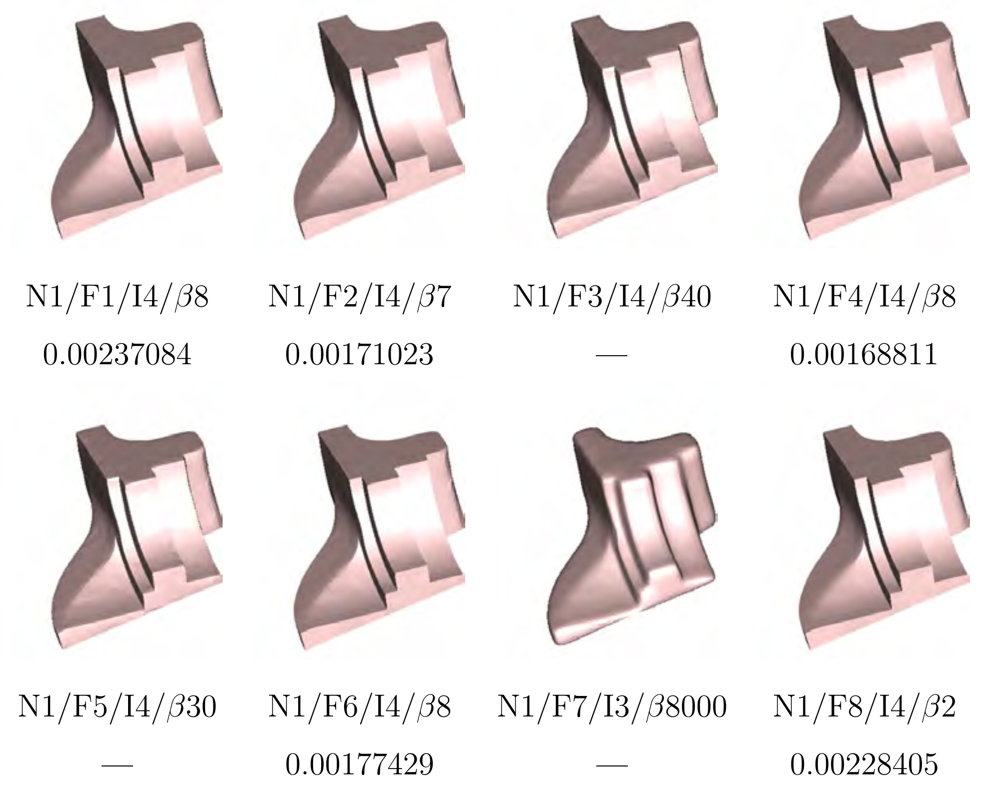
for models with sharp edges, we should use $f_4(x)$. Function $f_2(x)$ provides a compromise between good quality and low computational cost. Note that the difference in computation time between the slowest and fastest approach is little more than a factor of two, however. We have used the Gaussian function $f_1(x)$ elsewhere in the paper when discussing further details because it is the fastest function of the functions we have considered, and we note that Gaussian noise is the most common noise distribution.

Table 1 also shows that the time taken for normal updating varies approximately linearly with the number of faces no matter which function is used in our approach, as expected.

7.1.2 Experiments on variants using a given transition probability function

Next, we discuss experimental results on variants when using a given transition probability function. The transition probability function used hereafter is the Gaussian function $f_1(x)$.

Firstly, we discuss the effect of varying the number of random walk steps t. Fig. 6 shows the optimal MSAEs achieved, the corresponding total computa-



N1/F1/I4/β8	N1/F2/I4/β7	N1/F3/I4/β40	N1/F4/I4/β8
0.00237084	0.00171023	—	0.00168811
N1/F5/I4/β30	N1/F6/I4/β8	N1/F7/I3/β8000	N1/F8/I4/β2
—	0.00177429	—	0.00228405

Fig. 4. Experimental results for a fandisk model for different transition probability functions, original model courtesy of H. Hoppe.

tion times, and the numbers of iterations needed for convergence for several models illustrated elsewhere in the paper. It can be seen that lower t generally produces slightly smaller optimal MSAE, but there are no clear trends in overall computation time. Because the variations in these optimal MSAEs and computation times are not significant, no clear preference exists for the choice of t. We suggest choosing $t = 1$ for simplicity. It can also be seen from Fig. 6 that the optimal number of iterations reduces gradually as t increases, and in the limit, only one iteration is required, which corresponds to a non-iterative scheme. This corresponds with the qualitative analysis in Section 5.1.

Secondly, we consider the relationship between the MSAE and the number of iterations n_1. Fig. 7 shows the variation of MSAE with n_1 for several models. It can be seen that for the double-torus model, which has sharp edges and flat surfaces, as n_1 grows, the MSAE first decreases and then, after reaching a minimum, increases slightly. For other models having similar sharp features to the double-torus model, we can safely choose large n_1 to obtain good quality results. For the bunny model, which has a curved, textured surface, as n_1 grows, MSAE first decreases and then increases notably. Thus for models with similar features to the bunny model, careful choice of n_1 is necessary to get good quality results. Fig. 7 also shows that MSAE reaches a minimum faster for large t. Specifically, when $t = 10$, only one iteration leads to the minimum

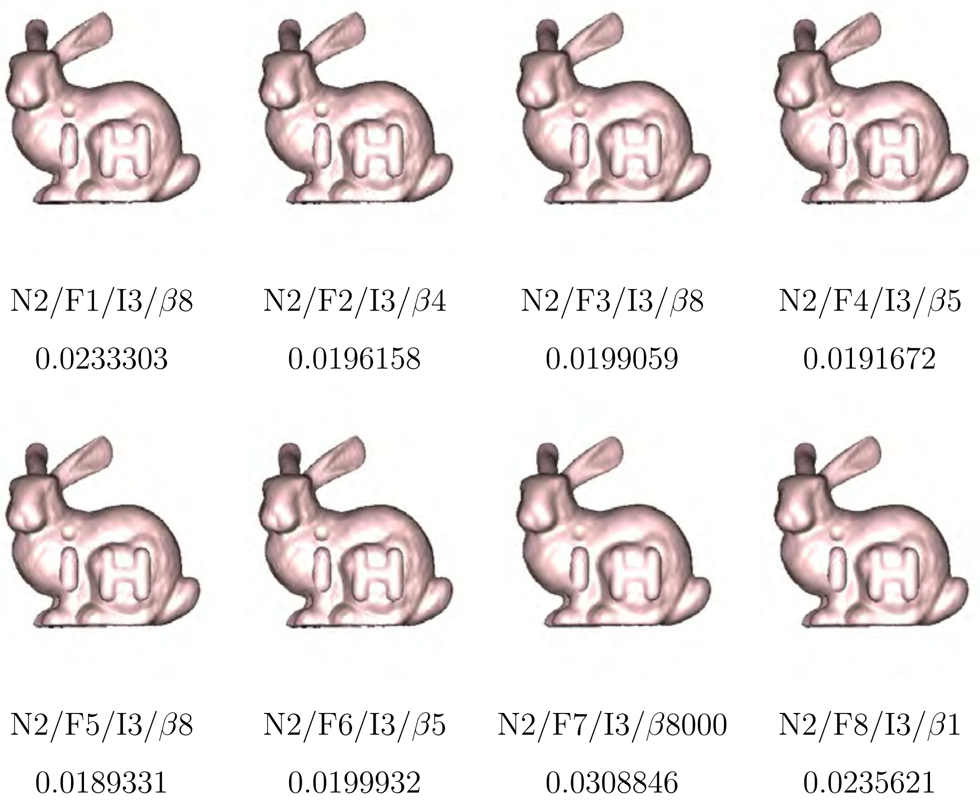


Fig. 5. Experimental results for an "iH" embossed Standard bunny model for different transition probability functions, original model courtesy of Y. Ohtake.

MSAE for the bunny model.

In the rest of this section, we only discuss the case $t = 1$ since, as we have shown above, the results obtained for $t \geq 2$ do not differ significantly from those for $t = 1$; we only analyse results from the double-torus model for reasons of space.

Thirdly, we discuss the effect of adaptively adjusting the parameter β. Experiments show that when $\beta \in [6, 12]$, both adaptive and non-adaptive schemes yield good results (for adaptive schemes the β here is the initial value). However, if β is chosen smaller, the non-adaptive scheme causes sharp edges to become rounded, while the adaptive scheme can still yield good results if $\beta > 3$. Fig. 8 shows the results obtained after n_1 successive iterations using adaptive and non-adaptive schemes when $\beta = 8$, and $\beta = 5$. It can be seen that when $\beta = 8$, both schemes produce almost perfect results. However, when $\beta = 5$, the non-adaptive scheme distorts the mesh, while the adaptive scheme still yields very good result. This experiment shows that the adaptive scheme can robustly adjust the parameter β.

Fourthly, we discuss whether it is preferable to include or exclude the central

Table 1
Normal updating times for different transition probability functions (in seconds, for 50 iterations)

	Fandisk	"iH" Bunny	Igea
$\lvert V\rvert$	6,475	34,834	134,345
$\lvert F\rvert$	12,946	69,451	268,686
f_1	0.75	4.156	15.813
f_2	1.031	5.656	21.704
f_3	0.75	4.172	15.828
f_4	1.766	9.39	36.578
f_5	1.516	8.094	30.14
f_6	1.141	6.219	24.14
f_7	1.641	8.953	34.782
f_8	0.828	4.5	17.109

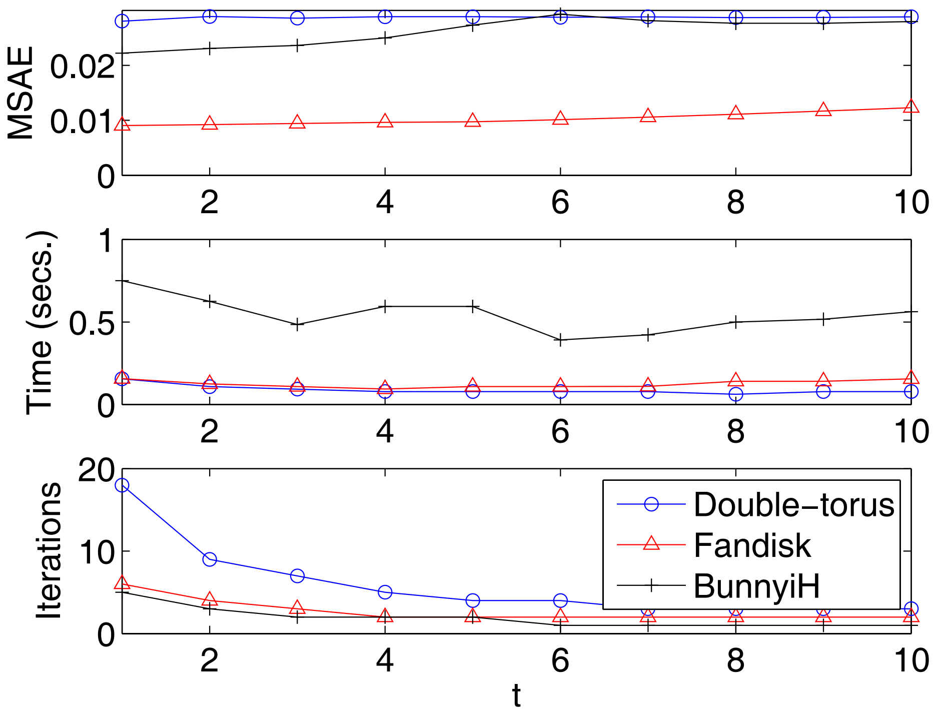


Fig. 6. Effect of varying numbers of random walk steps $t = 1, \dots, 10$. Top: optimal mean square angular errors achieved. Centre: computational times. Bottom: number of iterations needed for optimal errors.

face normal in the computation. Fig. 9 shows variation in MSAE with n_1 for the double-torus model when using the alternative methods with and without the central face. It can be seen that using the central face yields larger MSAE

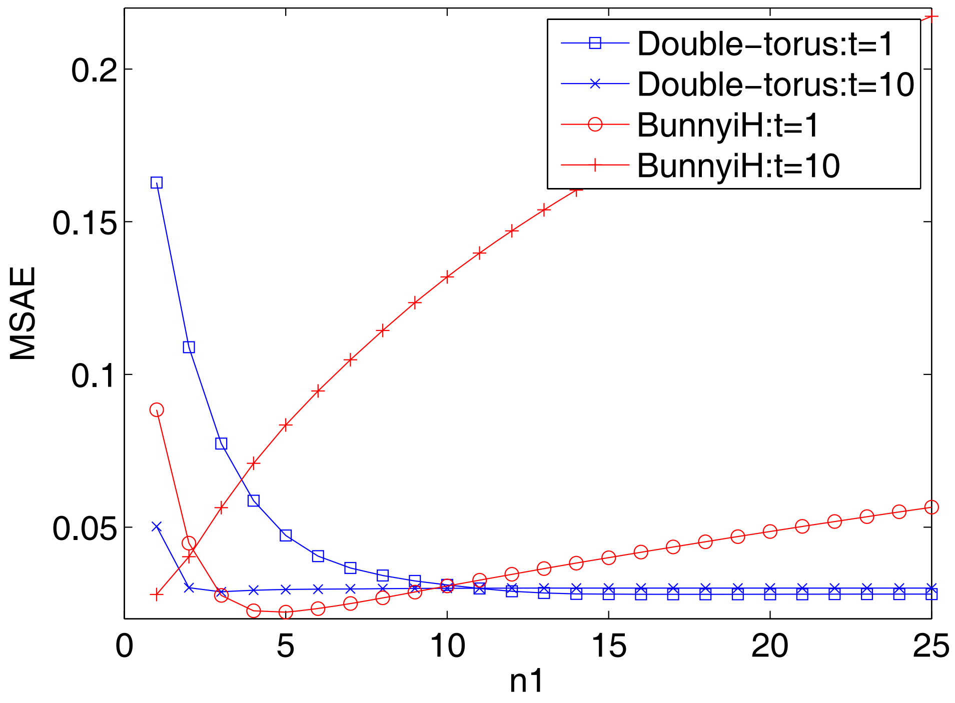


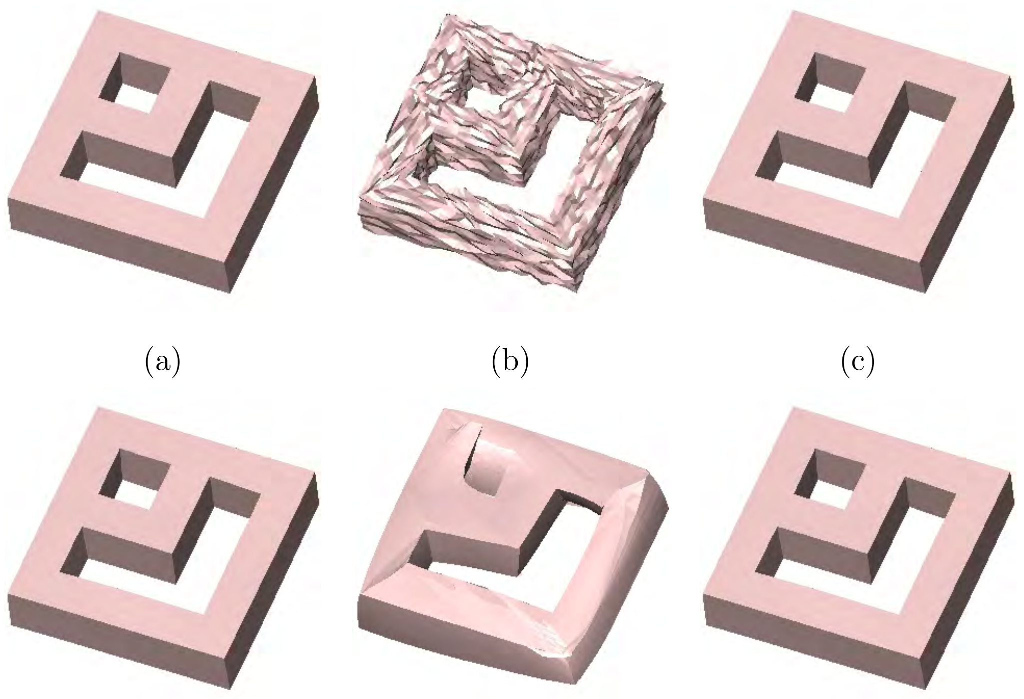
Fig. 7. MSAE for varying numbers of iterations n_1.

at first, but produces smaller MSAE after several iterations. Note that for models with sharp edges like the double-torus, a large value for n_1 produces good-quality results, while for models with a generally curved surface like the bunny, a relatively small value for n_1 yields better results. Overall, it is thus preferable to include the use of the central face for models with sharp edges, but to exclude its use for models lacking sharp edges. In our work, we generally work with models for which we wish to preserve sharp edges, and we thus normally use the central face.

Fifthly, we compare the results of our approach when used in either a progressive or a batch scheme. Fig. 10 shows variation in MSAE with n_1 when applying each scheme to the double-torus model. The progressive scheme almost always yields smaller MSAE than the batch scheme. Thus the algorithm based on a progressive scheme is preferable to one based on a batch scheme.

Finally, we briefly discuss the two possible types of face neighbourhood. Our experiments show that our algorithm using Type I face neighbourhoods generally produces better qualitative mesh results than using Type II face neighbourhoods. On the other hand, the algorithm is faster when using Type II face neighbourhoods. On balance, we suggest using Type I face neighbourhoods in our approach; all other comparisons are based on using Type I face neighbourhoods.

In summary, our experiments show that progressively using Equation (14) with


(a) (b) (c)

(d) (e) (f)

Fig. 8. Denoising a double-torus model ($|V| = 2,686$, $|F| = 5,376$) using our approach. (a) Original model, (b) noisy model (Gaussian noise, standard deviation = 0.2 mean edge length), (c) $\beta = 8$, no adaptive parameter ($t = 1$, $n_1 = 50$), (d) $\beta = 8$, adaptive parameter ($t = 1$, $n_1 = 50$), (e) $\beta = 5$, no adaptive parameter ($t = 1$, $n_1 = 50$), (f) $\beta = 5$, adaptive parameter ($t = 1$, $n_1 = 50$).

β adjusted adaptively is the best scheme when using our approach. Thus, we use this scheme in the rest of our experiments and comparisons.

7.1.3 Experiments on poor-quality triangle meshes

The models used in the above experiments are mainly regular triangle meshes. In this section, we show some results on triangle meshes with poor quality and different sampling density in different regions. Fig. 11(a)-(d) shows the results of our method used on a synthetic cube model with some long triangles. From the figures it can be seen that our method denoises this model very well. Fig. 11(e)-(h) shows the results of our method used on a laser-scanned human face model which has much larger triangles at the mouth, nose and eyes compared to the rest of the face. Note that there are even a few topological errors in the mouth and nose regions. Experimental results show that our method can effectively denoise this model, although it cannot correct its topological errors. We have also made experiments on other irregular triangle meshes; all of them have yielded satisfactory results.

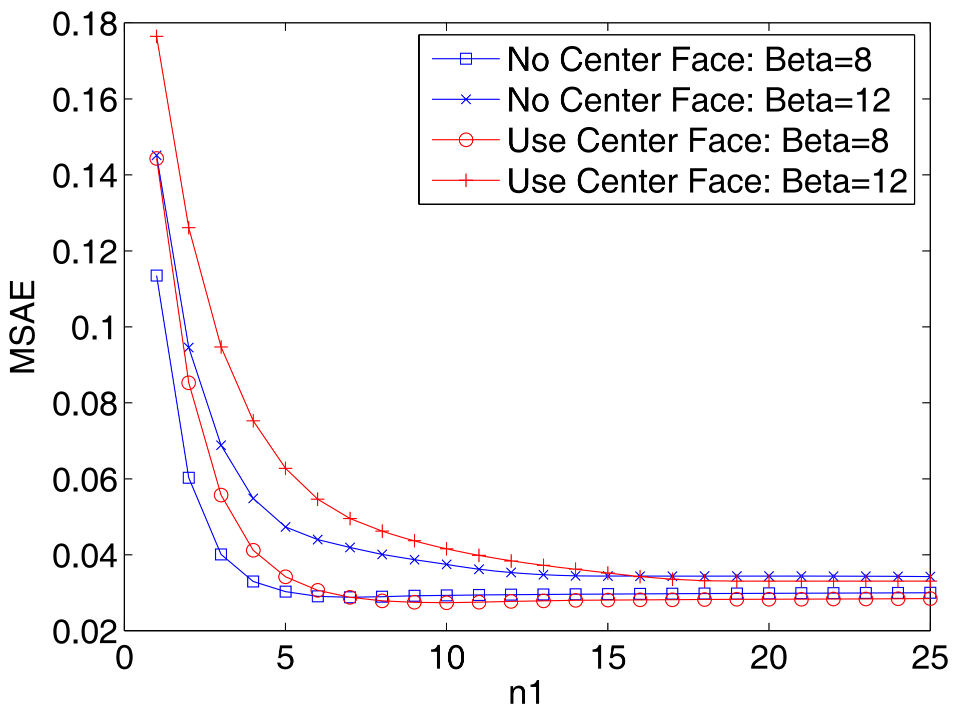


Fig. 9. Algorithms with and without centre face, using the double-torus model.

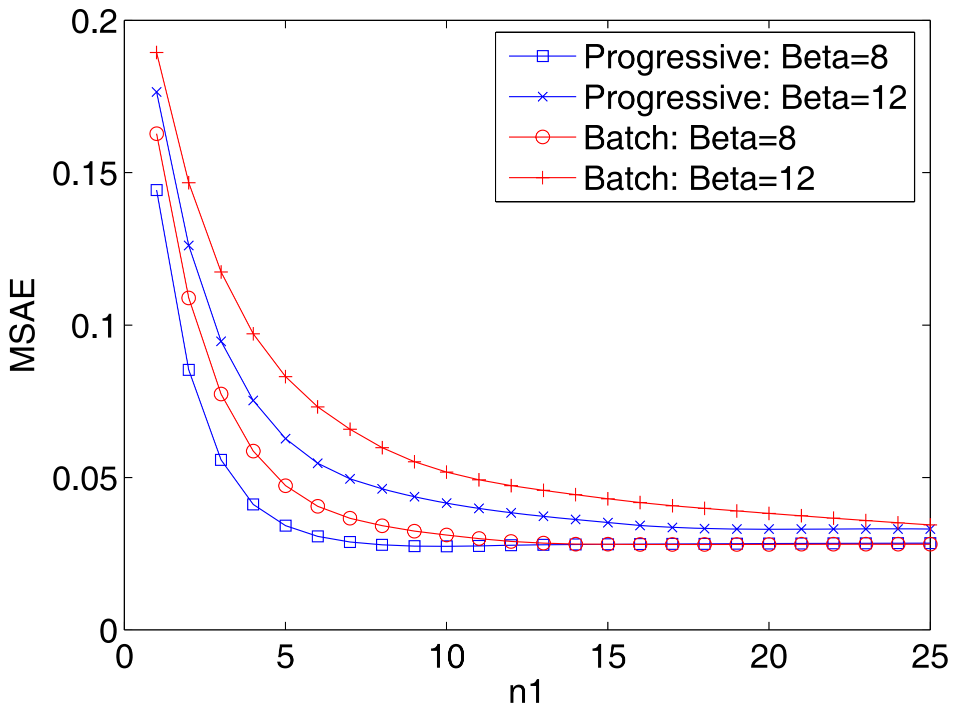


Fig. 10. Progressive and batch schemes, using the double-torus model.

7.2 Comparisons with other approaches

We now turn our attention to comparing our chosen RF approach with the MF (Yagou et al., 2002), BF (Fleishman et al., 2003), and FF (Shen and Barner,

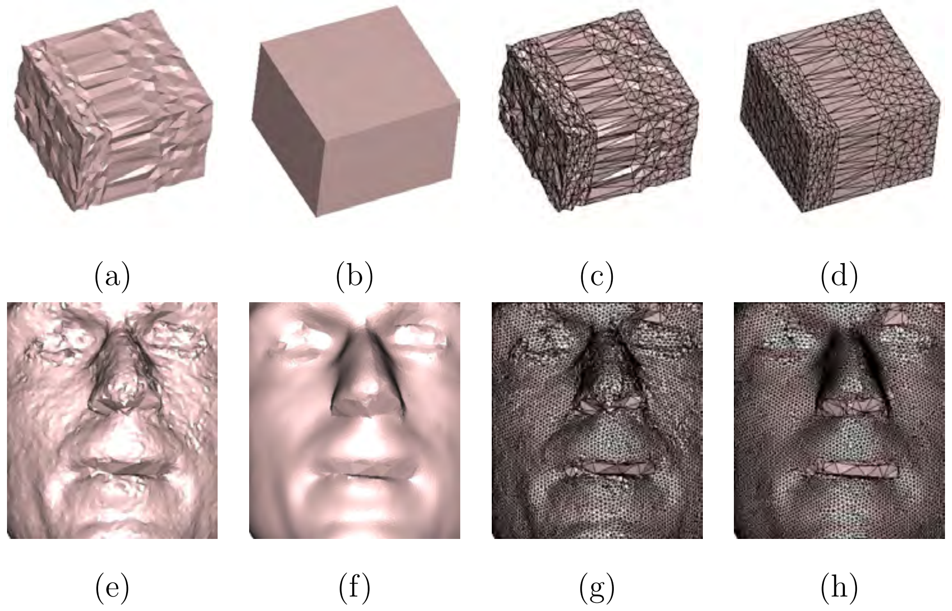


Fig. 11. Experimental results on poor-quality triangle meshes. (a) a noisy cube model with long triangles, (b) the denoising result ($n_1 = 50, \beta = 8$) of (a); (c) and (d) the triangle edges of (a) and (b), respectively; (e) a laser-scanned human face model with different triangle areas and topological errors at the mouth and nose parts, (f) the denoising result ($n_1 = 5, \beta = 8$) of (e), (g) and (h) the triangle edges of (e) and (f), respectively.

2004) approaches mentioned earlier.

7.2.1 Quality

We first visually compare the results obtained. In each case, we show the best results we were able to obtain for each approach after carefully tuning its parameters. All models are rendered using flat shading to aid the visual comparison of normals.

Fig. 12 shows denoising results for a CAD-like model with sharp edges—a double-pyramid. It can be seen that all the four filtering methods preserve sharp features to some extent. However, the BF approach cannot smooth vertices with large errors, as Fleishman et al. (2003) point out. The MF approach cannot smooth flat areas completely, and cannot preserve corner features. In contrast, the FF approach and our RF approach produce surfaces that look very much like the original model. (We also tested mean filtering (Yagou et al., 2002) and alpha-trimming filtering (Yagou et al., 2003), but both methods blur sharp edges, so we have not illustrated the corresponding poor results).

Fig. 13 shows denoising results for a faceted and triangulated cylinder, which has both flat and curved areas, and sharp edges. It can be seen that the

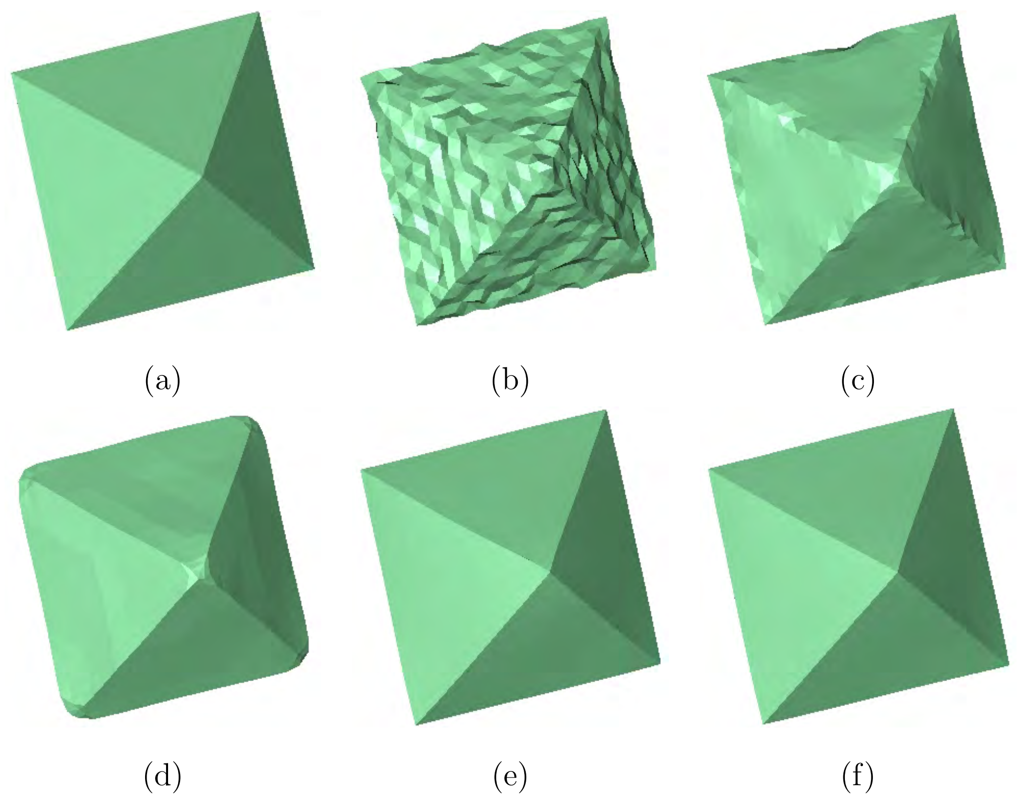


Fig. 12. Denoising of a double-pyramid model ($|V| = 1026, |F| = 2048$). (a) Original model, (b) noisy model (Gaussian noise, standard deviation = 0.2 mean edge length), (c) BF result, (d) MF result, (e) FF result, (f) RF result ($n_1 = 10, \beta = 12$).

BF approach does not preserve sharp edges in this case. The MF approach preserves sharp edges, but also introduces spurious additional sharp edges. The FF approach and our RF approach again preserve both sharp edges and the surface characteristics.

Fig. 14 shows denoising results for a fandisk model. All four approaches preserve most of the sharp edges. The BF and MF approaches even preserve those sharp edges with small angles between the neighbouring surfaces, but on the other hand some corner vertices are not correctly smoothed. Furthermore, the round side edge produced by the BF approach and much of the curved surfaces produced by the MF approach are not particularly smooth. The FF approach and our RF approach produce smooth surfaces and preserve most sharp edges, but have a greater tendency to blur edges with small angles. Although the BF approach appears to preserve sharp edges better on this model than on the previous models, it blurs the sharp edges heavily on the same model if higher levels of noise are added (e.g. 20% of the mean edge length), if adjusted to achieve a reasonably smooth final surface.

Fig. 15 shows denoising results on a mesh model with details at various sizes—the 'iH' embossed Stanford Bunny Model. All approaches do well apart from

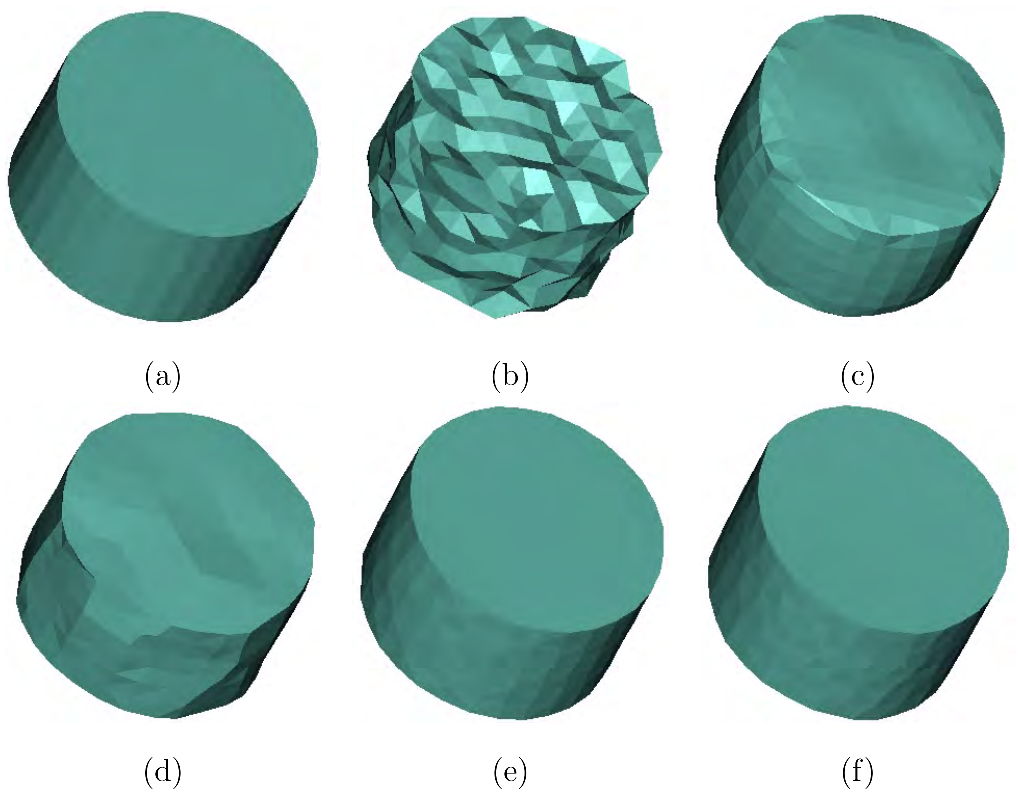


Fig. 13. Denoising of a cylinder model ($|V| = 404, |F| = 804$). (a) Original model, (b) noisy model (Gaussian noise, standard deviation = 0.2 mean edge length), (c) BF result, (d) MF result, (e) FM result, (f) RF result ($n_1 = 10, \beta = 8$).

the MF approach. The MF approach has a tendency to enhance features in the noisy model, and the resulting surface is not smooth. For this model, perhaps the BF approach provides the best overall result; the FF approach, and to a lesser extent the RF approach, lose a little of the finer detail.

Fig. 16 shows results of denoising a scanned model with tiny details—the Moai model. For this model, MF cannot effectively smooth the surface. FF smooths some tiny details away. The BF and RF approaches both preserve tiny details and smooth the surface better. Again, the BF approach seems provides the best overall result.

Figs. 17, 18, and 19 show the results of denoising three 3D photography mesh models. From the figures it can be seen that, as for the Moai model, the MF method enhances certain details, but cannot effectively smooth the surfaces. All three other methods smooth the mesh surfaces while preserving tiny details to some extent. The differences between the results of these three methods are visually very small. From Figs. 17 and 18, it seems that the results of the BF approach and our RF approach are very close and preserve details a little better than the FF approach. However, from Fig. 19, it seems that the results of the FF approach are the best. The result of our RF approach is very close

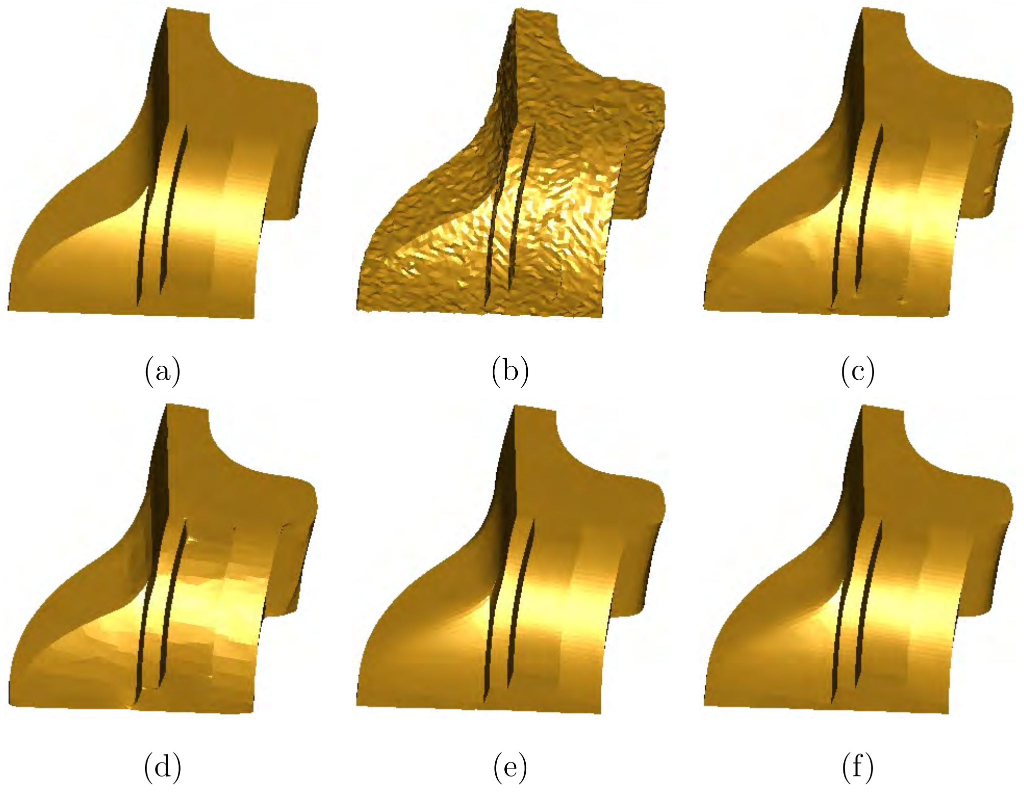


Fig. 14. Denoising of a fandisk model ($|V| = 6,475, |F| = 12,946$). (a) Original model, (b) noisy model (Gaussian noise, standard deviation = 0.1 mean edge length), (c) BF result, (d) MF result, (e) FF result, (f) RF result ($n_1 = 4, \beta = 8$).

to that of the FF approach, and both results of FF and RF approaches are a little better than that of the BF approach.

From the above comparisons using several models of varying types, we can see that the results from the FF approach and our RF approach are generally visually similar, and both produce better results than either the BF or MF approach in cases where sharp edges exist in the models. In cases where there are tiny details in the models, the BF approach probably produces the best results, while our RF approach produces results slightly better than the FF approach.

7.2.2 Speed

We now compare the computational cost of the approaches discussed above. Since the FF approach (Shen and Barner, 2004) generally produces similar results to our RF approach, we first compare these two approaches. Because the vertex position updating stage takes very little time compared to the normal updating stage, we first compare specifically the times taken by the normal updating stages of the RF and BF approaches. Table 2 shows the CPU times recorded in our experiments, including some large (well-known)

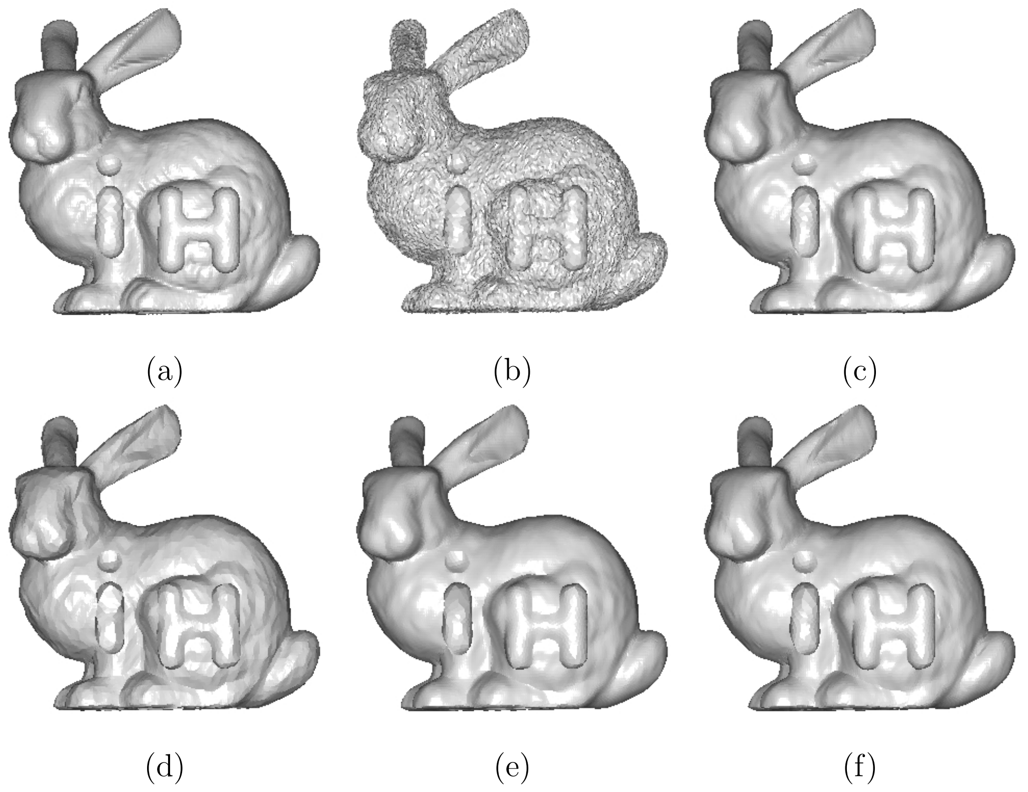


Fig. 15. Denoising of an "iH" embossed Stanford Bunny model ($|V| = 34,834, |F| = 69,451$). (a) Original model, (b) noisy model (Gaussian noise, standard deviation = 0.2 mean edge length), (c) BF result, (d) MF result, (e) FF result, (f) RF result ($n_1 = 3, \beta = 8$).

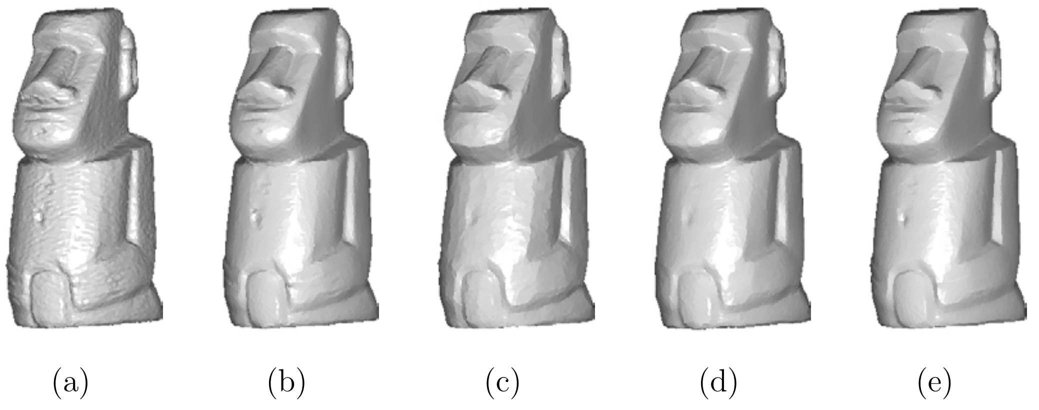


Fig. 16. Denoising of the Moai model ($|V| = 10,002, |F| = 20,000$). (a) Original model, courtesy of Y. Ohtake, (b) BF result, (c) MF result, (d) FF result, (e) RF result ($n_1 = 3, \beta = 30$).

models whose denoising results are not shown in this paper. For comparative purposes, we performed 50 iterations of normal updating for each algorithm, although it is not necessary in practice to use so many iterations. From the table it can be seen that our RF approach is more than ten times faster than the FF approach.

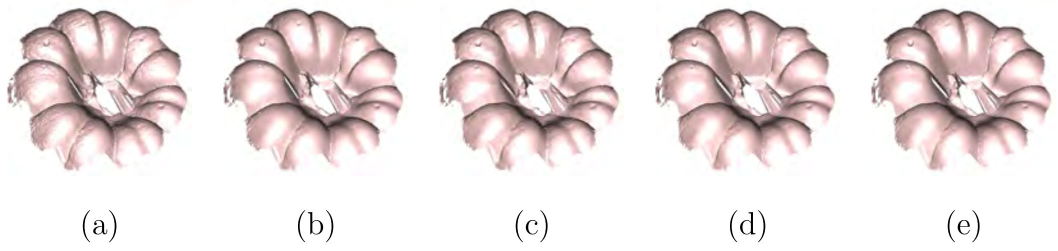


Fig. 17. Denoising of a 3D photography model ($|V| = 14,770, |F| = 28,878$). (a) Original model, courtesy of J.-Y. Bouguet, (b) BF result, (c) MF result, (d) FF result, (e) RF result ($n_1 = 5, \beta = 30$).

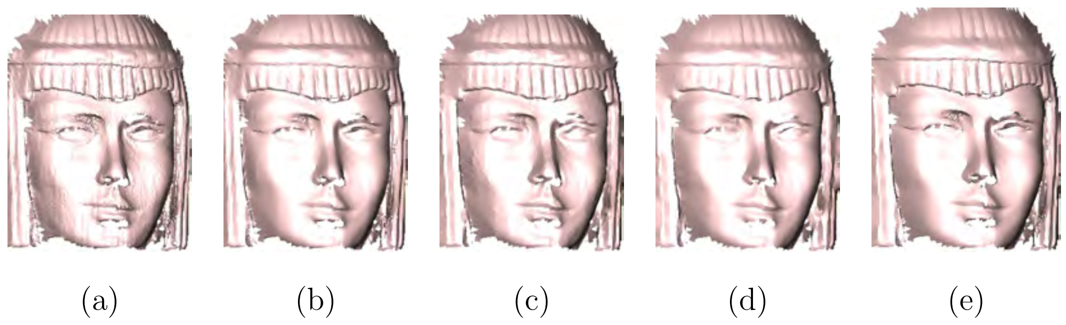


Fig. 18. Denoising of a head model ($|V| = 19,324, |F| = 37,922$). (a) Original model, courtesy of J.-Y. Bouguet, (b) BF result, (c) MF result, (d) FF result, (e) RF result ($n_1 = 3, \beta = 30$).

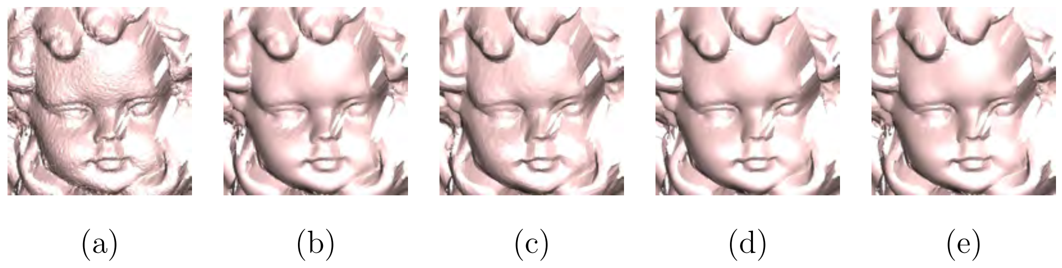


Fig. 19. Denoising of a angel model ($|V| = 24,566, |F| = 48,090$). (a) Original model, courtesy of J.-Y. Bouguet, (b) BF result, (c) MF result, (d) FF result, (e) RF result ($n_1 = 10, \beta = 30$).

We continue in Table 3 by comparing the *overall* time taken by our approach with that required by other approaches. The values in parentheses are the numbers of iterations we found necessary to satisfactorily denoise the models. For the BF and MF approaches these correspond to the only iteration parameter, for the FF approach to n_1, n_2 and for our RF approach to n_1. Overall, the BF approach is generally fastest. However, our approach requires a time similar to that of BF; sometimes, our approach is even faster than BF. The other approaches take significantly longer.

In summary, we conclude that our new method has significant advantages, providing denoising results of a *quality* often comparable to the slowest of these methods, with nearly the *speed* of the fastest.

Table 2
Normal updating times for RF and FF methods (seconds, for 50 iterations)

	Fandisk	iH Bunny	Igea	Dragon	Buddha
$\lvert V\rvert$	6475	34834	134345	437645	757490
$\lvert F\rvert$	12946	69451	268686	871414	1514962
RF	0.703	4.078	15.391	47.657	83.078
FF	9.391	48.406	196.531	677.719	1263.06

Table 3
Overall times for various methods (seconds, for given numbers of iterations)

	Fandisk	iH Bunny	Igea	Dragon	Buddha
BF	0.046	0.313	1.313	3.75	7.094
	(5)	(5)	(5)	(5)	(5)
MF	0.281	2.594	7.25	33.219	39.047
	(10)	(15)	(10)	(15)	(10)
FF	1.891	5.141	12.641	140.438	70.093
	(10, 10)	(5, 20)	(3, 10)	(10, 15)	(3, 10)
RF	0.078	0.422	1.484	6.078	6.922
	(4)	(3)	(3)	(5)	(3)

8 Conclusions and future work

In this paper, we have shown how to use *random walks* for mesh denoising, and proposed a new two-stage mesh denoising algorithm. Initially, the face normals are updated through weighted averaging of the face normals, with the weights being determined by probabilities of random walk steps between each face and its neighbours; these probabilities in turn depend on differences in face normals. Analysis and experiments show that the scheme in Equation (14), together with adaptively adjusting parameter β, and progressively updating face normals, provides the best implementation of our approach. To find the final vertex positions, we use a conjugate gradient algorithm to solve the appropriate least-squares problem, rather than the more generally used gradient descent algorithm (Taubin, 2001; Ohtake et al., 2001; Shen and Barner, 2004; Sun et al., 2007). The conjugate gradient approach is stable and converges rapidly, and is particularly suitable for solving the least-squares problem arising here as it leads to a sparse system.

A basic requirement for a mesh denoising algorithm is that it can both remove noise and preserve mesh features effectively. However, many early mesh denoising algorithms did not consider the feature-preserving requirement. Several more recent mesh denoising methods do consider it, but most such methods are computationally expensive. Our proposed mesh denoising algorithm effectively preserves features and yet is very simple and computationally cheap. Experiments presented here have compared our approach with other recent feature-preserving mesh denoising approaches. Bilateral filtering (Fleishman et al., 2003) is a fast feature-preserving mesh denoising approach. Experiments show that our approach is as fast as the bilateral filtering approach (Fleishman et al., 2003): e.g. it can denoise the well-known Buddha model with 1.5 million triangles within 7 seconds; however, our approach preserves sharp edges better than the bilateral filtering approach. Compared to the fuzzy vector median filtering approach (Shen and Barner, 2004), our approach is over ten times faster, yet produces a final surface quality similar to or better than that approach.

We have also compared our current RF approach with the algorithm proposed in Sun et al. (2007). In general, the difference between the results of the current approach and that in Sun et al. (2007) is small, so that it is hard to distinguish them visually. Also, their time costs are quite close. Although the RF approach proposed in this paper does not produce results that are significantly better than that in Sun et al. (2007), it is still very useful as it provides a general framework for researchers to develop new mesh denoising methods, maybe with quite different requirements. In addition, the distribution function used in this method has a control parameter which may be adaptively adjusted to further enhance the feature-preserving properties of the method, which the approach in Sun et al. (2007) cannot achieve. Furthermore, and most importantly, it is expected that the idea of using random walks has the potential for application to many other problems in mesh processing, such as mesh edge detection, mesh segmentation, saliency detection, feature point and feature triangle detection.

Although our algorithm is simple and efficient for feature-preserving mesh denoising, it is not immune to certain problems that other algorithms also meet. One is that we have to interactively determine the number of normal updating iterations. Using too few iterations fails to fully denoise the mesh normals, while too many causes oversmoothing of the mesh. Future work is needed to find an automatic method of determining the optimal number of iterations. Other problems such as mesh folding, self-interaction and poorly-shaped triangles caused by vertex position updates should also be considered in future work.

Acknowledgements

This work was supported by EPSRC Grant EP/C007972 and NSFC Grant 60674030.

References

Azzabou, N., Paragios, N., Guichard, F., 2006. Random walks, constrained multiple hypothesis testing and image enhancement. In: Leonardis, A., Bischof, H., Pinz, A. (Eds.), ECCV 2006 Proceedings. Vol. 1. Springer, pp. 379–390.

Bajaj, C. L., Xu, G., 2003. Anisotropic diffusion of surfaces and functions on surfaces. ACM Transactions on Graphics 22 (1), 4–32.

Botsch, M., Bommes, D., Kobbelt, L., 2005. Efficient linear system solvers for mesh processing. In: IMA Conference on the Mathematics of Surfaces. Vol. 3604 of Lecture Notes in Computer Science. Springer, pp. 62–83.

Botsch, M., Kobbelt, L. P., 2001. Resampling feature and blend regions in polygonal meshes for surface anti-aliasing. In: Chalmers, A., Rhyne, T.-M. (Eds.), EG 2001 Proceedings. Vol. 20(3). Blackwell Publishing, pp. 402–410.

Chen, C.-Y., Cheng, H.-Y., 2005. A sharp dependent filter for mesh smoothing. Computer Aided Geometric Design 22, 376–391.

Choudhury, P., Tumblin, J., 2003. The trilateral filter for high contrast images and meshes. In: EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 186–196.

Clarenz, U., Diewald, U., Rumpf, M., 2000. Anisotropic geometric diffusion in surface processing. In: Proceedings of the Conference on Visualization 2000. IEEE Computer Society, pp. 397–405.

Desbrun, M., Meyer, M., Schröder, P., Barr, A. H., 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In: SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, pp. 317–324.

Desbrun, M., Meyer, M., Schröder, P., Barr, A. H., 2000. Anisotropic feature-preserving denoising of height fields and bivariate data. In: Graphics Interface'2000 Proceedings. pp. 145–152.

Diebel, J. R., Thrun, S., Brünig, M., 2006. A Bayesian method for probable surface reconstruction and decimation. ACM Transactions on Graphics 25 (1), 39–59.

Field, D. A., 1988. Laplacian smoothing and Delaunay triangulations. Communications in Numerical Methods in Engineering 4, 709–712.

Fleishman, S., Drori, I., Cohen-Or, D., 2003. Bilateral mesh denoising. ACM Transactions on Graphics 22 (3), 950–953.

Grady, L., 2006. Random walks for image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (11), 1768–1783.

Hildebrandt, K., Polthier, K., 2004. Anisotropic filtering of non-linear surface features. Computer Graphics Forum 23 (3), 391–400.

Jones, T. R., Durand, F., Desbrun, M., 2003. Non-iterative, feature-preserving mesh smoothing. ACM Transactions on Graphics 22 (3), 943–949.

Jones, T. R., Durand, F., Zwicker, M., 2004. Normal improvement for point rendering. IEEE Computer Graphics and Applications 24 (4), 53–56.

Kim, B., Rossignac, J., 2005. Geofilter: Geometric selection of mesh filter parameters. Computer Graphics Forum 24 (3), 295–302.

Kobbelt, L., Campagna, S., Vorsatz, J., Seidel, H.-P., 1998. Interactive multi-resolution modeling on arbitrary meshes. In: SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques. ACM Press, New York, NY, USA, pp. 105–114.

Lee, K.-W., Wang, W.-P., 2005. Feature-preserving mesh denoising via bilateral normal filtering. In: CAD-CG '05: Proceedings of the Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05). IEEE Computer Society, Washington, DC, USA, pp. 275–280.

Li, W., Goh, L. P., Hung, T., Xu, S., 2005. Adaptive mesh smoothing for feature preservation. In: Computational Science and Its Applications ICCSA 2005. Vol. 3483 of Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 906–915.

Nehab, D., Rusinkiewicz, S., Davis, J., Ramamoorthi, R., 2005. Efficiently combining positions and normals for precise 3D geometry. ACM Transactions on Graphics 24 (3), 536–543.

Nehorai, A., Hawkes, M., 2000. Performance bounds for estimatin vector systems. IEEE Transactions on Signal Processing 48 (6), 1737–1749.

Ohtake, Y., Belyaev, A., Bogaevski, I., 2001. Mesh regularization and adaptive smoothing. Computer-Aided Design 33 (11), 789–800.

Ohtake, Y., Belyaev, A., Seidel, H.-P., 2002. Mesh smoothing by adaptive and anisotropic Gaussian filter applied to mesh normals. In: Vision, Modeling, and Visualization 2002. pp. 203–210.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P., 1992. Numerical Recipes in C: The Art of Scientific Computing, 2nd Edition. Cambridge University Press, Cambridge, U.K.

Schall, O., Belyaev, A., Seidel, H.-P., 2007. Feature-preserving non-local denoising of static and time-varying range data. In: SPM '07: Proceedings of the 2007 ACM symposium on Solid and physical modeling. ACM Press, New York, NY, USA, pp. 217–222.

Shen, J., Maxim, B., Akingbehin, K., 2005. Accurate correction of surface noises of polygonal meshes. International Journal for Numerical Methods in Engineering 64 (12), 1678–1698.

Shen, Y., Barner, K. E., 2004. Fuzzy vector median-based surface smoothing. IEEE Transactions on Visualization and Computer Graphics 10 (3), 252–265.

Shimizu, T., Date, H., Kanai, S., Kishinami, T., 2005. A new bilateral mesh smoothing method by recognizing features. In: CAD-CG '05: Proceedings of the Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05). IEEE Computer Society, Washington, DC, USA, pp. 281–286.

Smolka, B., Wojciechowski, K. W., 2001. Random walk approach to image enhancement. Signal Processing 81 (3), 465–482.

Spitzer, F., 2001. Principles of Random Walk, 2nd Edition. Springer, New York.

Sun, X., Rosin, P. L., Martin, R. R., Langbein, F. C., 2007. Fast and effective feature-preserving mesh denoising. IEEE Transactions on Visualization and Computer Graphics 13 (5), 925–938.

Szczepanski, M., Smolka, B., Plataniotis, K. N., Venetsanopoulos, A. N., 2003. On the geodesic paths approach to color image filtering. Signal Processing 83 (6), 1309–1342.

Tasdizen, T., Whitaker, R., Burchard, P., Osher, S., 2002. Geometric surface smoothing via anisotropic diffusion of normals. In: Proceedings of the Conference on Visualization 2002. IEEE Computer Society, pp. 125–132.

Taubin, G., 1995. A signal processing approach to fair surface design. In: SIGGRAPH'95 Conference Proceedings. pp. 351–358.

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

Vollmer, J., Mencl, R., Müller, H., 1999. Improved Laplacian smoothing of noisy surface meshes. Computer Graphics Forum 18 (3), 131–138.

Wechsler, H., Kidode, M., 1979. A random walk procedure for texture discrimination. IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-1 (3), 272–280.

Welch, W., Witkin, A., 1994. Free-form shape design using triangulated surfaces. In: SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques. ACM Press, New York, NY, USA, pp. 247–256.

Yagou, H., Ohtake, Y., Belyaev, A. G., 2002. Mesh smoothing via mean and median filtering applied to face normals. In: Proceedings of Geometric Modeling and Processing. pp. 124–131.

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

Yoshizawa, S., Belyaev, A., Seidel, H.-P., 2006. Smoothing by example: Mesh denoising by averaging with similarity-based weights. In: SMI '06: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006 (SMI'06). IEEE Computer Society, Washington, DC, USA, p. 9.

Yu, Y., Zhou, K., Xu, D., Shi, X., Bao, H., Guo, B., Shum, H., 2004. Mesh editing with Poisson-based gradient field manipulation. ACM Transactions on Graphics 23 (3), 644–651.