Random Walks for Mesh Denoising

Xianfang Sun*
Cardiff University, UK
Beihang University, China

Paul L. Rosin†
Cardiff University, UK

Ralph R. Martin‡
Cardiff University, UK

Frank C. Langbein§
Cardiff University, UK

Abstract

This paper considers an approach to mesh denoising based on the concept of random walks. The proposed method consists of two stages: a face normal filtering procedure, followed by a vertex position updating procedure which integrates the denoised face normals in a least-squares sense. Face normal filtering is performed by weighted averaging of normals in a neighbourhood. The weights are based on the probability of arriving at a given neighbour after a random walk of a virtual particle starting at a given face of the mesh and moving a fixed number of steps. The probability of a particle stepping from its current face to a given neighboring face is determined by the angle between the two face normals, using a Gaussian distribution whose width 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. In particular, iterative application of a one-step random walk in a progressive manner effectively preserves detailed features while denoising the mesh very well. We observe that this approach is faster than many other feature-preserving mesh denoising algorithms.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling—Curve, surface, solid, and object representations

Keywords: Mesh Denoising, Mesh Smoothing, Random Walk, Feature Preservation

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. When there are intrinsic fine details or sharp features in a noisy mesh, it is hard to both denoise the mesh and preserve the mesh features. In this paper, we present a new feature-preserving mesh denoising method based on a random walk model.

In basic 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 their estimated noise-free positions. 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 are often computed in two steps. *One-step* approaches directly update vertex positions using the original vertex coordinates, sometimes together with face normal information, in a neighbourhood around the current vertex. *Two-step* approaches first adjust face normals and then update vertex positions using some error minimization criterion 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 $(\text{Step } 1 + \text{Step } 2)^{n_0}$, or two separate iteration procedures can be performed, informally $(\text{Step } 1)^{n_1} + (\text{Step } 2)^{n_2}$, where n_0, n_1 and n_2 are numbers of iterations. To distinguish these two schemes, we call these *interleaved* and *consecutive* iteration schemes respectively.

The relative advantages of these two schemes merits detailed investigation. One advantage of the interleaved scheme is that it only depends on choice of a single iteration parameter n_0, while the consecutive scheme involves two iteration parameters n_1 and n_2. However, in an interleaved scheme, normal updating and vertex position updating have different optimization end points, so their interaction generally means that more iterations are needed for a given degree of denoising than in a consecutive scheme, i.e. $\max(n_1, n_2) < n_0$. In addition, small normal errors, but not vertex position errors, may cause significant aliasing problems [Botsch and Kobbelt 2001], so we need to pay more attention to the normal updating step. Hence, it is preferable to separately process normal updating and vertex position updating. The approach proposed in this paper is thus a *consecutive* iterative method.

Random walks are used as models in many areas of mathematics and physics. Application of random walk models to computer vision and image processing first appeared in the work of Wechsler and Kidode [1979] for texture discrimination. More recently, random walks have been applied to 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, it appears that applying random walks to mesh processing is new. Random walks on an image only deal with a 2D domain, which furthermore has a regular structure. It is not straightforward to extend such 2D methods to 3D mesh applications. In this paper, we provide 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 organized as follows. Section 2 gives a brief review of previous work on mesh denoising. Section 3 describes the notation used in this paper. 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 introduce an adaptive parameter adjustment method, and analyze the feature-preserving property of our approach. Section 6 states 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.

*e-mail: Xianfang.Sun@cs.cardiff.ac.uk
†e-mail: Paul.Rosin@cs.cardiff.ac.uk
‡e-mail: Ralph.Martin@cs.cardiff.ac.uk
§e-mail: F.C.Langbein@cs.cardiff.ac.uk

SPM 2007, Beijing, China, June 04 – 06, 2007.
© 2007 ACM 978-1-59593-666-0/07/0006 $5.00

2 Previous Work

Many surface *smoothing*, *fairing* and *denoising* methods have been proposed, and to a large degree their differing aims have been confused—smoothing algorithms have often been suggested for noise removal. Classical Laplacian smoothing [Field 1988; Vollmer et al. 1999] 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 [1995] proposed a filtering method with positive and negative damping factors. A first-order filter with positive damping factor shrinks and smooths the mesh surface, while a first-order filter with negative damping factor expands the surface, to compensate for the shrinkage. This method is fast and simple, but still suffers from 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. [1999] introduced diffusion and curvature flow into surface fairing, proposing 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.

From the viewpoint of signal processing, Taubin's [1995] and Desbrun et al.'s [1999] methods can both be though of as filtering methods. The former can be considered as moving average (MA), or finite impulse response (FIR) filtering, while the later can be seen as autoregressive (AR), or infinite impulse response (IIR) filtering. Combining the above two approaches, Kim and Rossignac [2005] developed a general autoregressive moving average (ARMA) 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 does both well.

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, particularly edges. Thus, various *anisotropic* filtering schemes have been proposed which smooth surfaces while simultaneously preserving edge features. Anisotropic filtering schemes can be divided into three main classes, plus various others.

The first 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]. Such methods have been used for smoothing height fields and bivariate data [Desbrun et al. 2000], level set surfaces [Tasdizen et al. 2002], and general discretized surfaces [Clarenz et al. 2000; Bajaj and Xu 2003; Hildebrandt and Polthier 2004].

The second class is based on bilateral filters [Jones et al. 2003; Fleishman et al. 2003]. Fleishman et al. [2003] 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 used 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 [2003] 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 [Ohtake et al. 2001; Taubin 2001; Yagou et al. 2002; Yagou et al. 2003; Shen and Barner 2004; Chen and Cheng 2005]. Ohtake et al.'s [2001] nonlinear diffusion method, Yagou et al.'s [2002; 2003] mean, median, and alpha-trimming methods, and Chen and Cheng's [2005] sharpness dependent method are interleaved iterative two-step approaches. Taubin's [2001] anisotropic filtering algorithm and Shen and Barner's [2004] fuzzy vector median filtering approach are consecutive iterative two-step approaches.

Other approaches have also been considered. Shen et al.'s [2005] 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 methods. Nehab et al. [2005] give an energy minimization method in which the energy is the sum of the position error and the normal error. Diebel et al. [2006] use a Bayesian technique for reconstruction and decimation of noisy 3D surface models, again based on an energy minimization problem where the energy is a sum of position and normal errors. [Nehab et al. 2005] uses additional information about the measured normals which the latter does not, yielding a linear solution unlike Diebel's method.

The above-mentioned anisotropic filtering approaches generally suffer from one of two problems: either they do not preserve features effectively, or they are complex and computationally expensive. Our approach preserves features efficiently *and* is computationally inexpensive.

3 Notation

We use $T = (V, E, F, X)$ to reporesent a triangular mesh, where $V = \{i : i = 1, ..., n\}$ 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 face set, and $X = \{\mathbf{x}_i : \mathbf{x}_i \in \mathbb{R}^3,\ i \in V\}$ is the vertex coordinate set. We use $|\cdot|$ to denote the cardinality of a set: 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 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 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 $N_V(i)$, 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, and 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. Note that the $N_{FI}(i) \supset N_{FII}(i)$. We will also wish to refer to the union of F_i and its neighbourhood, so we define $N^*_{FI}(i) = N_{FI}(i) \bigcup \{F_i\}$ and $N^*_{FII}(i) = N_{FII}(i) \bigcup \{F_i\}$.

4 Markov Chains and Random Walks

In this section, we introduce random walks from a practical point of view. For more mathematical descriptions of these concepts, readers are 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, given the present state, the

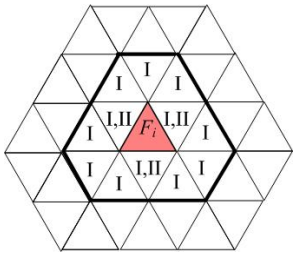

Figure 1: Face neighbourhoods. Faces labeled I belong to $N_{FI}(i)$; faces labeled II belong to $N_{FII}(i)$.

future state is conditionally independent of earlier states. In other words,

$$P(X_{n+1}=x_{n+1}|X_n=x_n,X_{n-1}=x_{n-1},\ldots,X_0=x_0)= \\ P(X_{n+1}=x_{n+1}|X_n=x_n). \quad (1)$$

The possible values of X_t form a countable set called the *state space* which can be either finite or infinite. In this paper, we only consider finite state spaces, and we simply use an index set $\mathscr{I}=\{1,\ldots,N\}$ to represent the state space.

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

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

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 $P(n)=P(0)\Pi^n$ after n steps, where $\Pi^n=\Pi(1)\cdots\Pi(n)$. We call Π^n the *n-step transition probability matrix*. The $(i,j)^{th}$ element of Π^n is denoted by $p^n_{i,j}$, and is the probability of going from state i to j after n steps.

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

5 Normal Filtering

In this section, we discuss we use a random walk model to denoise the face normals. The basic motivation is that if probabilities of stepping from one triangle to another depend on how similar their normals are, and then we average normals according to the final probabilities, we will give greater weight to similar triangles (similar parts of the surface) and less weight to ones that are e.g. on the other side of an edge feature. Similar ideas were used by Smolka and Wojciechowski [2001] for image denoising.

5.1 Random Walk for Normal Filtering

At the initial time, we suppose that 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 can move to a neighbour of its current face, or stay in its current position, with probabilities that depend on the face normals. After n steps of such a random walk, the particles will have been redistributed on the mesh surface according to the n-step transition probability matrix Π^n, which we then use to compute the new face normals in our normal filtering algorithm. We perform normal updating using

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

The updated normal here is a weighted averaging of the normals of the faces on the whole mesh. Note that weighted averaging is a frequently used approach to updating normals in the literature: [Taubin 2001; Ohtake et al. 2001; Ohtake et al. 2002; Yagou et al. 2002; Yagou et al. 2003; Shen and Barner 2004].

To implement Equation (2), we need to know how to compute $p^n_{i,j}$. This depends on two elements: the choice of n, the number of steps, and $p_{i,j}(n)$, 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 we should use of the virtual particle on one face visiting the other face. Hence, the single-step transition probability $p_{i,j}(n)$ should be a decreasing function of the normal difference $\|\mathbf{n}_i-\mathbf{n}_j\|$. Moreover, it is usually required that this function is also convex in $[0,\infty)$, and tends to zero as its variable tends to infinity (of course, the normal difference here is in the range $[0,2]$, rather than $[0,\infty)$, but this makes little difference for the function we choose later). Numerous functions satisfy the above conditions. Typical functions used in the literature are [Szczepanski et al. 2003]

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

where $\beta\in(0,\infty)$ is a parameter, and C is a normalization coefficient to make the probabilities of all possible events sum to one.

A priori knowledge and experiments can help to choose a suitable function. In this work, we have used the Gaussian function $f_1(x)$

since the Gaussian distribution occurs commonly in the real world, and our experiments show that it yields very good results. 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 Ohtake et al.'s [2001; 2002] is that we have chosen different variables, and our variable—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 have, after combining the above coefficient 2 into the parameter β of the Gaussian function $f_1(x)$,

$$p_{i,j}(n) = \begin{cases} Ce^{-\beta(1-\mathbf{n}_i\cdot\mathbf{n}_j)} & \text{if } j \in N_F(i) \\ 0 & \text{otherwise} \end{cases}, \tag{12}$$

where the normalization coefficient C is given by

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

and $N_F(i)$ 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 the choice of the number of steps, n, for the walk. As discussed in Section 4, as n becomes larger, more nonzero elements appear in the matrix Π^n, which means that more face normals are used in the computation of the new normal in Equation (2), and better results can be obtained. However, because the number of nonzero 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, n must be large enough to obtain a result satisfying the qualitative requirements of denoising. However, if we adopt an iterative scheme, then using small n can also produce good results in conjunction with several iterations. 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 n on the final quality and computational cost, we first give the face normal updating formulae for different n. In the simplest case $n = 1$, we get $p^1_{i,j} = p_{i,j}(1)$. So the face normal updating formula in Equation (2) becomes

$$\mathbf{n}'_i = \frac{\sum_{j\in N^*_F(i)} e^{-\beta(1-\mathbf{n}_i\cdot\mathbf{n}_j)}\mathbf{n}_j}{\left|\sum_{j\in N^*_F(i)} e^{-\beta(1-\mathbf{n}_i\cdot\mathbf{n}_j)}\mathbf{n}_j\right|}, \tag{14}$$

or,

$$\mathbf{n}'_i = \frac{\sum_{j\in N_F(i)} e^{-\beta(1-\mathbf{n}_i\cdot\mathbf{n}_j)}\mathbf{n}_j}{\left|\sum_{j\in N_F(i)} e^{-\beta(1-\mathbf{n}_i\cdot\mathbf{n}_j)}\mathbf{n}_j\right|}, \tag{15}$$

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 normalization coefficient C does not need to explicitly appear in the above formulae because the last step of computing $\mathbf{n}'_i$ is to normalize 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 $\mathbf{n}'_j$ in this iteration when computing $\mathbf{n}'_i$. We call the former scheme the *batch* scheme and the latter the *progressive* one. It is expected that the progressive scheme will 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 $n > 1$. If we directly use Equation (2) to update normals, we need to compute Π^n. Because Π^n will become non-sparse as n grows, the computational cost will grow quickly, and additional memory will be required to store the whole matrix Π^n. To save memory and computation time, we propose to update normals sequentially:

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

or, with a different neighborhood,

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

and finally,

$$\mathbf{n'}_i = \frac{\mathbf{n}_i(n)}{|\mathbf{n}_i(n)|}. \tag{18}$$

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 $k = n$ 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 given k, only a sparse matrix $\Pi(k)$ is required in the computation of Equation (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 $n > 1$. If we adopt a *progressive* scheme, i.e. once we obtain a $\mathbf{n}_i(k)$, we immediately normalize 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 n-step algorithm in Equation (16) or (17) is equivalent to n 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 a 1-step or n-step algorithm.

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 time—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 n-step walks. We can consider this procedure as n iterations of 1-step random walks. We can also consider it as one iteration of n-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 give a qualitative indication for choice of β: when the model noise is high, β should be small, and vice versa. On the other hand, 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 $\mathbf{n}'_i$. While the above qualitative analysis can guide the choice of β, more 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 will reduce after each iteration. The qualitative analysis suggests that we should dynamically adjust β so that it becomes larger after each iteration. In their work, Smolka and Wojciechowski [2001] suggested that β should be adjusted using $\beta' = \delta\beta$, where $\beta > 1$; see also [Szczepanski et al. 2003]. We have performed experiments using such a parameter adjustment scheme, but found that it only provides a small improvement.

Instead, we introduce an alternative adaptive method for parameter adjustment, using a similar idea to that introduced by Shen and Barner [2004]. 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. We wish to minimize the cost function $J(\beta) = E(|\mathbf{n}_{i0} - \mathbf{n}'_i(\beta)|)$, where E is the expectation operator. This is equivalent to

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

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

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

where μ is the 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 n-step scheme in Equation (16) or (17) is equivalent to n iterations of the 1-step scheme in Equation (14) or (15), respectively. Let us now define

$$\tilde{\mathbf{n}}_i = \sum_{j \in N_F^*(i)} e^{-\beta(1-\mathbf{n}_i \cdot \mathbf{n}_j)} \mathbf{n}_j. \tag{21}$$

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

$$\dot{\tilde{\mathbf{n}}}_i = \frac{\partial \tilde{\mathbf{n}}_i}{\partial \beta} = -\sum_{j \in N_F^*(i)} (1 - \mathbf{n}_i \cdot \mathbf{n}_j) e^{-\beta(1-\mathbf{n}_i \cdot \mathbf{n}_j)} \mathbf{n}_j. \tag{22}$$

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 parameter updating rule based on only one face on the mesh. Because the optimal parameter 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), initial values of β and μ 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. We have used $\beta = 8$ 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 the neighbouring face, the further the normal of a neighbouring face deviates from that of the central face, 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 the neighbouring faces have small influence on each other, and 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 minimize 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 time-consuming in comparison to 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 to do this [Taubin 2001; Ohtake et al. 2001; Ohtake et al. 2002; Yu et al. 2004]. 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. Since in general this system of equations has no non-trivial solution, he solves it in a least-squares sense using the gradient descent method. Unfortunately, this method converges slowly, and if the step size is not suitably chosen, it may be unstable. Ohtake et al. [2001] gave a vertex updating algorithm similar to Taubin's with a particular step size and additional area weights. Because it is intrinsically a gradient descent method, it also converges slowly. Ohtake et al. [2002] propose another vertex position updating algorithm based on the minimization of the

area-weighted sum of the squared differences between the original and the new face normals. The solution to this minimization problem is also performed using gradient descent. However, since its gradient computation is more complex, it is computationally more expensive than Taubin's algorithm. Again, it also has the problem of choosing a suitable step size. Yu et al. [2004]'s method 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 computation of 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]:

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

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 generalize 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. In fact, 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 here to make our paper self-contained.

Let us 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 together with $\mathbf{g}_i$, $\mathbf{p}_i$, and $\mathbf{q}_k$ in the following way:

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

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

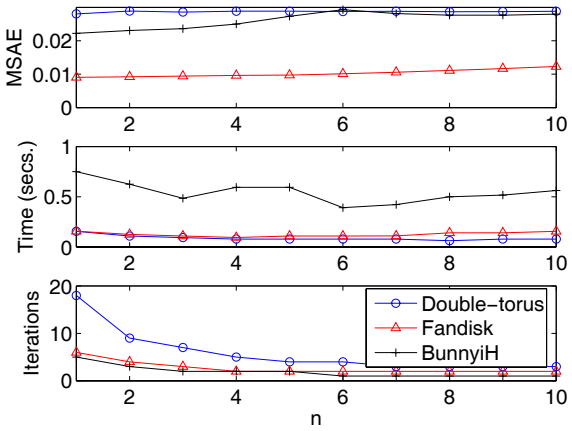

Figure 2: Effect of varying numbers of random walk steps $n = 1, \ldots, 10$. Top: optimal mean square angular errors acheived. Center: computational times. Bottom: number of iterations needed for optimal errors.

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

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

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

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

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 $\|X' - X\| < \varepsilon$, or $\|G\| < \varepsilon$. 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 demonstrates results of tests carried out on our random walk filtering (RF) approach, which we also compare to several other approaches: Yagou et al.'s [2002] median filtering (MF), Fleishman et al.'s [2003] bilateral filtering (BF), and Shen and Barner's [2004] fuzzy vector median filtering (FF). The algorithms were implemented in VC++.net, and our experiments were performed on a PC with a 3.2GHz Intel Xeon CPU and 2.0GB 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 the best scheme among them. 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

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

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

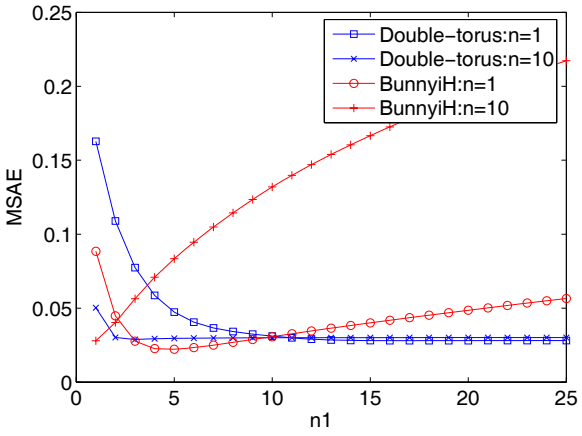

Figure 3: MSAE for varying numbers of iterations n_1.

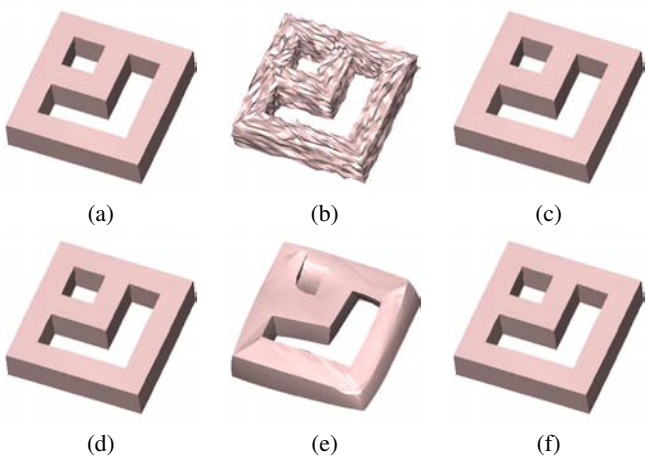

Figure 4: Denoising a double-torus model ($|V| = 2,686$, $|F| = 5,376$) using our approach. (a) Original model, courtesy of MPII, (b) noisy model (Gaussian noise, standard deviation = 0.2 mean edge length), (c) $\beta = 8$, no adaptive parameter ($n = 1$, $n_1 = 50$), (d) $\beta = 8$, adaptive parameter ($n = 1$, $n_1 = 50$), (e) $\beta = 5$, no adaptive parameter ($n = 1$, $n_1 = 50$), (f) $\beta = 5$, adaptive parameter ($n = 1$, $n_1 = 50$).

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.

Firstly, we discuss the effect of varying the number of random walk steps n. Fig. 2 shows the optimal MSAEs achieved, the corresponding computation times, and the numbers of iterations needed for several models illustrated later in the paper. It can be seen that lower n generally produces slightly smaller optimal MSAE, but there are no clear trends in computation time. Because the variations in these optimal MSAEs and computation times are not significant, no clear preference exists for the choice of n. We suggest choosing $n = 1$ for simplicity. It can also be seen from Fig. 2 that the optimal number of iterations reduces gradually as n 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.

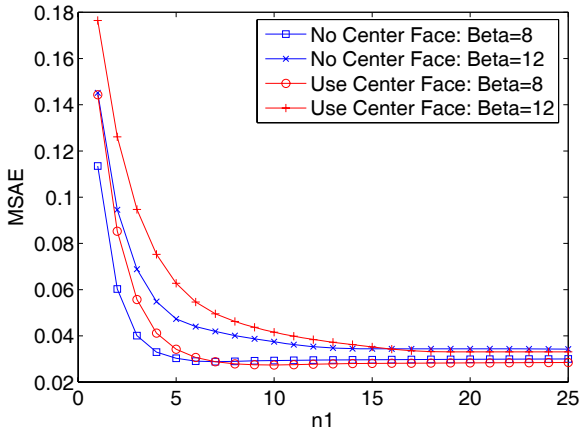

Figure 5: Algorithms with and without center face, using the double-torus model.

Secondly, we consider the relationship between the MSAE and the number of iterations n_1. Fig. 3 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. 3 also shows that MSAE reaches a minimum faster for large n. Specifically, when $n = 10$, only one iteration leads to the minimum MSAE for the bunny model.

In the rest of this Section, we only discuss the case $n = 1$ since, as we have shown above, the results obtained for $n \geq 2$ do not differ significantly from those for $n = 1$; we only analyze 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. However, when 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. 4 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 the effect of whether it is preferable to include the central face normal in the computation. Fig. 5 shows variation in MSAE with n_1 when applying the variant methods with and without the central face to the double-torus model. It can be seen that using the central face yields larger MSAE at first, but produces smaller MSAE after several iterations. Thus, using the central face is to be preferred

Fifthly, we compare the results of our approach when used in either a progressive or a batch scheme. Fig. 6 shows variation in MSAE with n_1 when applying the 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.

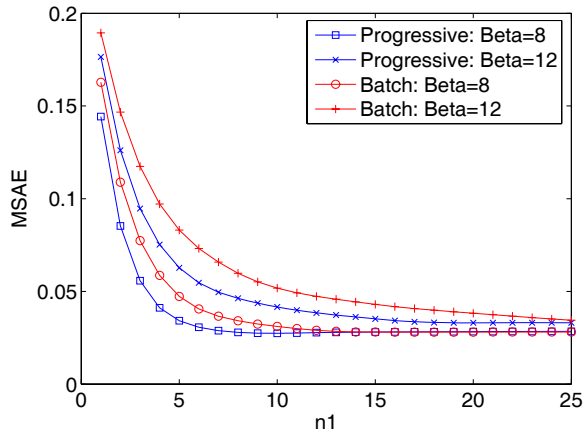

Figure 6: Progressive and batch schemes, using the double-torus model.

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

7.2 Comparisons with Other Approaches

We now turn our attention to comparing our RF approach with Yagou et al.'s [2002] MF, Fleishman et al.'s [2003] BF, and Shen and Barner's [2004] FF approaches.

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 in comparing normals.

Fig. 7 shows denoising results for a CAD-like model with sharp edges—a double-pyramid. It can be seen that all the four filtering method approaches 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 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. 8 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 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 preserve both sharp edges and the surface characteristics.

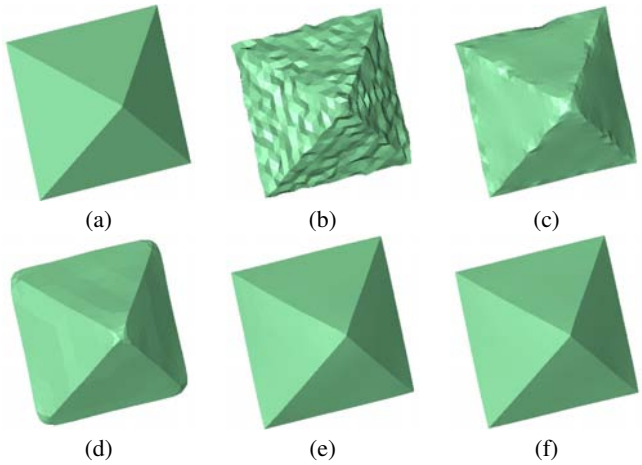

Figure 7: Denoising of a double-pyramid model ($|V| = 1026, |F| = 2048$). (a) Original model, (b) noisy model (Gaussian noise, standard deviation = 0.2 mean edge length), (c) BF result, (d) MF result, (e) FF result, (f) RF result ($n_1 = 10, \beta = 12$).

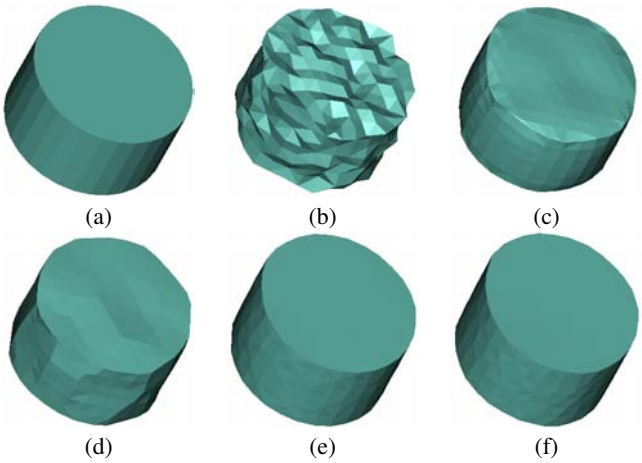

Figure 8: Denoising of a cylinder model ($|V| = 404, |F| = 804$). (a) Original model, (b) noisy model (Gaussian noise, standard deviation = 0.2 mean edge length), (c) BF result, (d) MF result, (e) FM result, (f) RF result ($n_1 = 10, \beta = 8$).

Fig. 9 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 surfaces produced by the MF approach are not particularly smooth in other areas. The FF approach and our RF approach produce smooth surfaces and preserve most sharp edges, but blur edges with small angles. The BF approach preserves sharp edges better on this model than on the previous models. The reason seems to be that the noise level in this model is relatively small, and few iterations of filtering are required. We also performed a test on the fandisk model after adding Gaussian noise with a standard deviation of 20% of the mean edge length. In this case, bilateral filtering blurs the sharp edges if we try to achieve a reasonably smooth final surface.

Fig. 10 shows denoising results on a mesh model with details at various sizes—the "iH" embossed Stanford Bunny Model. All approaches do well apart from the MF approach. The MF approach

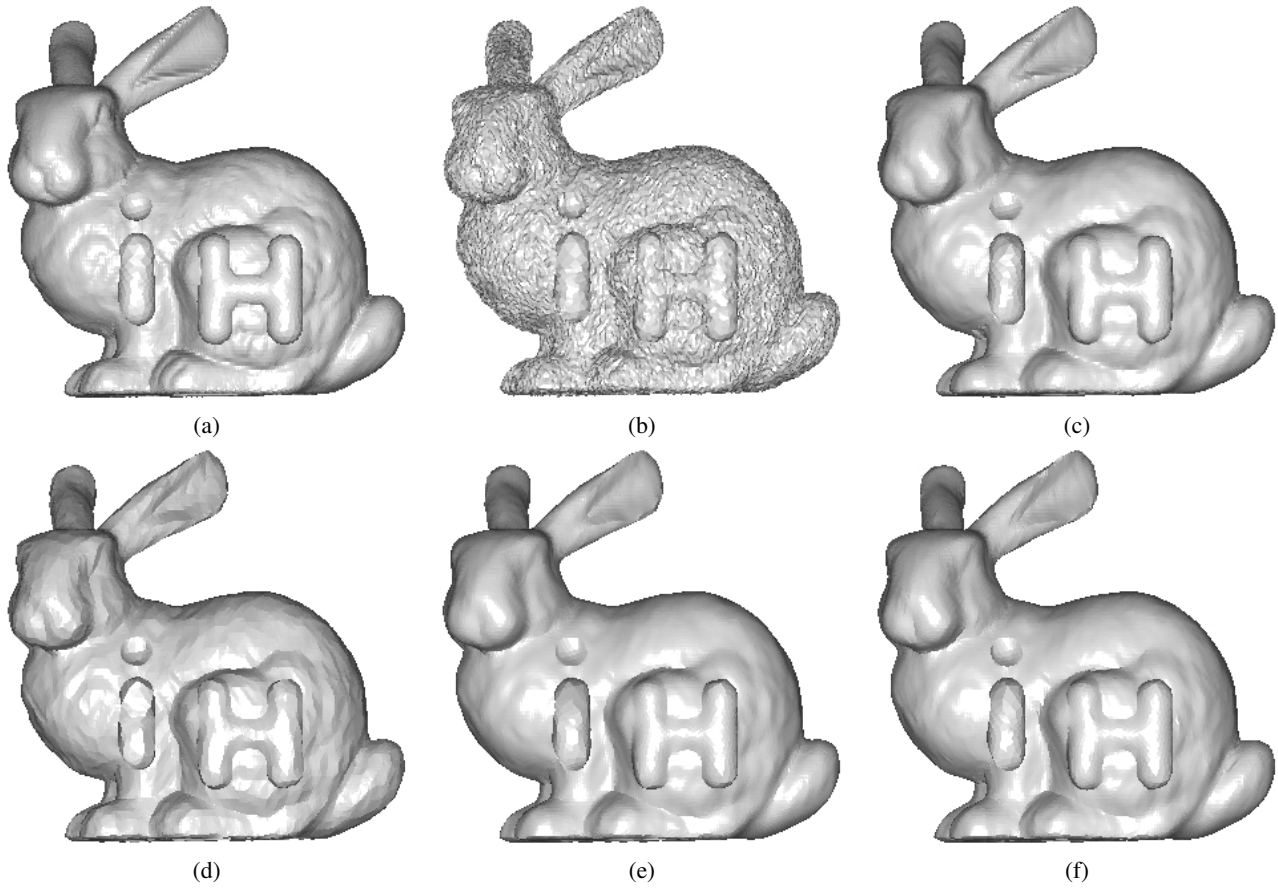

Figure 10: Denoising of an "iH" embossed Stanford Bunny model ($|V| = 34,834, |F| = 69,451$). (a) Original model, courtesy of A. Belyaev, (b) noisy model (Gaussian noise, standard deviation = 0.2 mean edge length), (c) BF result, (d) MF result, (e) FF result, (f) RF result ($n_1 = 3, \beta = 8$).

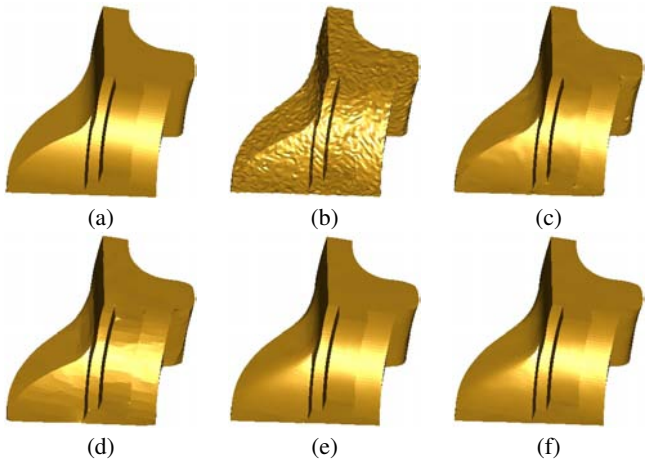

Figure 9: Denoising of a fandisk model ($|V| = 6,475, |F| = 12,946$). (a) Original model, courtesy of H. Hoppe. (b) noisy model (Gaussian noise, standard deviation = 0.1 mean edge length), (c) BF result, (d) MF result, (e) FF result, (f) RF result ($n_1 = 4, \beta = 8$).

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, with the FF approach, and to a lesser extent the RF approach, losing a little of the finer detail.

Fig. 11 shows results of denoising a scanned model with tiny details—the Maoi model. For this model, MF cannot effectively smooth the surface. FF smoothes 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.

Fig. 12 and 13 show the results of denoising two 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 both smooth the mesh surfaces while preserving tiny details to some extent. The differences between the results of these three methods are visually very small. 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.

From the above comparisons 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.

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

Figure 11: 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 ($n_1 = 3, \beta = 30$).

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

Figure 12: 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 ($n_1 = 5, \beta = 30$).

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

Figure 13: 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 ($n_1 = 3, \beta = 30$).

7.2.2 Speed

We now compare the computational cost of the approaches discussed above. Since Shen and Barner's [2004] FF approach generally produces similar results to our RF approach, we first compare these two approaches. Both approaches use a *consecutive, iterative* scheme. 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 1 shows the CPU times recorded in our experiments. Note that we have used some other large (well-known) models which 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.

We continue in Table 2 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 n_0, for the FF approach to n_1, n_2 and for our RF approach to n_1. Overall, the BF (bilateral filtering) approach is generally fastest. However, our approach requires a time similar to that of bilateral filtering; sometimes, our approach is even faster than bilateral filtering. The other approaches take significantly longer.

	Fandisk	iH Bunny	Igea	Dragon	Buddha
$\|V\|$	6475	34834	134345	437645	757490
$\|F\|$	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 1: Normal updating times for RF and FF methods (seconds, for 50 iterations)

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

Table 2: Overall times for various methods (seconds, for given numbers of iterations)

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

8 Conclusions

In this paper, we have shown how to use *random walks* for mesh denoising, and proposed a new *consecutive iterative* mesh denoising algorithm. In the first stage, the face normals are updated through weighted averaging of the face normals, with the weights being determined by probabilities of random walk steps between the current face and neighbors. 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. In the second stage, we use a conjugate gradient algorithm to solve the vertex position update least-squares problem rather than the more generally used gradient descent algorithm [Taubin 2001; Ohtake et al. 2001; Shen and Barner 2004]. The conjugate gradient algorithm is stable and converges rapidly, and is particularly suitable for solving the least-squares problem arising here as it results from 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.

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.

Acknowledgment

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

References

AZZABOU, N., PARAGIOS, N., AND GUICHARD, F. 2006. Random walks, constrained multiple hypothesis testing and image enhancement. In *ECCV 2006 Proceedings*, Springer, A. Leonardis, H. Bischof, and A. Pinz, Eds., vol. 1, 379–390.

BAJAJ, C. L., AND XU, G. 2003. Anisotropic diffusion of surfaces and functions on surfaces. *ACM Trans. Graphics 22*, 1, 4–32.

BOTSCH, M., AND KOBBELT, L. P. 2001. Resampling feature and blend regions in polygonal meshes for surface anti-aliasing. In *EG 2001 Proceedings*, Blackwell Publishing, A. Chalmers and T.-M. Rhyne, Eds., vol. 20(3), 402–410.

BOTSCH, M., BOMMES, D., AND KOBBELT, L. 2005. Efficient linear system solvers for mesh processing. In *Proceedings of the IMA Mathematics of Surfaces XI, Lecture Notes in Computer Science*, vol. 3604, 62–83.

CHEN, C.-Y., AND CHENG, H.-Y. 2005. A sharp dependent filter for mesh smoothing. *Computer Aided Geometric Design 22*, 376–391.

CLARENZ, U., DIEWALD, U., AND RUMPF, M. 2000. Anisotropic geometric diffusion in surface processing. In *Proceedings of the Conference on Visualization 2000*, IEEE Computer Society, 397–405.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of SIGGRAPH'99*, 317–324.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. 2000. Anisotropic Feature-Preserving denoising of height fields and bivariate data. In *Graphics Interface'2000 Proceedings*, 145–152.

DIEBEL, J. R., THRUN, S., AND BRÜNIG, M. 2006. A bayesian method for probable surface reconstruction and decimation. *ACM Trans. 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., AND COHEN-OR, D. 2003. Bilateral mesh denoising. *ACM Trans. Graphics 22*, 3, 950–953.

GRADY, L. 2006. Random walks for image segmentation. *IEEE Transactions Pattern Analysis and Machine Intelligence Accepted*.

HILDEBRANDT, K., AND POLTHIER, K. 2004. Anisotropic filtering of non-linear surface features. *Computer Graphics Forum 23*, 3, 391–400.

JONES, T. R., DURAND, F., AND DESBRUN, M. 2003. Non-iterative, feature-preserving mesh smoothing. *ACM Trans. Graphics 22*, 3, 943–949.

KIM, B., AND ROSSIGNAC, J. 2005. Geofilter: Geometric selection of mesh filter parameters. *Computer Graphics Forum 24*, 3, 295–302.

NEHAB, D., RUSINKIEWICZ, S., DAVIS, J., AND RAMAMOORTHI, R. 2005. Efficiently combining positions and normals for precise 3D geometry. *ACM Transactions on Graphics (Proc. SIGGRAPH) 24*, 3, 536–543.

NEHORAI, A., AND HAWKES, M. 2000. Performance bounds for estimatin vector systems. *IEEE Trans. Signal Processing 48*, 6, 1737–1749.

OHTAKE, Y., BELYAEV, A., AND BOGAEVSKI, I. 2001. Mesh regularization and adaptive smoothing. *Computer-Aided Design 33*, 11, 789–800.

OHTAKE, Y., BELYAEV, A., AND SEIDEL, H.-P. 2002. Mesh smoothing by adaptive and anisotropic gaussian filter applied to mesh normals. In *Vision, Modeling, and Visualization 2002*, 203–210.

PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., AND VETTERLING, W. T. 1992. *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, Cambridge, U.K.

SHEN, Y., AND BARNER, K. E. 2004. Fuzzy vector median-based surface smoothing. *IEEE Trans. Visualization and Computer Graphics 10*, 3, 252–265.

SHEN, J., MAXIM, B., AND AKINGBEHIN, K. 2005. Accurate correction of surface noises of polygonal meshes. *International Journal for Numerical Methods in Engineering 64*, 12, 1678–1698.

SMOLKA, B., AND WOJCIECHOWSKI, K. W. 2001. Random walk approach to image enhancement. *Signal Processing 81*, 3, 465–482.

SPITZER, F. 2001. *Principles of Random Walk*, 2nd ed. Springer, New York.

SZCZEPANSKI, M., SMOLKA, B., PLATANIOTIS, K. N., AND 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., AND OSHER, S. 2002. Geometric surface smoothing via anisotropic diffusion of normals. In *Proceedings of the Conference on Visualization 2002*, IEEE Computer Society, 125–132.

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

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

VOLLMER, J., MENCL, R., AND MÜLLER, H. 1999. Improved laplacian smoothing of noisy surface meshes. *Computer Graphics Forum 18*, 3, 131–138.

WECHSLER, H., AND KIDODE, M. 1979. A random walk procedure for texture discrimination. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-1*, 3, 272–280.

YAGOU, H., OHTAKE, Y., AND BELYAEV, A. G. 2002. Mesh smoothing via mean and median filtering applied to face normals. In *Proceedings of Geometric Modeling and Processing*, 124–131.

YAGOU, H., OHTAKE, Y., AND 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)*, 28–34.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H. 2004. Mesh editing with poisson-based gradient field manipulation. *ACM Transactions on Graphics 23*, 3, 644–651.