Journal of Zhejiang University SCIENCE A
ISSN 1673-565X (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Mesh sharpening via normal filtering*

Jian-guo SHEN[†1], San-yuan ZHANG[†‡1], Zhi-yang CHEN[2], Yin ZHANG[1], Xiu-zi YE[1]

([1]*State Key Laboratory of CAD & CG, Zhejiang University, Hangzhou 310027, China*)

([2]*Software College, Zhejiang University of Technology, Hangzhou 310014, China*)

[†]E-mail: shenjg@163.com; syzhang@cs.zju.edu.cn

Received July 2, 2008; Revision accepted Aug. 7, 2008; Crosschecked Feb. 16, 2009

Abstract: We present a robust mesh sharpening approach to reconstructing sharp features from blended or chamfered features, even with noise and aliasing errors. Feature regions were first recognized via normal variation according to the user's input, and then normal filtering was applied to faces of feature regions. Finally, the vertices of the feature region were gradually updated based on new face normals using a least-squares error criterion. Experimental results demonstrate that the method is effective and robust in sharpening meshes.

Key words: Normal filtering, Sharp feature, Mesh sharpening, Blend, Chamfer
doi:10.1631/jzus.A0820505 **Document code:** A **CLC number:** TP391.7

INTRODUCTION

Triangle meshes are used in a wide array of fields such as reverse engineering, computer vision, virtual reality, and terrain modeling. They can be acquired with a 3D scanner, extracted from volume datasets or generated via modeling. As important parts of a mesh model, sharp features play a key role in mesh processing such as reconstruction, remeshing, denoising, and simplification. There is a growing demand for sharpening mesh in the field of mesh processing.

Sharp features are often blurred, distorted or lost due to noise and aliasing errors [produced by remeshing or reconstruction approaches, e.g., Marching Cube (Lorensen and Cline, 1987)], and this generally leads to quite bad approximation behaviors. Over-sampling could decrease the aliasing error, but the associated aliasing error shall not be totally eliminated, as observed by Kobbelt *et al*.(2001), since the surface normals on the reconstructed model usually do not converge to the normal field of the original object. To all denoising algorithms, it is a main challenge to preserve sharp features while removing noise because there is no simple way to distinguish features from noise. In mesh editing, reconstruction of sharp features is effective in removing blends and chamfers. Furthermore, the sharp feature is an important issue in simplification (Hussain *et al*., 2004), segmentation (Lavoué *et al*., 2005), hole filling (Chen and Cheng, 2008), etc. A robust approach is therefore needed to reconstruct sharp features for mesh processing.

Reconstructing sharp features can be called 'sharpening' for short. In this paper we propose a sharpening method, which deals with not only blended and chamfered features, but also features with noise and aliasing errors.

The rest of this paper is organized as follows. In Section 2, related work is reviewed. In section 3, some basic notations used in this paper are introduced and an overview of our algorithm is given. The detection of feature regions is discussed in Section 4. In Section 5, details of the normal filtering procedure and ways of improving the sharpening quality are

‡ Corresponding author

* Project supported by the Hi-Tech Research and Development Program (863) of China (Nos. 2007AA01Z311 and 2007AA04Z1A5), the Doctoral Fund of MOE of China (No. 20060335114), and the Science and Technology Program of Zhejiang Province, China (No. 2007C21006)

discussed. In Section 6 we explain how to update vertex positions and avoid face flipping. The results obtained by applying the proposed method are shown in Section 7. In Section 8, conclusions of the paper and future work are discussed.

RELATED WORK

First, we briefly recall some denoising approaches that have close relationship with sharpening. Shen *et al*.(2005) proposed a convergent denoising method. Its basic idea is to apply different smoothing algorithms to the feature and non-feature regions. The algorithm is slow, however, because of extra costs in pre-smoothing and partitioning. Another related technique is called bilateral filtering, which was originally presented for image processing (Tomasi and Manduchi, 1998). Fleishman *et al*.(2003) iteratively applied the bilateral filter to mesh denoising, which nevertheless often drifted the vertex and did not preserve fine features. A non-iterative approach was proposed by Jones *et al*.(2003), but much slower than the convergent denoising method because their approach treated normal smoothing and vertex updating as global problems. Feature enhancement was added into the smooth process by Clarenz *et al*.(2000). Sun *et al*.(2007) proposed a fast and effective feature-preserving denoising approach. Two stages, i.e., face normal filtering and vertex position updating, constitute the method and are both iterative. A very similar idea is found in (Nie *et al*., 2004), which applied the Perona-Malik method (Perona and Malik, 1990) in normal filtering. Mesh sharpening is similar to mesh denoising; however, there is one distinction between them: in mesh denoising the aim is to remove noise while preserving or enhancing existing sharp features, whereas in mesh sharpening, the aim is to reconstruct sharp features from arbitrary features including blend and chamfer.

In recent years, a wide variety of mesh sharpening algorithms have been proposed. Chen and Cheng (2008) applied a sharpness-dependent filtering algorithm to recover sharp features in the repaired mesh. Attene *et al*.(2003) presented a sharpening approach, assuming that chamfered edges contain no sample points, as in the case of reconstructing by using the Marching Cubes algorithm (Lorensen and Cline, 1987). Some other approaches for the similar purpose (Wang, 2006a; 2006b), without relying on Attene *et al*.(2003)'s assumption, assume instead that the non-feature region does not contain noise or aliasing errors.

Motivated by the impressive results of normal filtering (Sun *et al*., 2007) for mesh denoising, we adopt normal filtering to govern the sharpening of manifold triangle meshes. In contrast to other methods for mesh sharpening, our approach is more robust and simpler to implement.

OVERVIEW

We first define some notations used in this paper. A two-manifold triangle mesh is denoted by $M=(V, E, F, X)$, where $V=\{i|i=1, 2, \ldots, n\}$ is the vertex set, $E=\{e|e=(i, j), i, j\in V\}$ is the edge set, $F=\{f|f=(i, j, k), i, j, k\in V\}$ is the face set, and $X=\{x_i|x_i\in\mathbb{R}^3, i\in V\}$ is the vertex coordinate set. For any triangle f_i in mesh, $\boldsymbol{n}_i$ is its normal and c_i is its center. We use '' to represent the updated value, relative to the current value. The 1-ring face neighborhood of a vertex v_i, denoted by $N_V(i)$, is the set of faces that are connected to v_i by edges. The set of faces that share a common edge with f_i is denoted by $N_{FI}(i)$. We use '|·|' to represent the cardinality of a set.

In basic terms, mesh sharpening can be regarded as a requirement to adjust vertex positions. The idea behind our approach is to evaluate normals of a feature region with those of a smooth region and then to gradually update vertex positions based on new normals. The method consists of three main stages: (1) Detect the feature region that needs processing through normal variations; (2) Filter the face normals progressively from outer to inner of the sharp regions; (3) Update vertex positions iteratively to agree with the adjusted face normals.

Our approach targets two-manifold triangle meshes, assuming that noise and errors do not exist on the non-sharp-feature region. The main contributions include:

(1) In addition to processing chamfered and blended features, our approach processes features with noise and aliasing errors.

(2) The face normal filtering method is modified to reconstruct sharp features.

(3) Face flipping is avoided through optimizing in position updating.

FEATURE DETECTION

There are many methods to detect sharp features. The simplest one is through the dihedral angle on an edge, which is called second-order difference (SOD) (Guskov *et al.*, 1999). Extended second-order difference (ESOD) was proposed by Hubeli *et al.*(2000) for the identification of features. However, neither SOD nor ESOD is robust enough to recognize the feature where the normals vary smoothly. Another method for identifying sharp features is to analyze the curvature tensor (Dong and Wang, 2005); but just as Wang (2006a) suggests, it is difficult to detect chamfer features using curvature.

Wang (2006a) presented a uniformly supported second-order difference (USSOD) method instead, successfully recognizing features through measuring maximal normal variation around a vertex. The USSOD method, however, fails to distinguish the correct sharp features (i.e., sharp features with no noise or aliasing errors) from the features that need sharpening.

We extend the USSOD method, in which the set of neighboring faces around f_i, denoted by $N_{FII}(i)$, is first generated by the following steps:

Step 1: Add i to the queue Q.

Step 2: Fetch a face s from Q and add it to $N_{FII}(i)$.

Step 3: If $d(c_i, f_k)<\rho$ and $\boldsymbol{n}_s\cdot\boldsymbol{n}_k>\lambda$ where $k\in N_{FI}(s)$ and $k\notin N_{FII}(i)$, add k to Q.

Step 4: Exit if Q is empty; otherwise, go to Step 2.

In $d(c_i, f_k)<\rho$, ρ is a threshold specified by the user, analogous to the threshold in USSOD and usually a little larger than the maximal width of the feature to be sharpened. $d(c_i, f_k)$ represents the minimal distance between c_i and f_k. For efficiency, $d(c_i, f_k)$ simply takes the minimal Euclidean distance from c_i to each vertex of f_k and c_k. $\boldsymbol{n}_s\cdot\boldsymbol{n}_k>\lambda$ indicates a sharp edge, where λ ranges from -1 to 1 and is determined by the user. Note that λ is set to zero in all our experiments. The effect of this condition is to stop searching if there is a sharp edge in that direction and to make the neighborhood more reasonable (Fig.1). Furthermore, it is the critical condition for recognizing the correct sharp features. Note that $N_{FII}(i)$ includes i.

After determining the neighborhood of f_i, we use the formula below to measure the maximum variation of face normal around f_i:

$$S(i)=\min_{j,k\in N_{FII}(i)}\{\boldsymbol{n}_j\cdot\boldsymbol{n}_k\}.$$

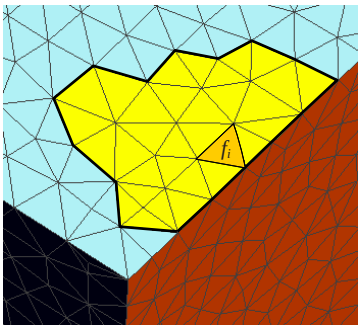


Fig.1 Neighborhoods (covered with bold edges) of the face f_i (λ=0), which is near a sharp edge

f_i is a feature face if $S(i)<\eta$, where η ranges from 0 to 1 and is determined by the user; otherwise, f_i is a non-feature face. When the feature is relatively sharp, a smaller value of η is appropriate; when the feature is relatively smooth, η should be larger. The set of the feature faces is denoted by F^{Ft}.

v_i is a feature vertex if there is one feature face in $N_V(i)$. The set of the feature vertices is indicated by V^{Ft}. v_i is an absolute feature vertex if all faces in $N_V(i)$ are features. The set of the absolute feature vertices is denoted by F^{AFt}.

The thresholds ρ and η are necessary in mesh sharpening as feature detection is heavily mesh and user dependent. In practice, they can be interactively assigned in a method similar to that in (Fleishman *et al.*, 2003): the user selects a region to be sharpened. The radius of the region is assigned to ρ and the minimal inner dot of two face normals in this region is assigned to η.

As compared with USSOD, the main advantage of our approach is that we can avoid processing the correct sharp features, which renders the algorithm more efficient and robust. In Fig.1, although the face f_i is near a sharp edge, $S(i)$ is close to 1 since all neighborhoods almost share the same plane. Besides, the detection is based on faces instead of vertices for simplicity.

NORMAL FILTERING

In this section, the normal filtering is to be described in detail and two auxiliary operations, pre-smoothing and normal rectifying, are to be added to improve its robustness.

Pre-smoothing

Before normal filtering, pre-smoothing is applied to the absolute feature vertices on account of its three functions: removing noises, blending chamfer, and improving mesh quality. In principle any curve smoothing scheme could be used here. We simply choose the bilaplacian operator (Kobbelt *et al*., 1998) to smooth the absolute feature vertices for simplicity and efficiency. The number of iterations is $\lceil 4L/\rho \rceil$, where L is the average edge length. In Fig.2 the effect of the pre-smoothing is shown.

Normal filtering

Normal filtering is the core of our algorithm. In our greedy procedure for normal filtering, a priority queue is created in which all feature faces are sorted based on the cost of filtering the normal. At each step, the face that has the lowest cost is selected prior to the circulation of its new normal and then removed from F^{Ft}. After that, the cost of all the adjacent feature faces is re-computed and their position in the queue is updated according to the new cost. This loop is iterated until the priority queue is empty. Fig.3 shows the procedure of the normal filtering.

We describe the above procedure in more detail. The cost of filtering $\boldsymbol{n}_i$ is defined as

$$C(i)=\min_{j\in N_{FI}(i)}\{w(i,j)\},$$

where $w(i, j)$ is a weight function defined as

$$w(i,j)=\begin{cases}1-\boldsymbol{n}_i\cdot\boldsymbol{n}_j, & j\notin F^{\mathrm{Ft}},\\ 2, & j\in F^{\mathrm{Ft}}.\end{cases}$$

The face corresponding to $C(i)$ is regarded as the face most similar to f_i.

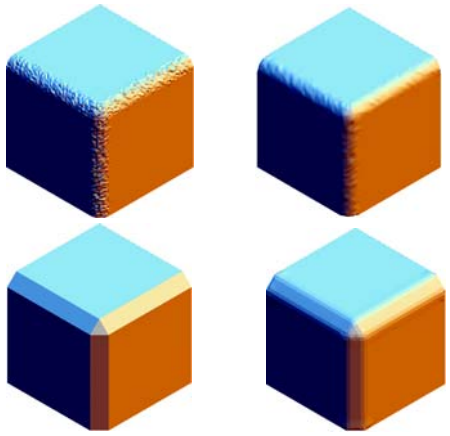
We modify the denoising filter in (Sun *et al*., 2007) to suit mesh sharpening based on the concept that the normal of the feature face tends toward similar normals of neighboring non-feature faces. Therefore, the new normal is calculate by

$$\boldsymbol{n}_i=\mathrm{normalize}\left(\sum h_k\boldsymbol{n}_k\right),$$

where h_k is a weight function defined as

$$h_k=\begin{cases}g(\boldsymbol{n}_k\cdot\boldsymbol{n}_m-\eta), & \boldsymbol{n}_k\cdot\boldsymbol{n}_m>\eta,\\ 0, & \boldsymbol{n}_k\cdot\boldsymbol{n}_m\le\eta,\end{cases}$$
$$k\in N_{FII}(m)\text{ and }k\notin F^{\mathrm{Ft}}.$$

f_m is the face most similar to f_i (Fig.4), which is different from (Sun *et al*., 2007). The motivation behind the weight function h_k is to give high weights to those face normals close to $\boldsymbol{n}_m$ and low weights to those far from $\boldsymbol{n}_m$. We also choose $g(x)=x^2$ (Sun *et al*., 2007).



(a) (b)

Fig.2 Smoothing absolute feature vertices

(a) Mesh before smoothing; (b) Mesh after smoothing
Top: noise removing; Bottom: chamfer blending

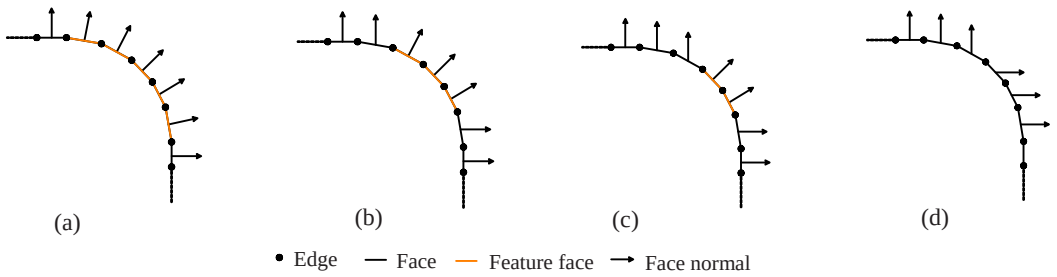


Fig.3 The procedure of normal filtering

(a) Original normals and feature faces; (b) Normals and feature faces after filtering two normals; (c) Normals and feature faces after filtering three normals; (d) The result of normal filtering

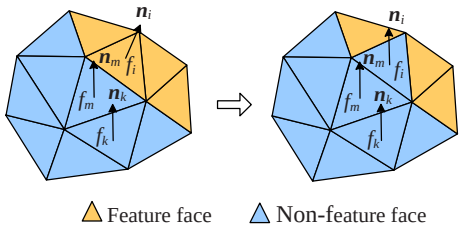


Fig.4 Filtering the normal of f_i

The main advantage of our scheme is that, unlike previous work (Sun *et al*., 2007; Chen and Cheng, 2008), the procedure of normal filtering is non-iterative. A greedy procedure for normal filtering is used to filter normals progressively from outer to inner of the sharp regions.

Normal rectifying

Due to noise and irregularity, some unreasonable normals (Fig.5b) may still exist after normal filtering. These normals will affect the next operation, leading to poor results (Fig.6), and therefore should be rectified.

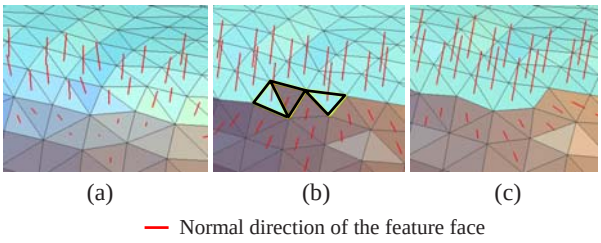


Fig.5 Normal rectifying

(a) Original normals; (b) Normals after filtering (faces with unreasonable normals are highlighted with bold edges); (c) Normals after rectifying

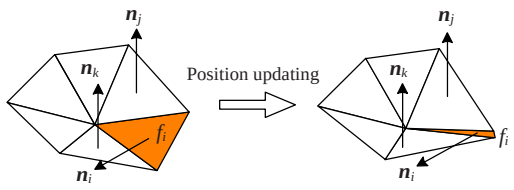


Fig.6 The face f_i with an incorrect normal becomes degenerate during position updating

The procedure of normal rectifying is very similar to that of normal filtering. f_i needs rectifying if (1) the normal of f_i is adjusted, and (2) $\boldsymbol{n}_j$ is almost the same as $\boldsymbol{n}_k$, $\boldsymbol{n}_i \cdot \boldsymbol{n}_j < \eta$ and $\boldsymbol{n}_i \cdot \boldsymbol{n}_k < \eta$, where $j, k \in N_{FI}(i)$, $j \neq k$ (Fig.6). We put those faces in the priority queue. The cost is evaluated by $1-\boldsymbol{n}_i \cdot \boldsymbol{n}_j$. f_j is regarded as the face most similar to f_i. At each step, the face that has the lowest cost is selected and its normal is assigned to the normal of the most similar face. The remaining procedure is the same as that of normal filtering. The effect of normal rectifying is shown in Fig.5c.

POSITION UPDATING

Vertex position updating is based on the integration of face normals using a least-squares error criterion. At each iteration, each vertex is moved to a new position and its adjacent face normals are updated to agree with the filtered face normals. Following the suggestions of (Sun *et al*., 2007), we perform position updating by

$$x_i' = x_i + \frac{1}{|N_V(i)|} \sum_{k \in N_V(i)} \boldsymbol{n}_k (\boldsymbol{n}_k \cdot (c_k - x_i)),$$

where $\boldsymbol{n}_k$ is the new normal of f_k after normal filtering.

The vertex updating algorithm is convergent, able to terminate the vertex updating process before the maximum iterations when the required precision is reached.

The position updating, though simple and efficient, has a problem. If the result of normal filtering is not very good, the vertex updating step may result in non-optimal positions and cause face flipping. In some situations, even if the result is right (Fig.7a), face flipping still occurs (Fig.7b). Chen and Cheng (2008) used a step length parameter γ to determine the speed for moving the vertex to a new position and to avoid flipping the faces. But it is unsuitable for our purpose since the vertex position does not agree with the filtered normals (Fig.7c) and there will be some degenerate faces in the resulting mesh. Hence, an optimization operation was applied to deal with these problems. For each position updating, we used the same method as that in (Chen and Cheng, 2008) to avoid flipping. After each iteration, we checked if the faces adjacent to the feature vertices are degenerate. If it is degenerate, the face was classified as a cap or a needle (Botsch and Kobbelt, 2001). We eliminated caps by swapping the longest edge (Fig.8a), while removed needles by simply collapsing the shortest edge (Fig.8a). Before the operation is applied, it must be ensured that no flipping occurs. A significant improvement was gained using these two operators (Fig.7d).

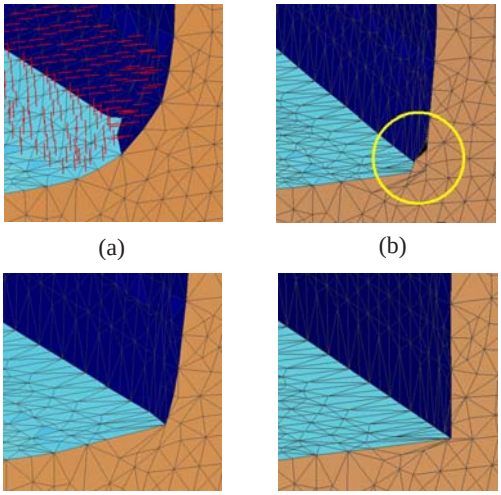


Fig.7 Vertex position updating
(a) After normal filtering; (b) Sun *et al.*(2007)'s result; (c) Chen and Cheng (2008)'s result; (d) Our result

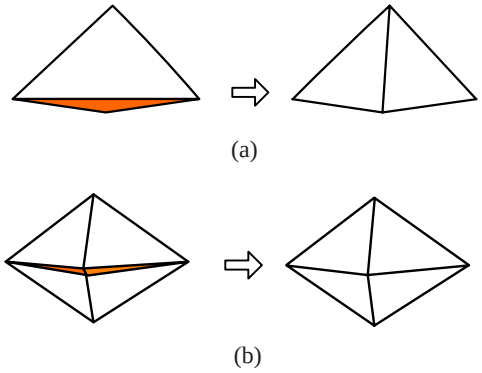


Fig.8 Eliminating degenerate faces
(a) Removing caps by swapping edge; (b) Removing needles by collapsing edge

RESULTS AND DISCUSSION

The algorithm introduced in this paper was implemented in VC 2005 and tested on a PC with a 2.8 GHz Intel Pentium IV CPU and 2 GB RAM. Our approach handles various features robustly. The examples in Fig.9 are conducted to demonstrate the ability of our approach in sharpening different features, which are blended, chamfered and embedded with noises respectively. The example in Fig.10 shows the impact of choosing parameter ρ. The radii of the blended features that are sharpened increase with a larger ρ. Some comparisons of our approach with those of (Sun *et al*., 2007; Chen and Cheng, 2008) are given in Figs.11~13. Our algorithm is aimed at reconstructing sharp features from other features, whereas the approach in (Sun *et al*., 2007) is aimed to remove noise and the one in (Chen and Cheng, 2008) is aimed to reconstruct features from existing features, which results in the different performances as shown by Figs.11~13, especially for the gear model in Fig.12. The experiments demonstrate that our proposed method works well for different models.

Execution time recorded in our experiments is given in Table 1. The time consuming steps are the feature detection and the position updating, especially when ρ is large. The results show that normal filtering is fairly efficient.

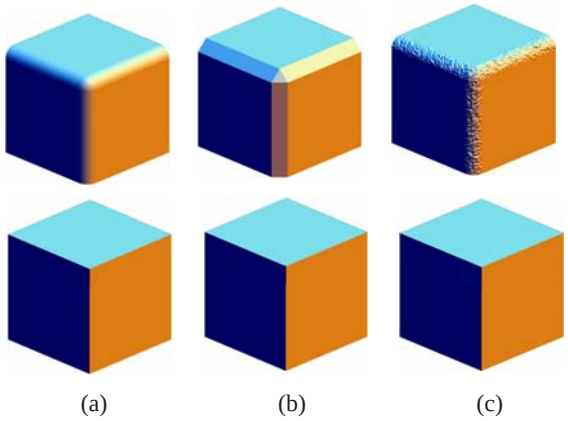


Fig.9 Sharpening three different kinds of features
(a) Blended feature; (b) Chamfered feature; (c) Feature with noises. Top: original meshes; Bottom: sharpened meshes

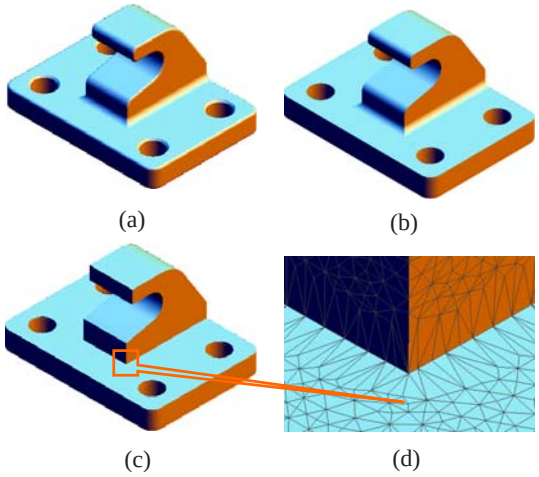
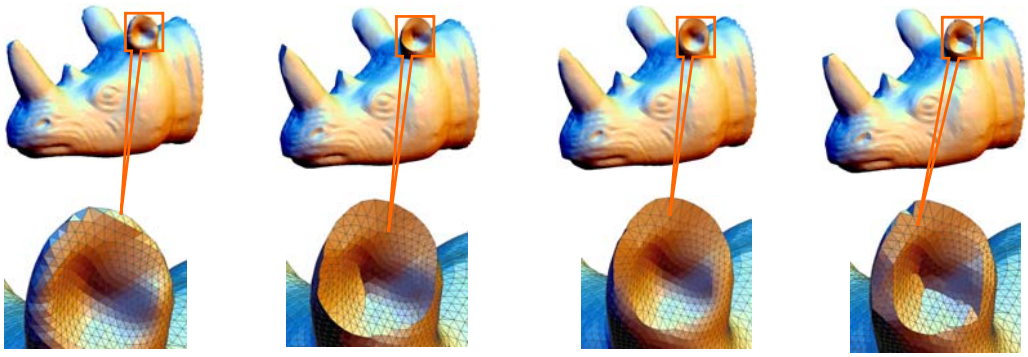


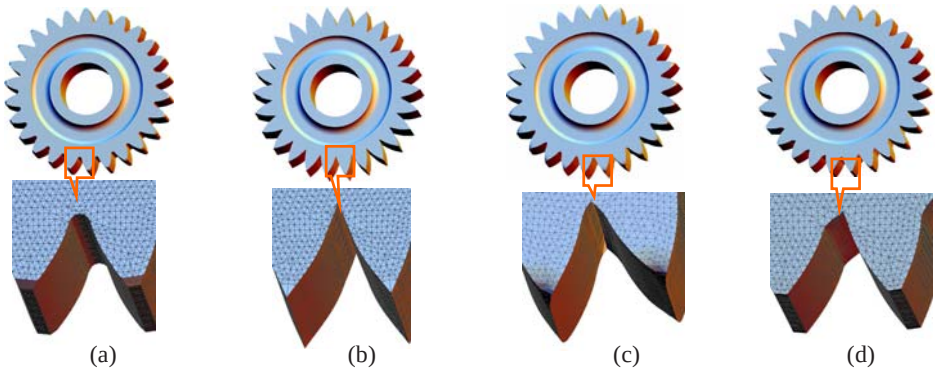
Fig.10 Anchor model
(a) Original mesh; (b) Mesh sharpened with a small ρ; (c) Mesh sharpened with a large ρ; (d) Local detail of (c)



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

Fig.11 Rhinoceros model

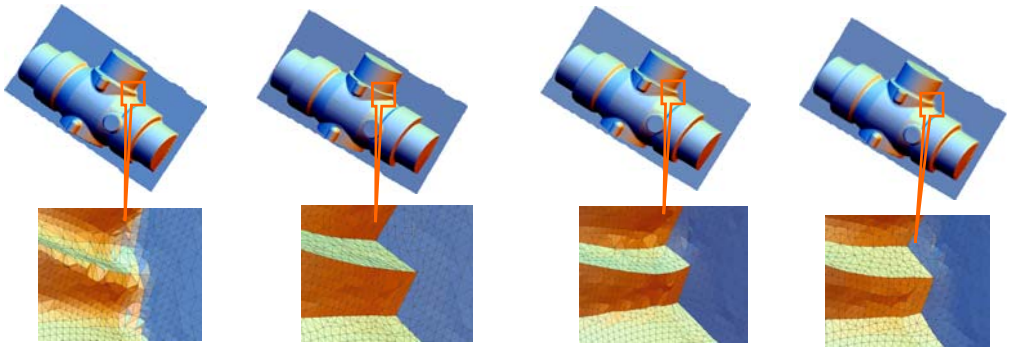
(a) Original mesh; (b) Our result; (c) Sun *et al*.(2007)'s result; (d) Chen and Cheng (2008)'s result



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

Fig.12 Gear model

(a) Original mesh; (b) Our result; (c) Sun *et al*.(2007)'s result; (d) Chen and Cheng (2008)'s result



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

Fig.13 Valve model

(a) Original mesh; (b) Our result; (c) Sun *et al*.(2007)'s result; (d) Chen and Cheng (2008)'s result

Table 1 Execution time of the illustrated examples (λ=0)

Model	Face number	ρ/L	η	Time (s)		
				Feature detecting	Normal filtering	Position updating
Blend	13 012	4.5	0.1	2.22	0.67	1.18
Chamfer	10 244	4.6	0.1	1.49	0.13	0.46
Noisy feature	13 325	3.0	0.1	1.43	0.55	1.71
Anchor (Fig.10b)	180 211	1.0	0.8	3.69	0.92	3.09
Anchor (Fig.10c)	180 211	3.0	0.8	12.74	2.89	8.34
Rhinoceros	23 509	1.0	0.5	0.67	0.02	0.11
Gear	160 248	2.5	0.0	13.95	1.84	17.58
Valve	93 215	1.2	0.5	2.55	0.96	7.20

CONCLUSION

In this paper, we present a robust mesh sharpening method, which can recover sharp features such as sharp edge and corners effectively. The approach is based on normal filtering, making the algorithm simple to implement. Experimental results show that our method works robustly on various meshes and particularly on optimized CAD meshes. The quality of sharpening is satisfactory, and the features to be sharpened can be governed by the user-specified parameters.

The main bottleneck of our current approach is the feature detection stage in which the computation cost is high. This is a disadvantage for interactive applications where a user may try different parameters to obtain the desired features. Thus a faster feature recognition method needs to be investigated in future research.

References

Attene, M., Falcidieno, B., Rossignac, J., Spagnuolo, M., 2003. Edge-sharpener: Recovering Sharp Features in Triangulations of Non-adaptively Re-meshed Surfaces. Proc. Eurographics Symp. on Geometry Processing, p.62-71.

Botsch, M., Kobbelt, L.P., 2001. A Robust Procedure to Eliminate Degenerate Faces from Triangle Meshes. Proc. Vision, Modeling and Visualization, p.283-290.

Chen, C.Y., Cheng, K.Y., 2008. A sharpness-dependent filter for recovering sharp features in repaired 3D mesh models. *IEEE Trans. Visual. Comput. Graph.*, **14**(1):200-212. [doi:10.1109/TVCG.2007.70625]

Clarenz, U., Diewald, U., Rumpf, M., 2000. Anisotropic Geometric Diffusion in Surface Processing. Proc. IEEE Visualization, p.397-405. [doi:10.1109/VISUAL.2000.885721]

Dong, C.S., Wang, G.Z., 2005. Curvatures estimation on triangular mesh. *J. Zhejiang Univ. Sci.*, **6A**(Suppl. I): 128-136. [doi:10.1631/jzus.2005.AS0128]

Fleishman, S., Drori, I., Cohen-Or, D., 2003. Bilateral mesh denoising. *ACM Trans. Graph.*, **22**(3):950-953. [doi:10.1145/882262.882368]

Guskov, I., Sweldens, W., Schröder, P., 1999. Multiresolution Signal Processing for Meshes. Proc. 26th Annual Conf. on Computer Graphics and Interactive Techniques, p.325-334. [doi:10.1145/311535.311577]

Hubeli, A., Meyer, K., Gross, M.H., 2000. Mesh Edge Detection. Technical Report, CS #351. ETH, Zürich.

Hussain, M., Okada, Y., Niijima, K., 2004. Efficient and feature-preserving triangular mesh decimation. *J. WSCG*, **12**(1):167-174.

Jones, T.R., Durand, F., Desbrun, M., 2003. Non-iterative, feature preserving mesh smoothing. *ACM Trans. Graph.*, **22**(3):943-949. [doi:10.1145/1201775.882367]

Kobbelt, L.P., Campagna, S., Vorsatz, J., Seidel, H.P., 1998. Interactive Multi-resolution Modeling on Arbitrary Meshes. Proc. 25th Annual Conf. on Computer Graphics and Interactive Techniques, p.105-114. [doi:10.1145/280814.280831]

Kobbelt, L.P., Botsch, M., Schwanecke, U., Seidel, H.P., 2001. Feature Sensitive Surface Extraction from Volume Data. Proc. 28th Annual Conf. on Computer Graphics and Interactive Techniques, p.57-66. [doi:10.1145/383259.383265]

Lavoué, G., Dupont, F., Baskurt, A., 2005. A new CAD mesh segmentation method, based on curvature tensor analysis. *Computer-Aided Design*, **37**(10):975-987. [doi:10.1016/j.cad.2004.09.001]

Lorensen, W.E., Cline, E., 1987. Marching cubes: a high resolution 3D surface construction algorithm. *ACM SIGGRAPH Comput. Graph.*, **21**(4):163-169. [doi:10.1145/37402.37422]

Nie, J.H., Zhou, L.S., Liu, S.L., 2004. A mesh smoothing algorithm for feature enhancing. *Mech. Sci. Technol.*, **23**(1):110-112 (in Chinese).

Perona, P., Malik, J., 1990. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell.*, **12**(7):629-638. [doi:10.1109/34.56205]

Shen, J., Maxim, B., Akingbehin, K., 2005. Accurate correction of surface noises of polygonal meshes. *Int. J. Numer. Methods Eng.*, **64**(12):1678-1698. [doi:10.1002/nme.1441]

Sun, X.F., Rosin, P.L., Martin, R.R., Langbein, F.C., 2007. Fast and effective feature-preserving mesh denoising. *IEEE Trans. Visual. Comput. Graph.*, **13**(5):925-938. [doi:10.1109/TVCG.2007.1065]

Tomasi, C., Manduchi, R., 1998. Bilateral Filtering for Gray and Color Images. Proc. 6th Int. Conf. on Computer Vision, p.839-846.

Wang, C.C.L., 2006a. Incremental reconstruction of sharp edges on mesh surfaces. *Computer-Aided Design*, **38**(6): 689-702. [doi:10.1016/j.cad.2006.02.009]

Wang, C.C.L., 2006b. Bilateral recovering of sharp edges on feature-insensitive sampled meshes. *IEEE Trans. Visual. Comput. Graph.*, **12**(4):629-639. [doi:10.1109/TVCG.2006.60]