Ridge-Valley Lines Smoothing and Optimizing

Hao Jing, Weixiang Zhang, and Bingfeng Zhou

Institute of Computer Science and Technology, Peking University, P.R. China
jinghao@icst.pku.edu.cn,
zhangweixiang@icst.pku.edu.cn,
cczbf@pku.edu.cn
http://sun1000e.pku.edu.cn/ cczbf/VITA.htm

Abstract. When detecting ridge-valley lines on 3D mesh model, estimation of the curvature and curvature derivatives may often yields to squiggly and noisy result, because the estimation is sensitive against unwanted surface noises. We present two algorithms to obtain smooth and noiseless ridge-valley lines. First, we apply an iterative procedure on ridge and valley vertices and their previous and next neighbors on connected feature lines, which leads to smooth lines. Secondly, we propose an algorithm to distinguish noises from meaningful feature lines based on graph theory model. Each separate feature line is considered as an undirected weighted graph which is called as **Feature Graph**. We can reasonably get rid of most noises and preserve meaningful feature lines through optimizing the minimal spanning tree of each feature graph.

1 Introduction

Lines are powerful shape descriptors which can convey most information on 3D models for designer and artist. Feature lines can be remarkably efficient at conveying shape and meaning while reducing visual clutter, especially in the interactive design of entertainment and engineering system. Feature lines extraction from discrete meshes has become an hot area of intense research in recent decade [1] [2] [3] [4] [5]. In general, feature lines include **Silhouette Edges**, **Boundary Edges** and **Interior Feature Edges** [6]. Boundary edges only exist in non-closed surface meshes. Detection of boundary edges is simple. we just detect edges contained in only one triangle on triangle model. The fast extraction algorithms of silhouette edges have been developed and can achieve good results. [1] [2] [3] [4].

Besides silhouette edges and boundary edges, interior feature edges indicate the internal structure and details on mesh at a finer level. In our method, we mainly deal with interior feature edges as view- and scale- independent ridge-valley lines, which are curves on a surface along which the surface bends sharply. However, it is easy to apply our algorithm to dealing with view-dependent feature lines such as suggestive lines [7] [8].

There have been various existing interior feature edges detection algorithms [9] [10] [11] [12] [7] [8]. All existing feature line extraction methods, however,

Z. Pan et al. (Eds.): ICAT 2006, LNCS 4282, pp. 502–511, 2006.

treated the noises and meaningful feature lines of surface as the same role in computing process, the algorithm we present can reasonably separate noises from meaningful lines.

In our study, we first detect ridge and valley vertices via curvature and curvature derivatives analysis, then connect those vertices to generate feature lines. This lead to two problems as the left image of Figure 1 shows. One is squiggly lines result, the other is noises produced because of the computational error of curvature and curvature derivatives estimation. These two problems all come from the curvature-related estimation.

We present two algorithms to resolve these two problems respectively. After obtaining the ridge and valley vertices on mesh surface and connecting them, first we apply an iterative procedure working on local previous and next neighbors of a vertex on connected feature line. This iteration operation which is similar to Laplacian smoothing will achieve smooth feature lines. Secondly, we visit all ridge and valley vertices. Naturally, a separate feature line can be treated as an undirected graph weighted by the length of edges. The extracted feature lines can be viewed as a set of undirected weighted graphs. We call these graph **Feature Graphs** in our paper. For each feature graphs, a root node is chosen by a special condition, then the minimal spanning tree and the longest path of the tree is computed. After optimizing the minimal spanning tree through weight of nodes on graph, we can wipe out noises, and obtain meaningful feature lines as the right image of Figure 1 shows.

The contribution of our paper focus on not only the smoothing procedure of feature lines, but also providing a reasonable and extendable scheme on how to divide noise from meaningful feature lines. In the remaining of this paper, we will introduce previous work in Section 2. In Section 3, the ridge-valley lines smoothing method will be given. Section 4 will describe the optimizing algorithm used to wipe out noises.

2 Previous Work

Various feature detection algorithms have been proposed during the past several years [9] [10] [11] [12] [7] [13] [8]. Earlier papers focus on extraction of view-dependent silhouette edges [2] [3] [4], which separate the visible part from invisible ones of mesh model, i.e. edges shared by front-facing and back-facing polygons, and can be detected fast in real time [1].

However, much more finer details are indicated by interior feature edges. Most of existing algorithms on extracting interior feature edges mainly focused on global feature detection via curvature analysis. DeCarlo [7] define interior feature edges as view-dependent suggestive contour, which is extracted through radial curvature [14] analysis. The authors improved their algorithm in [13] with increasing of detecting speed. Sousa and Prusinkiewicz [8] present another automated algorithm to produce suggestive line drawing using graph theory, but

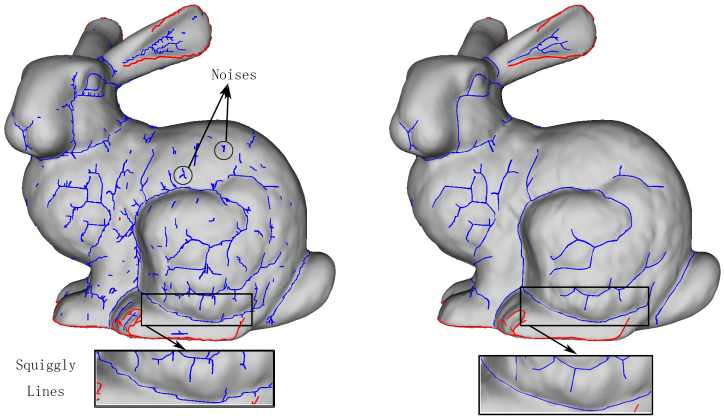

Fig. 1. A feature lines smoothing and optimizing example of bunny model with 69k triangles. The left image is the result before line smoothing and optimizing [10], noises and squiggly lines can be seen. The right image is the result after optimizing, feature lines are smoother and noiseless.

their results are not satisfied when the method is applied on some model, such as the bunny model. Other authors define interior feature edges as view- and scale-independent Ridge-Valley edges [9] [11], and extract feature lines via principle curvature analysis. The benefit of using view- and scale-independent ridge-valley lines is that they only need to be computed once, and are not necessary to be recomputed each time the view point changed. In our study, we focus on ridge-valley lines and mainly follow the definition in [10].

Ohtake [10] proposed simple and effective method for detecting ridge-valley lines defined via first- and second-order curvature derivatives on meshes. The high order surface derivative is achieved by combing multi-level implicit surface fitting. The common drawbacks of these curvature based algorithms are related with the sensitiveness of both curvature and derivative estimations against unwanted surface noise, and they do not make any different from noises and meaningful feature lines.

Those approaches are based on global curvature analysis may yield to squiggly and noisy feature lines. [15] present a modification of Laplacian smoothing algorithm to smooth feature lines, but their scheme works on the entire mesh before tracing feature lines. The algorithm we present in Section 3 is also similar with Laplacian smoothing [16], but works after detecting feature lines. Our algorithm can pay more attention to the local property of extracted feature lines and works well. Isenberg et al. [17] also present image-space algorithm to detect and remove artifact such as zig-zag style. In their study, they use their own visibility determination algorithm not the traditional hardware z-buffer algorithm for the conveniency of generating stroke. The artifact they mentioned in their paper such as zig-zag lines is produced by their own visibility determination algorithm,

not from the original model. They didn't mention how to removal the noise like short branches which comes from the original model and curvature estimation error.

Some researchers on image processing focus on the interactive feature detection method. e.g. [18] proposed a method called geometric snakes which is an extension of image snakes [19], [20] proposed an semi-automatic algorithm to obtain smooth lines. But these interactive methods are not fit for automatic 3D applications.

3 Achieving Smooth Feature Lines

In our study, we detect view- and scale- dependent ridge-valley lines as interior feature edges. We estimate the curvature first using [21]. Mainly following the definition in [10], we can get the result of feature lines as the left image in Figure 2 shows.

After feature lines are detected, we apply an iterative procedure on ridge-valley lines as follow to smooth the extracted feature lines. Our approach is similar with laplacian smoothing which is inspired by normal based mesh filtering [22], for each ridge vertex(or valley vertex)$\boldsymbol{v}$, with its normal vector $\boldsymbol{n_v}$, we find its two neighbors $\boldsymbol{v}_{prev}$ and $\boldsymbol{v}_{next}$ from the feature line. If a vertex has only one neighbor on the feature line(the start vertex or the end vertex of the line), or a vertex has more than two neighbors on the feature line(the branch vertex), we skip these vertices and don't apply the iteration operation on these vertices.

If the vertex have exactly two neighbors on the line, then let the middle point between $\boldsymbol{v}_{prev}$ and $\boldsymbol{v}_{next}$ be $\boldsymbol{v}_{middle} = (\boldsymbol{v}_{prev} + \boldsymbol{v}_{next})/2$. The vector from $\boldsymbol{v}$ to $\boldsymbol{v}_{middle}$ is $\boldsymbol{v}' = \boldsymbol{v}_{middle} - \boldsymbol{v}$. Define the difference of $\boldsymbol{v}$ on the line as

$$\Delta\boldsymbol{v} = \boldsymbol{v}' - (\boldsymbol{v}' \cdot \boldsymbol{n_v})\boldsymbol{n_v} \tag{1}$$

the iterative procedure will move $\boldsymbol{v}$ to a new position according to the difference $\Delta\boldsymbol{v}$. After each iteration step, the $\Delta\boldsymbol{v}$ should be updated with the new position of

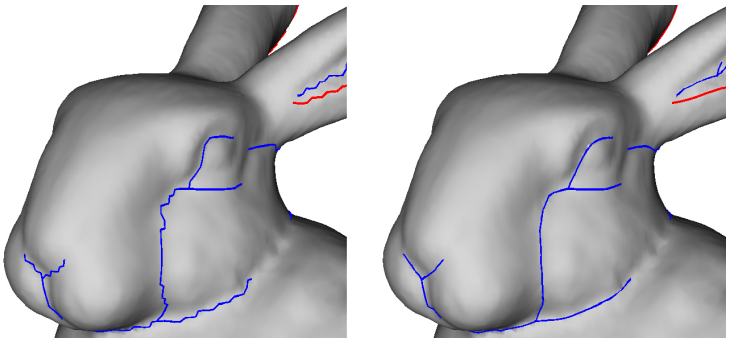

Fig. 2. A feature lines smoothing example of bunny model with 69k triangles. The left image is the result before line smoothing, squiggly lines exists. The right image is the result after smoothing without squiggly lines with $\lambda = 0.4$ after 20 iterations.

$\boldsymbol{v}$, we use $\boldsymbol{v}_0$ to represent the original position of vertex $\boldsymbol{v}$, and $\boldsymbol{v}_n$ to represent the new position of $\boldsymbol{v}$ after the nth iteration. the updated $\Delta\boldsymbol{v}$ after the nth iteration is represented by $\Delta\boldsymbol{v}_{(n)}$. The new position of the vertex $\boldsymbol{v}$ is calculated by Eq.2

$$\boldsymbol{v}_{(n)} = \boldsymbol{v}_{(n-1)} + \lambda\Delta\boldsymbol{v}_{(n-1)} \tag{2}$$

where λ is a controlling parameter which satisfy $0 \leq \lambda \leq 1$. After applied Eq.2 on ridge and valley vertex, the connected feature line can be more smooth as the right image of Figure 2 shows. For all of our experiments, $\lambda = 0.4$ proves to be a good choice, and the convergence can be achieved after 20 iteration times.

Compared with Laplacian smoothing scheme [15] working on the entire mesh, the proposed smoothing algorithm concentrate more on the local property of feature lines and is more direct and effective.

4 Feature Lines Optimizing

After being smoothed, the results still keep much noises and useless little branch lines which interfere with our observation. The key challenge is to distinguish these noises from meaningful feature lines.

For the feature lines extraction, each separate line can be denoted as $G_i =< V_i, E_i, W_i >$, where V_i is the list of vertices, E_i is the list of edges satisfying $E \subseteq [V]^2$, W_i is the weight function of E_i, in which $w_k \in W_i$ is weight of $e_k \in E_i$. we used the length of edge as weight in our study. Naturally, G_i is an undirected weighted graph. We call G_i a **Feature Graph**. Furthermore, by construction, a Feature Graph is also a connected graph. The extracted feature lines can be represented by a set of Feature Graph $\mathbb{G} =< G_0, G_1, ..., G_m >$, each $G_i \in \mathbb{G}$ correspond a line segment. We develop denoising algorithm through optimizing each $G_i \in \mathbb{G}$.

For each $G_i \in \mathbb{G}$, we computed the longest path P, with length L. Since feature graph G_i is a connected graph, it certainly contains a spanning tree. We build the Minimal Spanning Tree of G_i using the start vertex or the end vertex of the longest path as the root of tree.

Computed the *Longest* Path: Follow the definition of *path* in graph theory [23], a *path* is a non-empty graph $P =< V, E >$ of the form

$$V = x_0, x_1, ..., x_k, E = x_0x_1, x_2x_2, ..., x_{k-1}x_k$$

where the x_i are all distinct. Here we refer **the *Longest* Path** as the path $P_j \subseteq G_i$ with the maximal sum of edge weight in P of all path in graph G_i, mathematically, a path P_j is the longest path in the graph G_i if and only if $\sum_{e_k \in E(P_j), P_j \subseteq G_i} w_k$ is the maximum, e.g.

$$max\{\sum_{e_k \in E(P_0)} w_k, \sum_{e_k \in E(P_1)} w_k, ..., \sum_{e_k \in E(P_n)} w_k\}$$

Since we use the length of edge as weight, **the *Longest* Path** visually is the longest line branch in a separate feature line. Figure 3 shows three different

situations during computing, for each branch vertex t in a tree, the length from the root to the vertex t is $L_{t_{root}}$, length of the two longest low level branch of vertex t is L_{t_1} and L_{t_2}, then the longest path $L_{t_{max}}$ crossing the vertex t is $max\{L_{t_{root}} + L_{t_1}, L_{t_{root}} + L_{t_2}, L_{t_1} + L_{t_2}\}$. The longest path of tree T should be $max\{L_{v0}, L_{v1}, ..., L_{vn}\}, (v0, v1, ..., vn \in V(T))$.

Fig. 3. Example of three situation of the longest path computation. blue vertex is the root node of tree, and red path is the longest path of tree.

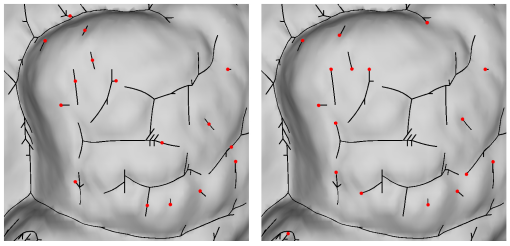

Fig. 4. Example of choosing root of Minimal Spanning. The red points are roots of MSTs. In the left image, we choose a vertex randomly as the MST's root to build the first MST T_1. Then we choose root as the start node(or the end node) of the longest path in T_1, rebuild the MST T_2 as the right image shows.

To compute the longest path, we randomly chose a vertex in graph as root, then build the minimal spanning tree of the graph with PRIM [23] algorithm. The left image of Figure 4 shows the random root choosing result. We compute the longest path in the tree, which is also the longest path in graph G_i.

After finding out the longest path in the minimal spanning tree builded first time, we choose the start vertex or the end vertex as root node to rebuild the minimal spanning tree of graph G_i. Figure 4 shows the result of rebuilded minimal spanning tree.

Finishing rebuilding minimal spanning tree of each feature graph, we optimize these trees to wipe out noises. Let P_i be the longest path of feature graph G_i, the length L_i of path P_i can be denoted as

$$L_i = \sum_{e_k \in E(P_i)} w_k$$

A global threshold θ and a local threshold ε are proposed to distinguish noises from meaningful feature, which satisfies

$$min\{L_0, L_1, ..., L_m\} \leq \theta \leq max\{L_0, L_1, ..., L_m\}$$

and

$$0 \leq \varepsilon \leq 1$$

We delete noises on extracted lines in the following two cases.

CASE I: If the length L_i of the longest path of feature graph G_i satisfies $L_i < \theta$, the line corresponded feature graph G_i is considered as noise and not rendered.

CASE II: Giving a feature graph G with its minimal spanning tree T. The longest path P with length L has been found. For each node $v_i \in T$, we compute the longest path l_i among all pathes from the node to all leaf nodes. We check each branch node $v_k \in P$ which has more than one child node. With preserving branch belongs to the longest path of tree, if length of one of other branches $l_{branch} < \varepsilon * L$, the branch is considered as noises and deleted. On the contrast, if length of one of other branches $l_{branch} > \varepsilon * L$, we apply same checking step on the subtree T' with the branch node being the root of T'.

Implementation of CASE II: CASE I is easy to implement. Implementation of CASE II is a little difficult. We implement the CASE II through a recursive function.

CHECK-ALL-BRANCH(GraphVertexIndex root)
for all vertex v in the longest path P
  for all branch b of vertex V
    if $l_{v.b} < l_{root} * \varepsilon$
      CUT-BRANCH($v.b$)
    else
      CHECK-ALL-BRANCH($v.b$)
    end if
  end for
end for

Figure 5 shows the optimizing result of Figure 4 with global threshold $\theta = 16$ and local threshold $\varepsilon = 0.05$. Meaningful result is obtained.

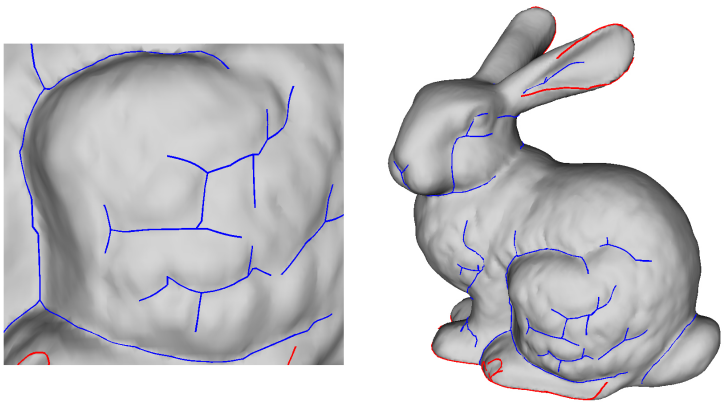

Fig. 5. Example of feature lines optimizing. $\theta = 16$, $\varepsilon = 0.05$

Limitation: In our method, the parameters θ and ϵ in the optimization step are set interactively now, it would be great if the program can compute these two parameters automatically according to different triangle mesh models. For

this purpose, more attention should be paid to the relationship between the two parameters and the average of the length of the longest path on all feature graphs of the entire model. This is also one of our future works.

5 Results and Discussion

Feature lines carry essential information about the geometry of a surface. But existing method can not work perfectly because of the computational error during curvature analysis on discrete mesh models, The unwanted noise sometimes annoy the feature detection results very much. The methods we proposed in this paper can not only smooth lines extracted through curvature analysis, but also provide a reasonable scheme to distinguish noise from meaningful feature lines.

Our method works well on various models and can obtain high quality feature line result. Figure 6 and 7 give two examples and the comparison with existing method [13].

Although we mainly focus on the view-independent feature lines, it's also easy to apply our algorithm to deal with view-dependent lines such as suggestive lines [7] [8]. But the computational time may increased. For view-independent lines optimization, we need to build feature graphs only once. But for view-dependent lines, different line-drawing result corresponds to different viewpoint. Each time the viewpoint changed, the feature graphs need to be re-generated and the computational time increases much. In our future work, more efficient algorithm dealing with view-dependent lines should be developed.

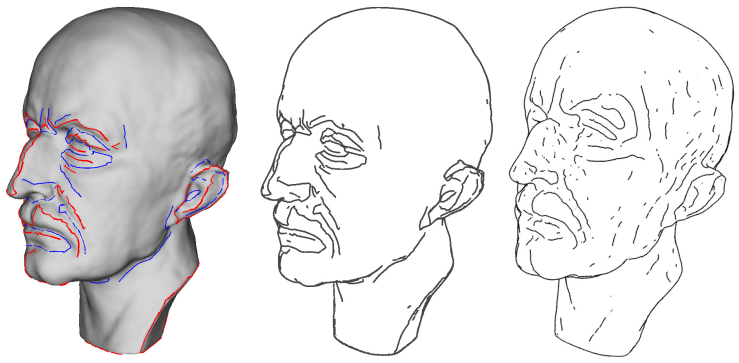

Fig. 6. Example of feature lines optimizing. Left image is the optimizing result with $\theta = 20$, $\varepsilon = 0.3$. Middle image is rendered with ridge-valley lines and silhouette edges. Right image is rendered using the algorithm [13].

Finally, greater precision on distinguish noise and feature line is required. Sometimes, meaningful feature lines would be deleted as noises and noises may be preserved as feature lines. In our future work, we will try to define noise not only by length of line segments, but also by other properties of lines, such as line curvatures.

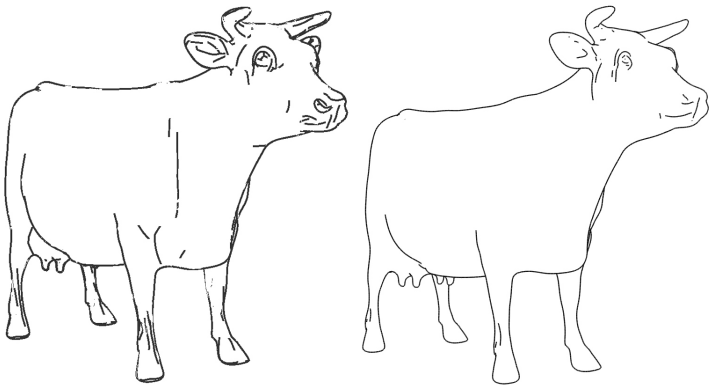

Fig. 7. Example of feature lines optimizing. Left image is the optimizing result with ridge-valley lines and silhouette edges, $\theta = 0.2$, $\varepsilon = 0.05$. Right image is rendered using the algorithm [13].

References

1. Markosian, L., Kowalski, M.A., Trychin, S.J., Bourdev, L.D.: Real-time nonphotorealistic rendering. In: Proceedings of SIGGRAPH 1997, ACM (1997)
2. Kalnins, R.D., Markosian, L., Meier, B.J., Kowalski, M.A., Lee, J.C.: Wysiwyg npr: Drawing strokes directly on 3d models. In: Proceedings of SIGGRAPH 2002, ACM (2002)
3. Hertzmann, A., Zorin, D.: Illustrating smooth surface. In: Proceedings of SIGGRAPH 2000, ACM (2000)
4. Raskar, R., Cohen, M.: Image precision silhouette edges. In: Symposium on Interactive 3D Graphics 1999, ACM (1999)
5. Winkenbach, G., Salesin, D.H.: Computer-generated pen-and-ink illustration. In: Proceedings of SIGGRAPH 1994, ACM (1994)
6. Strothotte, T., Schlechtweg, S.: Non-Photorealistic Computer Graphics, Modeling, Rendering, and Animation. Morgan Kaufmann Publisher (2002)
7. DeCarlo, D., Finkelstein, A., Rusinkicwicz, S., Santella, A.: Suggestive contours for conveying shape. ACM Transactions on Graphics **22**(3(July)) (2003) 848–855
8. Sousa, M.C., Prusinkiewicz, P.: A few good lines: Suggestive drawing of 3d models. Computer Graphics Forum **22**(3) (2003) 381–390
9. Ohtake, Y., Belyaev, A.: Automatic detection of geodesic ridges and ravines on polygonal surfaces. The Journal of Three Dimensional Images **15**(1) (2001) 127–132
10. Ohtake, Y., Belyaev, A.: Ridge-valley lines on meshes via implicit surface fitting. In: Proceedings of SIGGRAPH 2004, ACM (2004)
11. Belyaev, A., Ohtake, Y., Abe, K.: Detection of ridges and ravines on range images and triangular meshes. In: Proceedings of Vision Geometry IX, SPIE (2000)
12. Watanabe, K., Belyaev, A.: Detection of salient curvature features on polygonal surfaces. Computer & Graphics **20**(3) (2001) 385–392
13. DeCarlo, D., Finkelstein, A., Rusinkiewicz, S.: Interactive rendering of suggestive contours with temporal coherence. In: NPAR 2004, ACM (2004)

14. Koenderink, J.: What does the occluding contour. tell us about solid shape? Perception **13** (1984) 321–330
15. Hildebrandt, K., Polthier, K., Wardetzky, M.: Smooth feature lines on surface meshes. In: Eurographics Symposium on Geometry Processing. (2005)
16. Sorkine, O.: Laplacian mesh processing. In: Proceedings of Eurographics 2005, ACM (2005)
17. Isenberg, T., Halper, N., Strothotte, T.: Stylizing silhouettes at interactive rates: From silhouette edges to silhouette strokes. In: Proceedings of EUROGRAPHICS 2002. (2002)
18. Lee, Y., Lee, S.: Geometric snakes for triangular meshes. In: Proceedings of Eurographics 2002. (2002) 229–238
19. Kass, M., Witkin, A., Terzopoulos, D.: Snakes: Active contour models. IEEE Transactions on Pattern Analysis and Machine Intelligence **20**(11) (1998) 1260–1265
20. Guo, Y., Peng, Q., Hu, G., Wang, J.: Smooth feature line detection for meshes. The Journal of Zhejiang University SCIENCE **6A**(5) (2005) 460–468
21. Rusinkiewicz, S.: Estimating curvatures and their derivatives on triangle meshes. In: Symposium on 3D Data Processing, Visualization, and Transmission. (2004)
22. Chen, C.Y., Cheng, K.Y.: A sharpeness dependent filter for mesh smoothing. Computer Aided Geometric Design **22** (2005) 376–391
23. Diestel, R.: Graph Theory. Springer-Verlag (2000)