Feature Line Smoothing and Adaptive Optimizing Method for 3D Models

Hao Jing* and Bingfeng Zhou
Institute of Computer Science and Technology, Peking University, Beijing , 100871， China

Abstract: Feature lines, especially view- and scale-independent ridge-valley lines can indicate most prominent characteristics on surface. Mathematically, ridge-valley lines are defined as local extrema of principle curvatures along corresponding principle directions. When detecting ridge-valley lines on 3D mesh model, estimation of the curvature and first even higher order curvature derivatives is necessary. This may often yields to squiggly and noisy result since the estimation is sensitive against unwanted surface noises. We present an adaptive algorithm to obtain smooth and noiseless ridge-valley lines based on graph theory model. After an iteration procedure acting on local ridge (or valley) vertices and their previous and next neighbors on connected lines is applied to smooth feature lines, each separate feature line is considered as an undirected weighted graph which is called as **Feature Graph**. With a correct root node choosing, the minimal spanning tree of the feature graph is computed. We can automatically get rid of most noises and preserve meaningful feature lines through optimizing those minimal spanning trees.

Keywords： Feature Line， Line Smoothing, Graph Optimizing

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 meanwhile 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 [15] [11][7][18][23]. In general, feature lines include **Silhouette Edges, Boundary Edges and Interior Feature Edges** [21]. 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 models. For extraction of silhouette edges, many fast extracting algorithms have been developed and can achieve good results[15] [11] [7] [18].

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 [3][20].

There have been various existing interior feature edges detection algorithms [16][17][1][22][3][20]. All existing feature line extraction methods, however, treated the noises and meaningful feature lines of surface as the same role in computing procedure, the algorithm we present can reasonably separate noises from meaningful lines automatically.

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 may leads to two visual clutters as left image of figure 1 shows. First, The curvature based estimation may lead to much noises in feature lines extracting results. Secondly, because we connected the triangle vertices as feature lines, squiggly lines may be produced. The left image of figure 2 shows this situation.

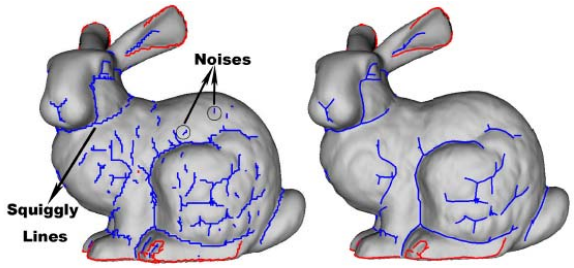

Figure 1: A feature lines smoothing and optimizing example of bunny model with 69k triangles. The left image is the result before line smoothing and optimizing, squiggly and noiseful lines can be seen. The right image is the result after adaptive optimizing. Feature lines are smooth and noiseless.

In our paper, we proposed an adaptive algorithm to optimize feature lines results and to preserve meaningful lines automatically. After obtaining the ridge and valley

* Tel: +86-10-82529690; E-mail: jinghao@icst.pku.edu.cn

vertices on mesh surface and connecting them, naturally, a separate feature line can be treated as an undirected graph weighted by the length of edges. All the extracted feature lines on a 3D model can be viewed as a set of undirected weighted graphs. We call these graphs as Feature Graphs. 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 automatically 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 providing a reasonable and extendable scheme on how to divide noise from meaningful feature lines automatically. In the remaining of this paper, we will introduce previous work in Section 2. In Section 3, the ridge-valley lines smoothing algorithm will be given. Section 4describe the adaptive optimizing algorithm used to wipe out noises. The conclusions and more results are given in Section 5.

2. Previous Work

Various feature detection algorithms have been proposed during the past decade, including ridge-valley lines detection ([16][17][1] [22]) and suggestive feature lines ([3][4][20]). Earlier papers focus on extraction of view-dependent silhouette edges ([11][7][18]), 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[15]. However, much 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[3] defined interior feature edges as view-dependent suggestive contour, which is extracted through radial curvature[13] analysis. The authors improved their algorithm in [4] with increasing of detecting speed. Sousa and Prusinkiewicz[20] present another automated algorithm to produce suggestive line drawing, but 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 [16] [1], and extract these 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 [17].

Ohtake[17] 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. [10] present a algorithm similar with the Laplacian scheme to obtain feature lines on 3D modes. Researchers present a modification of Laplacian smoothing algorithm to smooth feature lines [8], 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 [19], 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. [9] also present image-space algorithm to detect and remove artifact such as zigzag style. In their study, they use their own visibility determination algorithm not the traditional hardware z-buffer algorithm for the convenience of generating stroke. The artifact they mentioned in their paper such as zigzag 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. [14] proposed a method called geometric snakes which is an extension of image snakes [12], [6] 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, and mainly follow the definition in [17] to extract ridge-valley lines. Because the line is connected by many ridge (or valley) vertices on the 3D model(figure 2 shows this situation), we get the result of squiggly ridge-valley lines as the left image in Figure 3 shows.

After feature lines 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 [2], for each ridge vertex(or valley vertex) $\vec{v}$, with its normal vector $\vec{n}_v$, we find its two neighbors $\vec{v}\_prev$ and $\vec{v}\_next$ from the feature line.

If a vertex has only one neighbor on the feature line (the start node or the end node 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 $\vec{v}\_prev$ and $\vec{v}\_next$ be $\vec{v}\_middle = (\vec{v}\_prev + \vec{v}\_next)/2$. The vector from $\vec{v}$ to $\vec{v}\_middle$ is $\vec{v}' = \vec{v}\_middle - \vec{v}$. Define the difference of $\vec{v}$ on the line as equation 1.

$$\Delta\vec{v}' = \vec{v}' - (\vec{v} \cdot \vec{n}_v)\vec{n}_v \quad (1)$$

The iterative procedure will move $\vec{v}$ to a new position according to the difference $\Delta\vec{v}$. After each iteration step, the $\Delta\vec{v}$ should be updated with the new position of $\vec{v}$, we use $\vec{v}_{(0)}$ to represent the original position of vertex $\vec{v}$, and $\vec{v}_{(n)}$ to represent the new position of $\vec{v}$ after the *n*th iteration. The updated $\Delta\vec{v}$ after the *n*th iteration is represented by $\Delta\vec{v}_{(n)}$. The new position of the vertex $\vec{v}$ is calculated by Eq.2

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

Where λ is a controlling parameter which satisfies $0 < \lambda < 1$. After applied Eq.2 on ridge (or valley) vertex, the connected feature line can be smoother as the right image of Figure 2 and 3 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[8] working on the entire mesh, the proposed smoothing algorithm concentrate more on the local property of feature lines and is more direct and effective.

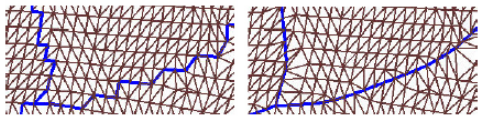

Figure 2: A feature lines smoothing example on part of bunny model with 69k triangles. The left image shows the result before line smoothing on the original triangle mesh, squiggly lines exists. The right image is the result after smoothing without squiggly lines with λ = 0.4 after 20 iterations, vertices are moved through smoothing algorithm to obtain smooth lines.

4. Adaptive Optimizing Method of Feature Lines

In our research, the key challenge is to distinguish these noises from meaningful feature lines. In this section, we first build feature graph according to extracted feature line. After building minimal spanning trees, we will describe the automated algorithm to optimize these feature graphs in two cases.

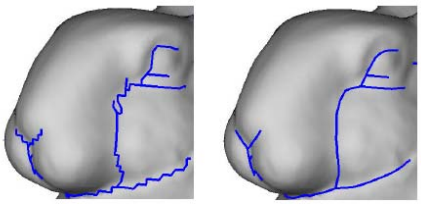

Figure 3: A feature lines smoothing example of bunny model with 69k triangles. The left image is the result before line smoothing, squiggly lines exists. The right image is the result after smoothing without squiggly lines with λ = 0.4 after 20 iterations.

4.1 Build Feature Graph

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 $e_k \in E_i$. In each graph G_i, the number of vertex is more than 2. we used the length of edge as weight in our study. Naturally, G_i is an undirected weighted graph. We call G_i as Feature Graph. Furthermore, by construction, a Feature Graph is also a connected graph [5]. The extracted feature lines can be represented by a set of Feature Graph $G\_set = <G_0, G_1, \ldots, G_m>$, each $Gi \in G\_set$ correspond a separate feature line segment. We develop denoising algorithm through optimizing each $Gi \in G\_set$.

4.2 Optimizing Each Feature Graph

For each $Gi \in G\_set$, 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 node or the end node of the longest path as the root of tree.

Computed the *Longest* Path: Follow the definition of *path* in graph theory [5], a ***path*** is a non-empty graph $P = <V, E>$ of the form

$$V = x_0, x_1, \ldots, x_k, E = x_0x_1, x_1x_2, \ldots, x_{k-1}x_k$$

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 paths in graph G_i. In our study, the edge

weight is the length of the edge. 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.

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

Since we use the length of edge as weight, the *Longest* Path visually is the longest line branch in a separate feature line. Figure 4 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_{t1} and L_{t2}, then the longest path L_{t_max} crossing the vertex t is $max\{L_{t\_root}+L_{t1}, L_{t\_root}+L_{t2}, L_{t1}+L_{t2}\}$. The longest path of tree T should be $max\{L_{v0}, L_{v1}, \ldots, L_{vn}\}$; $(v0, v1, \ldots, vn \in V(T))$.

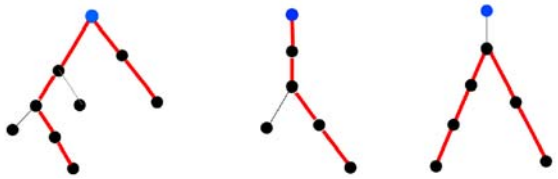

Figure 4: Example of three situations of the longest path computation. blue vertex is the root node of tree, and red path is the longest path of tree

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 [5] algorithm. The left image of Figure 5 shows the random root choosing result. We compute the longest path in the tree, which is also the longest path in graph G_i.

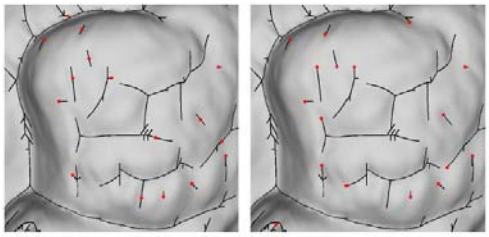

Figure 5: Example of choosing root of Minimal Spanning. The red points are roots of MSTs. In the left image, we choose a vertex randomly as the MST's root to build the first MST T_1. Then we choose root as the start node(or the end node) of the longest path in T_1, rebuild the MST T_2 as the right image shows.

After finding out the longest path in the minimal spanning tree builded first time, we choose the start node or the end node as root node, and then we rebuild the minimal spanning tree of graph G_i. The right image of figure 5 shows the result of choosing start node or the end node as root node after computing the longest path. 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

$$Li = \sum_{ek \in E(Pi)} w_k$$

A global threshold θ and local threshold ε are proposed to distinguish noises from meaningful feature. For each G, θ satisfies:

$$\min\{L_0, L_1, \ldots, L_m\} \le \theta \le \max\{L_0, L_1, \ldots, L_m\}$$

and ε satisfies:

$$0 \le \varepsilon \le 1$$

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 feature line corresponded feature graph G_i is considered as noise and not rendered.

CASE II: In this case, first we set a split parameter N_{split} which is a positive integer:

$$N_{split} = \max\{L_0, L_1, \ldots, L_m\} / average\{L_0, L_1, \ldots, L_m\}$$

Giving a feature graph $G \in G\_set$ with its rebounded minimal spanning tree T, if its longest path L satisfies $L > average\{L_0, L_1, \ldots, L_m\}$, we separate the longest path of G to Nsplit parts. Then the original set of feature graph $G\_set =< G_0, G_1, \ldots, G_m >$ becomes a new set of feature graph $G\_set' =< G_0, G_1, \ldots, G_m' >$. While separating, we ensure the vertex number of each graph is more than one; otherwise the only vertex of this graph would join the neighbor graph. The reason why we split feature graph is that we can preserve much more details when optimizing feature lines after splitting feature graphs. Figure 6 shows comparison between optimizing result with splitting and optimizing result without splitting, details on the eye part of the bunny model are preserve well.

Giving a feature graph $G \in G\_set'$ after splitting with its minimal spanning tree T. The longest path P with length L has been found. For each node $v_i \in T$, the longest path l_i among all paths from the node to all leaf nodes is computed. We first check each branch node $v_k \in P$ in the

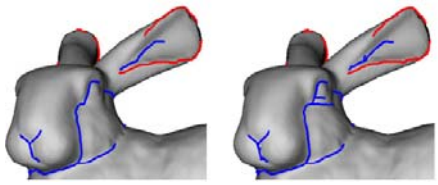

Figure 6: Comparison between optimizing result with splitting (right image) and optimizing result without splitting (left image). details on the eye part of the bunny model are preserve well.

longest path which has more than one child node. For branches 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'. This optimizing case can be implemented by a recursive procedure.

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 as follows. For each feature graph $G \in G\_set'$:

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

After optimizing *G_set* through θ and optimizing *G_set'* through ε. For vertex *v* in a 3D model, we render the vertex if *v* satisfies both of the following two conditions.

1. *v* belongs feature graph set *G_set* which is before splitting and *v* is meaningful node through optimizing procedure with θ.
2. *v* belongs feature graph set *G_set'* which is after splitting and *v* is meaningful node through optimizing procedure with ε.

4.3 Compute thresholds θ and ε automatically

In our adaptive algorithm, these two thresholds q and e are computed automatically. In this section, we describe the rules to compute thresholds automatically.

For each feature graph $G_i \in G\_set$, L_i is the longest path of G_i which is computed in section 4.2, θ is calculated as follows:

$$\theta = average\{L_0, L_1, ..., L_m\}$$

The computation of ε is much complicated than the computation of θ. For each branch node $\{v_{branch1}, v_{branch2}, ..., v_{branchn}\}$ in feature graph $G_i \in G\_set'$, we compute the longest path $\{L_{branch1}, L_{branch2}, ..., L_{branchn}\}$ of subtrees with each branch node being the root. We set level value for each node. The level value of each node in the longest path of feature graph G_i is 0. For each branch node in the longest path, the level value of nodes in the longest path of subtree with the branch node being the root is the level value of branch node plus 1, as figure 7 shows.

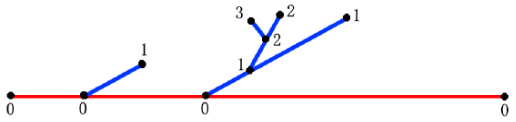

Figure 7: example of setting level value of nodes on tree

For each nodes $v_{branchi} \in \{v_{branch1}, v_{branch2}, ..., v_{branchn}\}$, we compute $e_{branchi}$ as follows:

$$e_{branchi} = L_{branchi} / L_{upleveli}$$

We denote $\boldsymbol{L}_{branchi}$ as the longest path of the subtree with $v_{branchi}$ being the root, $level_{branchi}$ as the level value of $v_{branchi}$, $\boldsymbol{L}_{upleveli}$ as the length of path which satisfies two condition. First, the path should passes $v_{branchi}$, secondly, the level value of nodes on the path should be $level_{branchi}$.

$$\varepsilon_i = average\{\varepsilon_{branch1}, \varepsilon_{branch2}, ..., \varepsilon_{branchn}\}$$

For each $Gi \in G\_set'$, $\varepsilon_i = average\{\varepsilon_1, \varepsilon_2, ..., \varepsilon_n\}$

Here θ and ε have been computed automatically.

5. Results and Conclusions

Feature lines carry essential information about the geometry of a surface. Existing feature line extraction method can not work perfect because of the computational error during curvature analysis on discrete mesh models. 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. So we call our method as adaptive optimizing method. Figure 8 and 9 give two examples and the comparison with existing method [4].

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 [3][20]. But the computational time may increase. For view- and scale- independent ridge-valley 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 future works, more efficient algorithm dealing with view-dependent lines should be developed.

References

[1] Alexander Belyaev, Yutaka Ohtake, and Kasumi Abe. Detection of ridges and ravines on range images and triangular meshes. In *Proceedings of Vision Geometry IX*. SPIE, 2000.

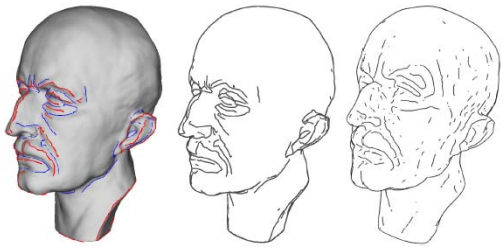

Fig. 7. Example of feature lines optimizing. Left image is the optimizing result. Middle image is rendered with both optimized lines and silhouette edges. Right image is rendered using the algorithm [3].

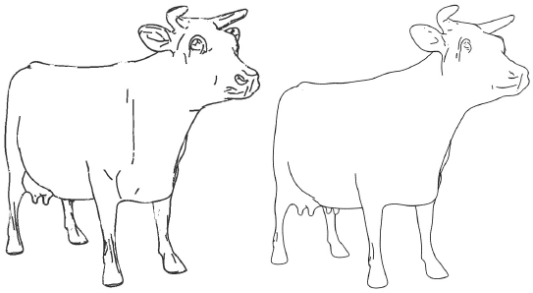

Figure 9: Example of feature lines optimizing. Left image is the optimizing result with ridge-valley lines and silhouette edges with our adaptive optimizing algorithm. Right image is rendered using the algorithm [3].

triangular meshes. In *Proceedings of Vision Geometry IX*. SPIE, 2000.

[2] Chun Yen Chen and Kuo Young Cheng. A sharpeness dependent filter for mesh smoothing. *Computer Aided Geometric Design*, 22:376–391, 2005.

[3] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkicwicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Transactions on Graphics*, 22(3(July)):848–855, 2003.

[4] Doug DeCarlo, Adam Finkelstein, and Szymon Rusinkiewicz. Interactive rendering of suggestive contours with temporal coherence. In *NPAR 2004*. ACM, 2004.

[5] Reinhard Diestel. *Graph Theory*. Springer-Verlag, 2000.

[6] Yanwen Guo, Qunsheng Peng, Guofei Hu, and Jin Wang. Smooth feature line detection for meshes. *The Journal of Zhejiang University SCIENCE*, 6A(5):460–468, 2005.

[7] Aaron Hertzmann and Denis Zorin. Illustrating smooth surface. In *Proceedings of SIGGRAPH 2000*. ACM, 2000.

[8] Klaus Hildebrandt, Konrad Polthier, and Max Wardetzky. Smooth feature lines on surface meshes. In *Eurographics Symposium on Geometry Processing*, 2005.

[9] Tobias Isenberg, Nick Halper, and Thomas Strothotte. Stylizing silhouettes at interactive rates: From silhouette edges to silhouette strokes. In *Proceedings of EUROGRAPHICS 2002*, 2002.

[10] Hao Jing and Bingfeng Zhou. A 3d model feature-line extraction method using mesh sharpening. *Lecture Notes in Computer Science 3942*, pages 840–848, 2006.

[11] Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, and Joseph C. Lee. Wysiwyg npr: Drawing strokes directly on 3d models. In *Proceedings of SIGGRAPH 2002*. ACM, 2002.

[12] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1260–1265, 1998.

[13] Jan Koenderink. What does the occluding contour. tell us about solid shape? *Perception*, 13:321–330, 1984.

[14] Yunjin Lee and Seungyong Lee. Geometric snakes for triangular meshes. In *Proceedings of Eurographics 2002*, pages 229–238, 2002.

[15] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, and Lubomir D. Bourdev. Real-time nonphotorealistic rendering. In *Proceedings of SIGGRAPH 1997*. ACM, 1997.

[16] Yutaka Ohtake and Alexander Belyaev. Automatic detection of geodesic ridges and ravines on polygonal surfaces. *The Journal of Three Dimensional Images*, 15(1):127–132, 2001.

[17] Yutaka Ohtake and Alexander Belyaev. Ridge-valley lines on meshes via implicit surface fitting. In *Proceedings of SIGGRAPH 2004*. ACM, 2004.

[18] Ramesh Raskar and Michael Cohen. Image precision silhouette edges. In *Symposium on Interactive 3D Graphics 1999*. ACM, 1999.

[19] Olga Sorkine. Laplacian mesh processing. In *Proceedings of Eurographics 2005*. ACM, 2005.

[20] Mario Costa Sousa and Przemyslaw Prusinkiewicz. A few good lines: Suggestive drawing of 3d models. *Computer Graphics Forum*, 22(3):381–390, 2003.