A 3D Model Feature-Line Extraction Method Using Mesh Sharpening

Hao Jing and Bingfeng Zhou

Institute of Computer Science and Technology, Peking University

Abstract. The feature-line extraction of a 3D model is a key step in the model-based Non-Photorealistic Rendering. In this paper, we introduce a new algorithm that is based on a sharpening filter to extract the feature-lines of 3D models. Experiments of feature-line rendering where our sharpening filter is introduced as a pre-calculation step are shown to compare with the existing algorithms ([1][2][3]). From these experiments it can be found that our rendering results reserve more feature details and contain less noise. Furthermore, in our algorithm, the computation time of rendering is also reduced.

1 Introduction

In model based Non-Photorealistic Rendering, the feature-line extraction of a 3D model is an important step, which is the basis of many post processing and rendering technologies. A feature-line, by its definition, is part of the surface points of a 3D model, where the important shape information of the model is delivered. The feature-line usually includes silhouettes, creases and boundaries [4]. In this paper, we describe a new way based upon which a better rendering result can be achieved with more details and also in real time (Figure 1 and 2). Compared with existing algorithms, the results obtained by our algorithm can produce more feature details of a model and remove more noise. In the meantime, the computation time of rendering can also be reduced.

Existing feature-line extraction algorithms can be divided into two categories. The first category is addressed as *traditional rendering algorithms*. These algorithms extract the feature-lines from a 3D model using the definition of silhouette directly and are well described in earlier papers ([1] [2] [5] [6][7]). The algorithms which fall into the second category ([2][3]) are addressed as *curvature based algorithms*. These algorithms extract feature-lines not only based on the silhouette definition, but also by using the surface curvature of the model. With these algorithms, the details can be shown better than algorithms in the first category.

Since the algorithms from the first category extract the feature-line based on the silhouette definition only, the rendering result contains only the simplest edge information as shown in Figure 3 and 4, where less detail can be seen. For instance, in Figure 3 and 4, the eyes of the model are missing.

Important features of a smooth surface include the curvature and the curvature derivative on each surface point along the view direction. These features are made use of by many existing methods as described in [8], [9], [10], [11], [12] and [13]. Considering the limitations made by the current status of 3D data acquisition technology that

Z. Pan et al. (Eds.): Edutainment 2006, LNCS 3942, pp. 840–848, 2006.
© Springer-Verlag Berlin Heidelberg 2006

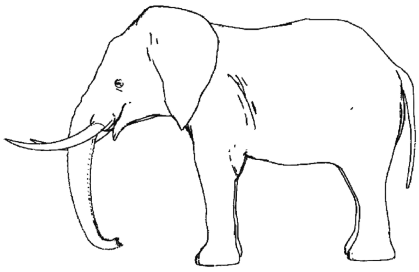


Fig. 1. A model rendered using our algorithm, with 5132 vertices and 10150 triangles

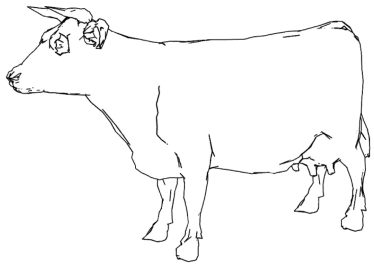


Fig. 2. A model rendered using our algorithm, with 2903 vertices and 5804 triangles and sharpening parameters the same as Figure 8

the smooth surfaces in real world are usually simulated by triangle mesh in a model, DeCarlo[2] suggested a rendering algorithm that takes not only the definition of silhouettes, but also those two features into account ([2]). Similar work is also suggested by Sousa and Prusinkiewicz in their paper ([3]). These algorithms can achieve acceptable results; however they do consume too much CPU time computing the directional derivative along the view direction. Each time the viewpoint is changed, the directional derivatives have to be recomputed.

The rendering algorithm we suggest in this paper is based on a sharpening filter. By sharpening the model with a filter first and then extracting the feature-lines from the sharpened model with existing algorithms[4], better results than those achieved in [2] and [3] are obtained. Experiments show that our algorithm works well on all the models we chose and obtains results such as those shown in Figure 1 and 2.

In the remaining of this paper, we will introduce related works; discuss the rendering framework of our experiment as well as our sharpening filter, in Sections 2, 3 and 4 respectively. We will then go on to describe the experimental results achieved and the analysis of our method in Sections 5 and 6. The last section will concludes this work and summarizes what we have shown in this paper.

2 Related Works

The descriptions of the algorithms falling into category one (*traditional rendering algorithms*) can be found in [1], [5], [6] and [7]. In the results of these algorithms, the feature-lines in the convex part on the model can not be rendered perfectly, as shown in Figure 9(a). Compared with the left image of Figure 7, which is the original image of the model, Figure 9(a) does not contain enough detail to represent the feature information of the original model (the left image of Figure 7).

Existing methods of rendering more lines to make the results contain more feature information can be found in [2] and [3]. These methods make use of the curvature of the surface of a model in their feature-line computing. DeCarlo et al. ([2]) suggest a method named *suggestive contour rendering* that is based on radial curvature computing, and can render more feature lines. However, because the radial curvature in each point is

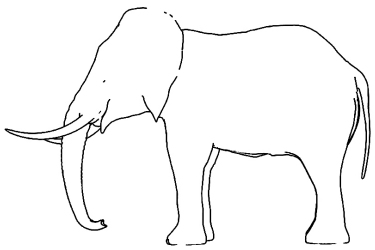


Fig. 3. Only the silhouette of elephant 3D model is rendered

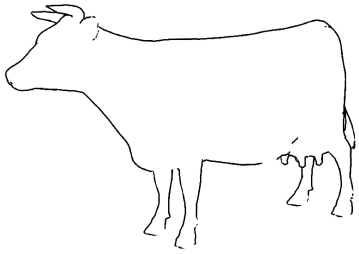


Fig. 4. Only the silhouette of cow 3D model is rendered

related to the view vector, each time the view vector changes, the radial curvature, the gradient of curvature and the directional derivative of radial curvature along view vector must all be recomputed. Thus, when rendering a model with too many vertices, the rendering efficiency of this algorithm decreases rapidly. Furthermore, the result of this method contains noise that is not satisfactory.

Inspired by DeCarlo's work, where there is a pre-process step using a smoothing filter algorithm, we introduced in our work a sharpening filter algorithm as a pre-process step that can improve the results of the existing algorithms significantly, producing results that are even better than DeCarlo's.

3 Rendering Framework

The rendering procedure where our sharpening filter takes effect mainly follows the framework of suggestive contour rendering [2]. The suggestive contour rendering includes three steps, that is: pre-processing, silhouette rendering and suggestive contour rendering. However our procedure only includes two steps, that is, pre-processing and feature-line rendering. DeCarlo et al. uses a Gaussian smoothing algorithm in pre-processing step whereas our work uses a sharpening filter. Our sharpening filter equation takes a similar form as Gaussian smoothing equation, but the purpose is completely opposite. After the pre-processing carried out by the sharpening filter, the convex part on the model is treated as the feature-line and is rendered using the surface property including silhouette and creases ([1][4]). The computation of curvature therefore becomes not necessary and can thus be removed.

In the following we shall describe the first two steps of the procedure of suggestive contour rendering, and in section 4, we will describe our modification to the pre-processing step which makes it become a sharpening operation. The two steps to be described in the next sections are Gaussian smoothing and silhouette extraction.

3.1 Gaussian Smoothing

In the framework of suggestive contour rendering, the Gaussian smoothing is employed as a pre-processing step, where a surface on a model is defined as a pair of lists in Eq. (1).

$$S = \{V, F\} \tag{1}$$

In Eq.(1), V is a list of vertices, $V = \{v_i | 1 \leq i \leq n\}$ where $\boldsymbol{v}_i \in R^3$ is the coordinate vector of the position of the i-th vertex, n is the number of vertices contained in the model. F is the set of faces constituting the surface of the model, that is, $F = \{f_k | 1 \leq k \leq m\}$, where f_k is a sequence of non-repeated vertex indices, m is the number of faces contained in the model. So for a triangle mesh model, each f_k contains three vertex indices. In the remaining of this paper, triangle models will be referred to as surface models. So for a given node v_i in a triangle mesh, a node v_j ($i \neq j$) is a neighbor of v_i if and only if they are connected by an edge. In the following, we denote B_i as the set of vertices that are neighbors of vertex v_i. Given the above definition, the Gaussian smoothing for each model vertex v_i is given by Eq.(2) and (3).

$$\Delta \boldsymbol{v}_i = \sum_{v_j \in B_i} w_{ij}(\boldsymbol{v}_j - \boldsymbol{v}_i) \tag{2}$$

$$\boldsymbol{v}_i^{\cdot} = \boldsymbol{v}_i + \lambda \Delta \boldsymbol{v}_i \tag{3}$$

where $\boldsymbol{v}_i^{\cdot}$ is the new position of vertex v_i after smoothing. w_{ij} is the average weight that satisfies $\sum_{v_j \in B_i} w_{ij} = 1$ and λ is the scale factor satisfying $0 < \lambda < 1$. In this framework, the Gaussian smoothing is an iteration procedure. It is performed repeatedly until the necessary smoothing effect is achieved [14].

3.2 Silhouette Rendering

A silhouettes, by its nature, is the boundary on the surface of a model which separates the visible part from invisible pieces of the surface of the model, as Figure 5 shows.

Defining the silhouettes mathematically, assuming the surface is smooth, we can say that silhouette points are where the dot product of view vector and normal vector equals zero, which is illustrated in Figure 6.

Silhouettes are extracted in the second step of the rendering framework. When implementing this step of the framework, in addition to extracting silhouettes, we also consider the two kinds of triangle edges that must be drawn when rendering. The two kinds of edges that are addressed in our paper are *boundary* and *crease* edges. The boundary only exists in non-closed surface meshes; they are the edges that are contained in only one triangle. A crease is an edge where the triangles are connected creating a dihedral angle that is less than 90°. In the rest of this paper, without ambiguity, we shall refer to the above three kinds of drawings all as feature-lines.

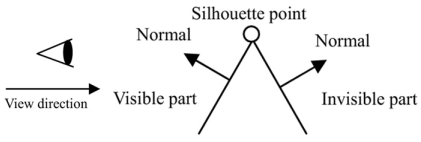


Fig. 5. Silhouette defined on polygon mesh

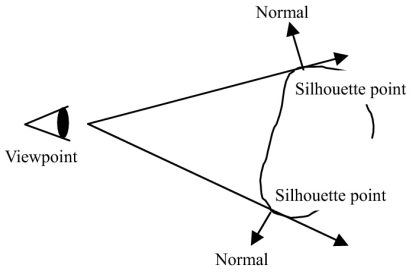


Fig. 6. Silhouette defined on a smooth surface

4 A Sharpening Filter for a 3D Model

In our method, the sharpening filter is a pre-processing procedure to a model. Its purpose is to amplify the convex nature of a surface model (mesh sharpening) so that the amplified nature is much more sensitive to the feature-line extracting step. After sharpening, we can render the feature-lines with less computation and to a lesser extend of visible noise which can be seen in other existing algorithms.

For each vertex v_i on the model, we calculate its difference with its neighbors $\Delta \boldsymbol{v}_i$ as defined in Eq.(2)(the weights w_{ij} here are of the same value). Instead of using Eq.(3), we use following equation to calculate the new position of vertex v_i to achieve the sharpening effect on the model:

$$\boldsymbol{v}_i^{,} = \boldsymbol{v}_i + (\mu - \lambda) p(\Delta \boldsymbol{v}_i) - \mu\lambda \Delta \boldsymbol{v}_i \tag{4}$$

where, if $\boldsymbol{v} = (x, y, z)$, then

$$p(\boldsymbol{v}) = (\sqrt{|x|}, \sqrt{|y|}, \sqrt{|z|}) \tag{5}$$

denotes a vector in R^3 whose component is the square root of the absolute value of each component of the vector $\boldsymbol{v}$ respectively. In Eq.(4), μ, λ are the scale factors which satisfy $0 < \lambda < \mu < 1$, and $\mu\lambda$ is much larger than $\mu - \lambda$ (which means μ and λ are very close). Given this precondition, it means that this equation function is completely opposite to the one in Gaussian smoothing [14], and thus is a sharpening filter. The operation can run iteratively. The number of iterations is determined by the actual sharpening effect.

A result of this sharpening filter can be found in Figure 1 and 2 with $\mu = 0.24$, $\lambda = 0.23$. More results and comparisons will be given in the following section. Without using the suggestive contour calculation, our algorithm can not only produce more detailed line drawings from the model, but can also reduce large portions of the computation caused by suggestive contour rendering. From Eq.(4), supposing the number of vertices is n, the average number of neighbors in a given mesh for each vertices is k, then for each sharpening operation, the computation estimate is $k{\cdot}o(n)$, which is approximately linear with the number of vertices on the model.

5 Implementation and Results

We implemented a rendering system based on the framework described in sections 3 and 4. We performed feature-lines extraction experiments with several 3D models using our algorithm and compared the results with the existing algorithms ([1], [2]). The models are chosen so that they are typical for different types of 3D models, thus the robustness of our algorithm could then be tested. The results and comparison can be found in Figures 8 to 12. From the comparison, it can be found that the results obtained by our algorithm contain much more feature details and eliminate the meaningless line drawings generated by using the existing algorithms (Figure 8(a), 8(b), 9(a), 9(b), 10(a), 10(b), 11(a), 11(b), 12(a), 12(b)). As can be seen, our drawings appear much more vivid (Figure 8(c), 9(c), 10(c), 11(c), 12(c)).

Table 1. Parameters used in the experiments

Figures	μ	λ	N
Fugure 8	0.24	0.23	6
Fugure 9	0.24	0.23	4
Fugure 10	0.53	0.52	1
Fugure 11	0.34	0.33	3
Fugure 12	0.24	0.23	5

The experiment environment we used is a personal computer with Pentium IV 2.4G CPU and a 64M NVIDIA GForce4 MX440 display card, the rendering results can be browsed as a 3D scene in real time .

The scale factor μ, λ and the number of sharpening iteration N for each experiment result can be found in Table 1. The iteration number N was decided by the visual inspection during the experiment. The μ, λ are decided by experience. Apparently and naturally, μ, λ are negatively related to N.

6 Analysis

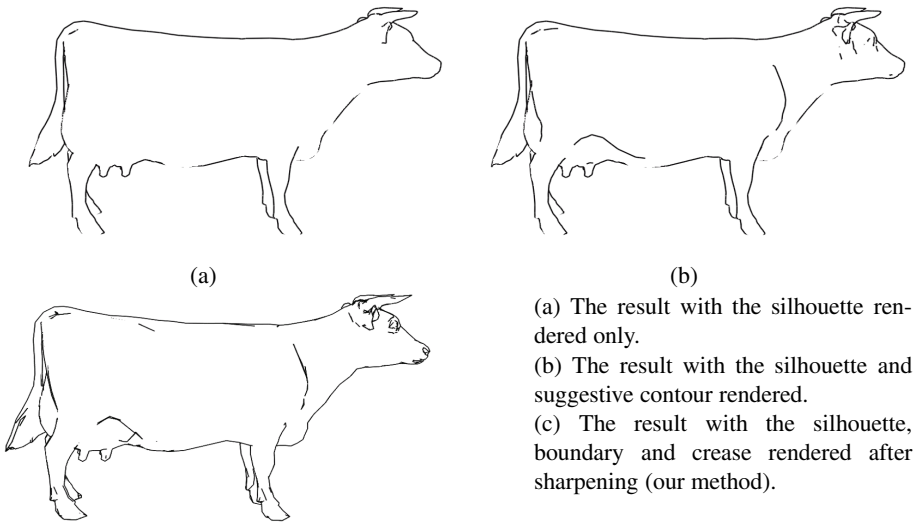
From the results given in the previous section, rendering with our sharpening filter can achieve better results. The main idea of our method is to modify the Gaussian smooth filter [14] so that it functions as a sharpening filter instead. Given this target, the sharpening filter should be written as following:

$$\boldsymbol{v}_i^{:} = \boldsymbol{v}_i - \mu\lambda\Delta\boldsymbol{v}_i \tag{6}$$

In our experiment, this filter may cause, just like the shrinkage effect in Gaussian smooth filter [14], an over-sharpening effect which can cause a messy result (middle image in Figure 7). To minimize this effect, we add a third term as described in Eq.(7) which can reduce this effect(right image in Figure 7).

$$(\mu - \lambda)p(\Delta\boldsymbol{v}_i) \tag{7}$$

Fig. 7. The left image is the original "Beethoven" model with 2655 vertices and 5030 triangles, the middle image is over-sharpening resulted from the filter defined by Eq.6. Right image is result in which over-sharpening is reduced using the filter defined by Eq.4. The parameters used are the same as those used in Figure 9.



(a) (b)

(a) The result with the silhouette rendered only.
(b) The result with the silhouette and suggestive contour rendered.
(c) The result with the silhouette, boundary and crease rendered after sharpening (our method).

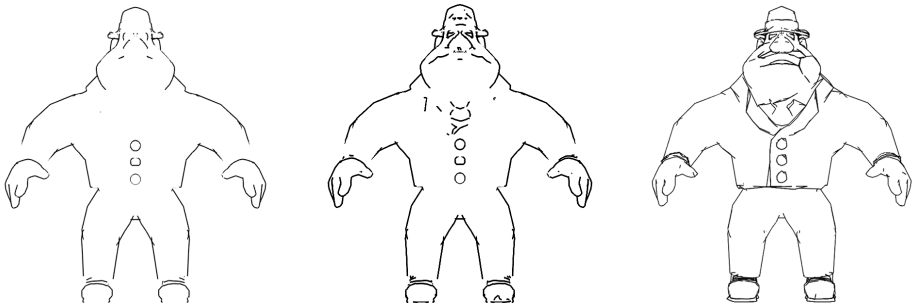
(c)

Fig. 8. Same model as Figure 2

(a) (b) (c)

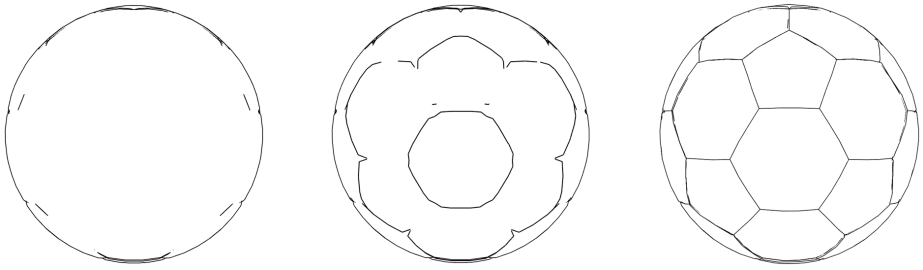
Fig. 9. Same model as Figure 7 with 2655 vertices and 5030 triangles. (a) The result with the silhouette rendered only. (b) The result with the silhouette and suggestive contour rendered. (c) The result with the silhouette, boundary and crease rendered after sharpening (our method).

The reason it works, we believe, is that it acts as a perturbation operator over the sharpening filter $\boldsymbol{v}_i - \mu\lambda\Delta\boldsymbol{v}_i$. The absolution operation in Eq.7 (Eq.5) makes the direction of the perturbation vector $(\mu - \lambda)p(\Delta\boldsymbol{v}_i)$ stay within the (+1,+1,+1) octant. When the changed direction is not the same as the direction of $-\mu\lambda\Delta\boldsymbol{v}_i$, it perturbs the growing of the sharp edge. Moreover, the square root operation further changes the direction of this vector. The shortcoming of the including of square root operation in our formula is that it makes the sharpening filter scaling-dependent. How to overcome this shortcoming is one of our future works. One of our idea is that we can remove the square root operation from the filter, our preliminary experiment shows that this can produce the similar results.



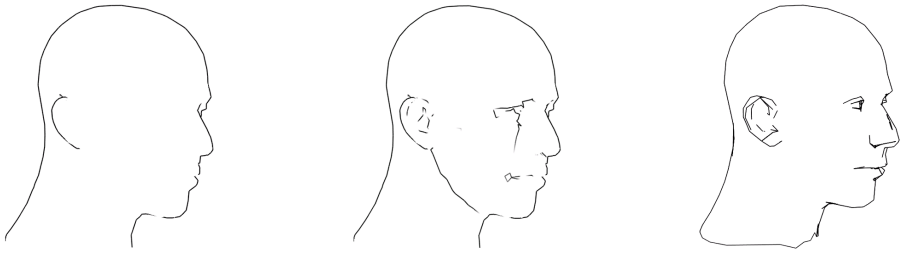
(a) (b) (c)

Fig. 10. A model with 3618 vertices and 7124 triangles. (a) The result with the silhouette rendered only. (b) The result with the silhouette and suggestive contour rendered. (c) The result with the silhouette, boundary and crease rendered after sharpening (our method).



(a) (b) (c)

Fig. 11. A model with 1760 vertices and 3516 triangles. (a) The result with the silhouette rendered only. (b) The result with the silhouette and suggestive contour rendered. (c) The result with the silhouette, boundary and crease rendered after sharpening (our method).



(a) (b) (c)

Fig. 12. A model with 689 vertices and 1355 triangles. (a) The result with the silhouette rendered only. (b) The result with the silhouette and suggestive contour rendered. (c) The result with the silhouette, boundary and crease rendered after sharpening (our method).

In our experiment, the choosing of μ and λ must satisfy that μ is larger than λ in the condition that μ and λ are very close. This means $\mu\lambda$ is much larger than $\mu - \lambda$, and this makes the term $\mu - \lambda$ in Eq.(7) works as a control of the above perturbation amount so that these amounts are very small, and thus keeps our result ideal.

7 Conclusions

In this paper, we suggested a feature-line rendering algorithm based on a sharpening filter. Using the sharpening filter to pre-process the model, we can perform non-photorealistic rendering of a 3D model not only with more feature details but also with less meaningless drawings than existing algorithms. Compared with existing algorithms, because of the removal of curvature estimation, the computation efficiency is also increased.

Acknowledgements. The authors would like to thank Doug DeCarlo of the department of Computer Science & Center for Cognitive Science, Rutgers University , for his offering of the suggestive contour source code which makes the comparison results for this paper feasible. Authors would also like to thank the Department of Computer Science of Duke University that publicized the 3D models that are used in our experiment.

References

1. Hertzmann, A., Zorin, D.: Illustrating smooth surface. In: Proceedings of SIGGRAPH 2000, ACM (2000)
2. DeCarlo, D., Finkelstein, A., Rusinkicwicz, S., Santella, A.: Suggestive contours for conveying shape. ACM Transactions on Graphics **22**(3(July)) (2003) 848–855
3. Sousa, M.C., Prusinkiewicz, P.: A few good lines: Suggestive drawing of 3d models. Computer Graphics Forum **22**(3) (2003) 381–390
4. Gooch, B., Gooch, A.: Non-photorealistic Rendering. A K Peters, Ltd. (2001)
5. Raskar, R., Cohen, M.: Image precision silhouette edges. In: Symposium on Interactive 3D Graphics 1999, ACM (1999)
6. Markosian, L., Kowalski, M.A., Trychin, S.J., Bourdev, L.D.: Real-time nonphotorealistic rendering. In: Proceedings of SIGGRAPH 1997, ACM (1997)
7. 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)
8. Decarmo, M.P.: Differential Geometry of Curves and Surface. Cambridge Univ. Press (1976)
9. Taubin, G.: Estimating the tensor of curvature of a surface from a polyhedral approximation. In: Proc. 5th International Conference on Computer Vision(ICCV-95). (1995)
10. Hameri, E., Shimshoni, I.: Estimating the principal curvature and the darboux frame from real 3d range data. In: Proc. International Symposium on 3D Data Processing Visualization and Transmission. (2002) 258–267
11. Neumann, L., Csebfalvi, B., Konig, A., Groller, E.: Gradient estimation in volume data using 4d linear regression. Computer Graphics Forum **19**(3) (2000) 351–357
12. Isenberg, T., Halper, N., Strothotte, T.: Stylizing silhouettes at interactive rates: from silhouette edges to silhouette strokes. Computer Graphics Forum **21**(3) (2002) 249–258
13. Page, D.L., Sun, Y., Koschan, A.F., Paik, J., Abidi, M.A.: Normal vector voting: crease detection and curvature estimation on large, noisy meshes. Graphical Models **64**(3-4) (2002) 199–229
14. Taubin, G.: Curve and surface smoothing without shrinkage. In: ICCV '95, IEEE (1995)