3ZC-7

Texture-Based Storage of Tracked Areas for Localized Stylization of View-Dependent Lines Extracted from 3D Models

Luis Cardona Suguru Saito
Graduate School of Information Science and Engineering, Tokyo Institute of Technology

1 Introduction

We propose a method to stylize individual lines by storing the ID of each line in a texture map as an area that can be used to recover its properties after arbitrary camera movement. As the camera position changes and view-dependent lines move across the surface, we track each line in order to extract and store the areas of the surface in which they appear. We call this region the *tracked area* of the line. Additionally, we provide a method for line segmentation of *occluding contours*[1] and *suggestive contours*[2, 3] as well as an user interface to select and stylize individual lines.

In our algorithm, we divide the contour lines at inflection points in order to segment them in a way that is better suited for localized stylization as well as to better approximate artist strokes. We also optimize the line tracking process by making use of the Gaussian curvature to separate the elliptic and hyperbolic areas of the surface. The results obtained in this paper show how each line only appears in a limited area of the surface corresponding to its *tracked area* and how we can store this data in a texture map. Finally, we highlight the achievements of our system for localized stylization of the contour lines of 3D models.

2 Algorithm

2.1 Line segmentation

The apparent curvature κ_{app} is the curvature of the contours in image-space[4]. κ_{app} is positive at convex parts of a contour, negative in the concave parts and zero at inflections. Koenderink[1] described the relationship between the apparent curvature κ_{app} and the Gaussian curvature K as follows:

$$\kappa_{app} = \frac{dK}{\kappa_r} \tag{1}$$

with d being the distance to the camera and κ_r the radial curvature.

Since the visible occluding contours only appear where κ_r is positive, both κ_{app} and K have the same sign. Consequently, we use K instead of κ_{app} to divide the contours at inflections(where $K = 0$).
In the case of *suggestive contours*, the lines only appear at hyperbolic regions(where $K < 0$). As result, we can not segment these lines in the same way as *occluding contours*.

2.2 Line tracking

One of the main problems we have to face is that for view-dependent lines, the shape and position of each line changes from frame to frame. Since they usually appear in different triangles compared to the previous frame, tracking becomes necessary to be able to recognize them as the same line.

In our method, we check the squared distance of both start and end of a given line with all terminal points of the lines of the next frame. For the line currently being checked, we define the points s^k and e^k as the start and end points respectively. A line in the frame k is considered the same as a line in frame $k-1$ if the following conditions are fulfilled:

- s^k and e^k are matched to the same line in frame $k-1$
- s and e are matched to different terminal points, i.e $s^{k-1} \neq e^{k-1}$

2.3 Gaussian curvature areas

We segment the surface using the sign of the Gaussian curvature to separate the elliptic and the hyperbolic parts of the model (Figure 1). Consequently, we can optimize the tracking process by checking only the lines contained in the same *Gaussian curvature area*. We make use of the connectivity data of the triangles as well as the zero-crossings of K to determine the ID at each vertex. Afterwards, we pass the K zero-crossing data to a fragment shader to assign an unique color for each ID. This shader also separates the IDs of the triangles containing a zero-crossing of K by assigning different colors to each side of the zero-crossing. Additionally, since the Gaussian curvature is view-independent, the segmentation process can be pre-computed (Figure 2).

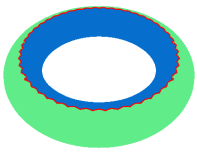

Figure 1: Gaussian curvature areas separated by the boundary where $K = 0$ (red)

2.4 Tracked areas

As the camera position changes, view-dependent lines move across the surface until they eventually merge with other lines or completely disappear. Therefore, contour lines can't be tracked at all viewpoints and by consequence, it is essential to be able to store and retrieve their IDs after arbitrary camera movements. Since contour lines can only appear around a limited area, we store their IDs in a color-coded texture map containing the regions of the surface where each line has previously appeared (Figure 4). For a given line, we retrieve the ID from the texture map only if both of its terminal points

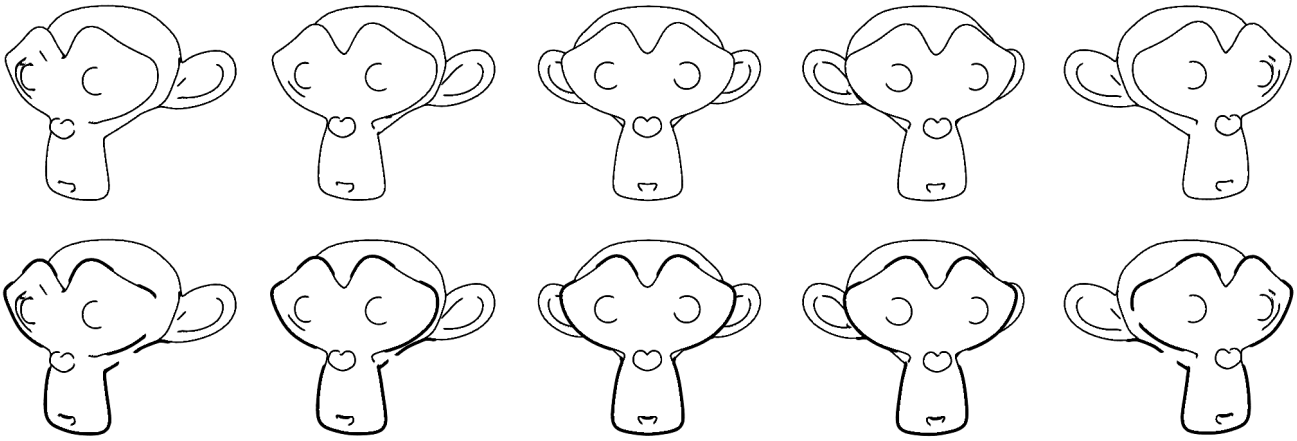

Figure 3: A 3D model without stylization (up) and with localized stylization (down) as the viewpoint changes

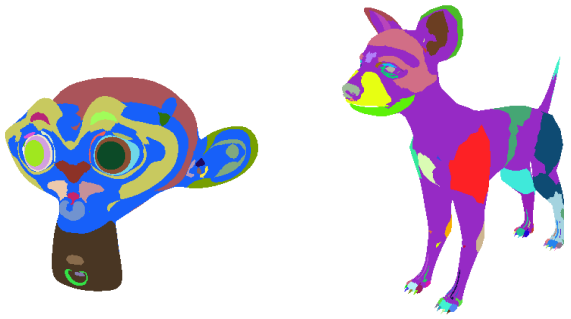

Figure 2: Pre-computed segmentation using the sign of the Gaussian curvature K

and its center point are in a the same *tracked area*. Since these areas can be used to identify each line, pre-computing all the areas could eventually make tracking unnecessary.

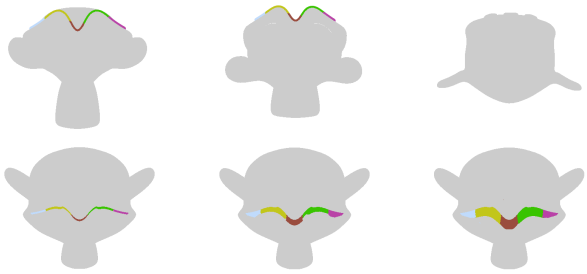

Figure 4: Five lines of a 3D model as the viewpoint changes(up) and the evolution of the corresponding tracked areas

3 Results

In our implementation, the user can select each line and change its properties. The process to stylize one line only needs to be done for one viewpoint because its properties are preserved as the camera moves. As result, compared to previous stylization approaches[5, 6], our system not only enables the user to stylize individual lines but also frees him from the tedious process of customizing them for each viewpoint. For an experienced user, the whole process of stylizing the contour lines of the model shown in Figure 3 can be done in about one minute.

4 Conclusion and Future Work

We have proposed a method to store in a texture map the IDs of contour lines for localized stylization of both *occluding contours* and *suggestive contours*. However, future approaches should implement more robust tracking methods to prevent incorrect matches between two different lines. Our system only provides control over the line thickness and therefore we should also extend the range of customizable properties. In addition, given that different lines may have common regions where they can appear, we should add support for overlapping *tracked areas*. Finally, the correlation between the lines in object-space and their image-space counterpart could be improved to better match how artists draw them in real line drawings.

References

[1] J.J. Koenderink et al. What does the occluding contour tell us about solid shape. *Perception*, 13(3):321–330, 1984.

[2] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Trans. Graph.*, 22(3):848–855, July 2003.

[3] Doug DeCarlo, Adam Finkelstein, and Szymon Rusinkiewicz. Interactive rendering of suggestive contours with temporal coherence. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, NPAR '04, pages 15–145, New York, NY, USA, 2004. ACM.

[4] Szymon Rusinkiewicz, Forrester Cole, Doug DeCarlo, and Adam Finkelstein. Line drawings from 3d models. In *ACM SIGGRAPH 2008 classes*, SIGGRAPH '08, pages 39:1–39:356, New York, NY, USA, 2008. ACM.

[5] Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes, and Adam Finkelstein. Wysiwyg npr: drawing strokes directly on 3d models. *ACM Trans. Graph.*, 21(3):755–762, July 2002.

[6] Stéphane Grabli, Emmanuel Turquin, Frédo Durand, and François X. Sillion. Programmable rendering of line drawing from 3d scenes. *ACM Trans. Graph.*, 29(2):18:1–18:20, April 2010.