論文 / 著書情報
Article / Book Information

Title	Customizable Strokes for Localized Stylization of View-Dependent Lines Extracted from 3D Models
Authors	Luis Cardona, Suguru Saito
Citation	Proceedings of the 2013 International Conference on Cyberworlds, pp. 140-146
Pub. date	2013, 12
DOI	http://dx.doi.org/10.1109/CW.2013.62
URL	http://www.ieee.org/index.html
Copyright	(c)2013 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Note	This file is author (final) version.

Customizable Strokes for Localized Stylization of View-Dependent Lines Extracted from 3D Models

Luis Cardona
Tokyo Institute of Technology:
Graduate School of Information Science and Engineering
Tokyo, Japan
Email: luis@img.cs.titech.ac.jp

Suguru Saito
Ochanomizu University:
Graduate School of Humanities and Science /
Tokyo Institute of Technology:
Graduate School of Information Science and Engineering
Tokyo, Japan
Email: suguru@is.ocha.ac.jp

***Abstract*—We propose a method to stylize individual lines and preserve their properties as the viewpoint is modified. 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. Each line has an unique reference that we use to check an indexed table from which we can recover its stylization after arbitrary camera movements. We call this region the *tracked area* of the line. Additionally, we provide a method for line segmentation of *occluding contours*. In our algorithm, we divide the contour lines at inflection points in order to segment them in a way that is 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. In this paper we show how each line only appears in a limited area of the surface corresponding to its *tracked area*. We also discuss how to eliminate the over-segmentation caused by numerical instability of the approximations in polygonal surfaces as well as how smoothing can be used in order to deal with highly detailed models. Finally, we highlight the achievements of our system for *localized stylization* of the contour lines of 3D models.**

***Keywords*-NPR; line drawings; stylization; segmentation; tracking;**

I. Introduction

A. Background and purpose

Computer-generated 3D models have been extensively used in applications that try to replicate realistic imagery. However, there is another branch of computer graphics called NPR (Non-photorealistic rendering), that focuses on how to imitate the principles of abstraction used by artists. NPR research covers different topics like line extraction algorithms, stylization and animation coherence. Yet, there is still many problematics to solve in order to generate imagery with the same degree of abstraction and stylization as human-made drawings.

As stated in a recent study[2], most of the lines drawn by artist to convey 3D shape can be explained by some of the most commonly-known line extraction algorithms. However, artists often make global decisions such as the omission of implicit lines or stylistic decisions which tend to depart from realism. Therefore, there is a need for flexible user interfaces which enables designers to customize the appearance of drawings as well as to emphasize or omit some of its details.

Usually line drawings have a consistent style for all the scene but artists also include subtle sytilization differences in each stroke. However, algorithms for extracting lines only provide local information like the location of lines or curvature data for each vertex of the mesh. Our system performs a segmentation of the extracted lines to differentiate individual continuous lines and divide them at inflection points. Once a line is selected, the user can customize it by changing its shape. We also provide an user interface to easily select each of them individually or select every line inside a *Gaussian curvature area*.

The main problem to solve in order to successfully achieve *localized stylization* of individual lines is how to recognize each of them as the same entity as the viewpoint changes.

B. Related work

Various methods have been proposed in order to extract lines from 3D models. Image-space algorithms focus on image processing methods such as edge detection to extract the lines at each pixel of the output image[3], [4], [5], [6]. This type of algorithm generates quite convincing images but suffers from noise problems and are unsuitable for further stylization.

Object-space algorithms are based on the field of differential geometry which analyzes the properties of curves and surfaces. Previous research has successfully characterized different type of lines by making use of

derivatives of different order. The main contributions in this category are occluding contours[1], suggestive contours[7], [8], ridges and valleys[9] and apparent ridges[10].

Recently, some researches have focused on how to propagate the parameters of the lines in order to maintain the coherence of stylized strokes as the viewpoint changes or the 3D model is animated[11], [12], [13], [14], [15].

Previous papers have stated the need for user interfaces to provide better control of the rendering styles[16], [17]. These systems are useful to quickly define the style of the whole scene but do not provide the means to customize individual details.

Finally, 3D model segmentation algorithms have focused in how to divide surfaces into meaningful parts consistent with how humans perceives the shape of objects[18].

II. Algorithm

In this paper, we make the assumption that the inflections of contour lines have high probability to be chosen by artists as terminal points. We will start by describing the relationship between inflections points and surface curvature.

The apparent curvature κ_{app} is the curvature of the contours in image-space[19]. κ_{app} is positive at convex parts of contours, 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 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*.

A. Gaussian curvature areas

We segment the surface using the sign of the Gaussian curvature by separating the elliptic ($K > 0$) and the hyperbolic ($K < 0$) parts of the model (Fig. 1). We call these regions *Gaussian curvature areas*. The benefit of the Gaussian curvature segmentation is that not only it enables us to divide the line tracking problem in similar subproblems but also it is directly related to the inflection points of contour lines. In the case of *suggestive contours*, the zero-crossings of K can not be related to inflection points. However, the *Gaussian curvature areas* are still useful to apply different styles and thresholds on different areas.

We make use of the connectivity data of the triangles as well as the zero-crossings of K to determine at which area each vertex belongs to. Afterwards, we pass the K zero-crossing data to a fragment shader in order to assign an unique color for each area. This shader also builds the boundaries between the areas by assigning different colors to each side of the triangles containing a zero-crossing of K. It should also be noted that, in the case of non-animated models, the Gaussian curvature is view-independent, i.e. constant, which means that the segmentation can be pre-computed. Additionally, we added the functionality of manually merge areas because the user may want to decide if a line should be segmented at the boundary between two areas or not.

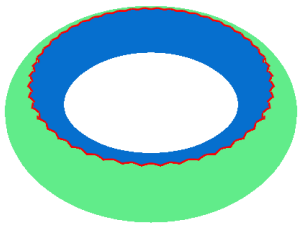

Figure 1. Gaussian curvature areas separated by the boundary where $K = 0$ (red)

1) Smoothing: Meshes with high level of detail usually result in a highly fragmented segmentation. Depending on the drawing style we want to replicate, we may want to ignore some of the surface detail in favor of a more simplified line segmentation.

Smoothing is commonly used for noise removal of rough meshes, e.g. 3D scan reconstructions of physical objects. A frequently used method for smoothing meshes is Laplacian smoothing. Laplacian smoothing can be thought as the equivalent of lowpass filtering in signal processing. Desbrun et.al. [20] describes the discrete approximation of the Laplacian as a weighted sum of the one-ring neighbors of a vertex:

$$\mathcal{L}(x_i) = \sum_{j \in N} w_{ij}(x_j - x_i) \tag{2}$$

where x_j are the neighbors of the vertex x_i and N is the set of neighbor vertices.

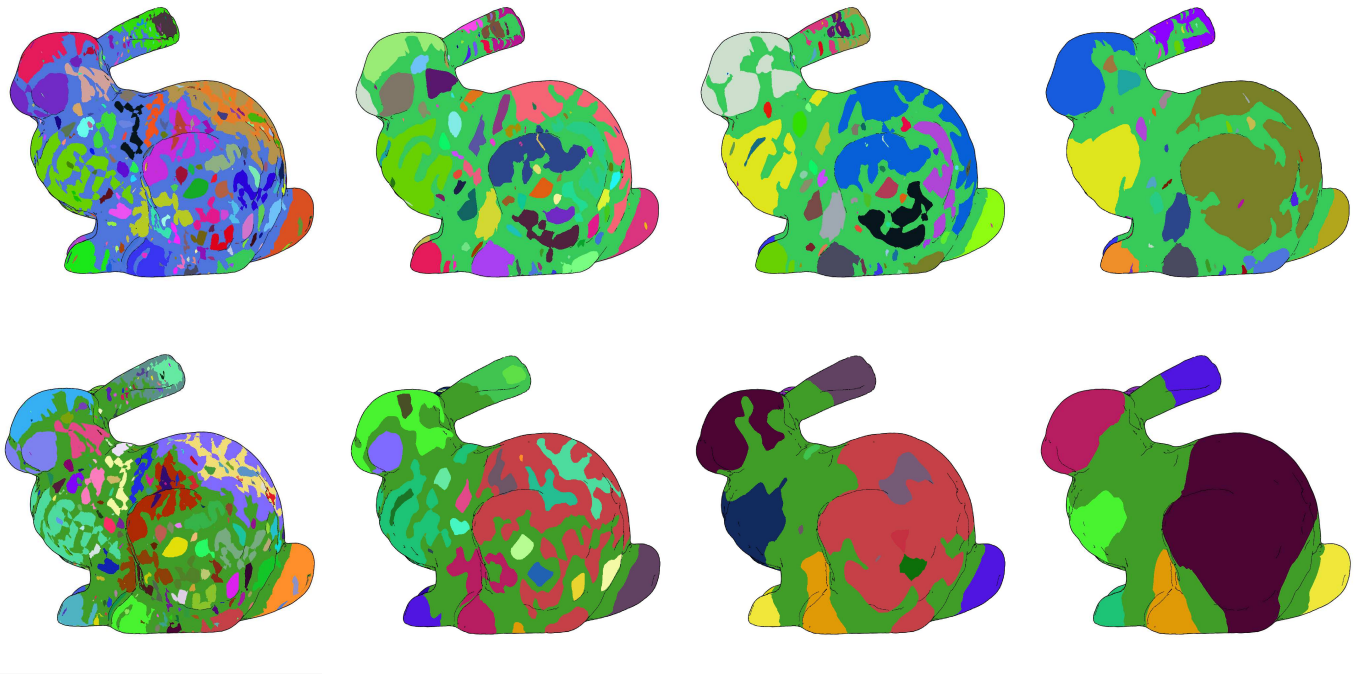

Figure 2. Original segmentation and Laplacian smoothing applied to the vertices positions (top) and the Gaussian curvature (bottom) for 5, 25 and 100 iterations.

For a given vertex, the most simple of the proposed approximations defines the Laplacian as the vector to the barycenter of its one-ring neighbors. This is also known as the *umbrella operator* and uses equal weights:

$$w_{i,j} = \frac{1}{m} \tag{3}$$

where m is the number of neighbors vertices.

The smoothed vertex positions can then be calculated as follows:

$$P_i(n+1) = P_i(n) + \frac{1}{m} \sum_{j \in N} (P_j(n) - P_i(n)) \tag{4}$$

Alternatively, Laplacian smoothing can also be directly applied to the values of the Gaussian curvature as follows:

$$K_i(n+1) = K_i(n) + \frac{1}{m} \sum_{j \in N} (K_j(n) - K_i(n)) \tag{5}$$

The results of both smoothing methods (Fig. 2) shows that as the number of iterations increases, the number of segmented areas decreases. Applying Laplacian smoothing directly to the Gaussian curvature is considerably faster because no recalculation of normals and curvatures is needed. Furthermore, the direct method tends to eliminate the over-segmentation caused by numerical instability. In comparison, the method that includes the recalculation yields better results because it does not modify as much the shape of the boundaries of the areas but the numerical instability of the approximation of K still produces unwanted areas. However, this problem can be solved by merging the unwanted areas as described in the following section.

2) Numerical instability: For polygonal surfaces, K is just an approximation which can introduce undesirable artifacts (especially where the minimum curvature κ_2 is around 0). In our case, we want to get rid of the areas which results in over-segmentation of the contour lines, i.e. the areas which are almost flat or are only bent in one direction.

We make use of a modified Sigmoid function to constrain the infinite values of the Gaussian curvature. The Sigmoid function is defined as:

$$s_\alpha(x) = \frac{1}{1 + e^{-\alpha x}} \tag{6}$$

where α defines the shape of the Sigmoid function.

The following modified version takes values between 0 and 1 for positive values of x:

$$S_\alpha(x) = \frac{2}{1 + e^{-\alpha x}} - 1 \tag{7}$$

For each area, we calculate the average κ_2 of all the vertices contained inside it:

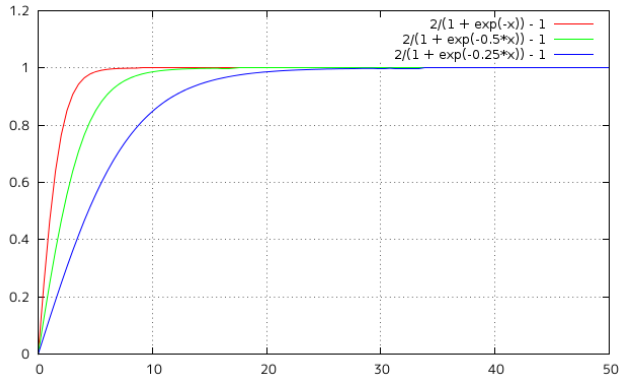

Figure 3. Modified Sigmoid function with different values of α

$$\bar{\kappa_2} = \frac{1}{N} \sum_{i \in V} S_m(|\kappa_{2_i}|) \tag{8}$$

where V is the set of vertices of the area.

Finally, we merge the areas which have only one adjacent area and fulfill the following condition (Fig. 4):

$$\bar{\kappa_2} < \delta \tag{9}$$

where δ is an user-defined threshold.

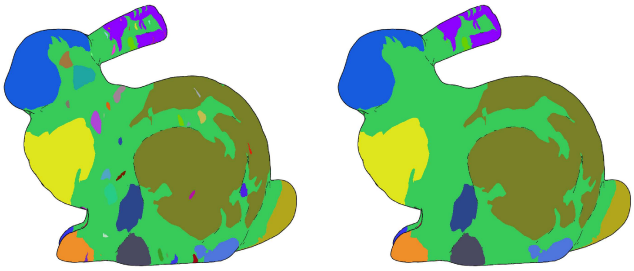

Figure 4. The unwanted areas produced by numerical instability are eliminated

B. Generation of Tracked Areas

One of the main problems we face is that for view-dependent lines, the shape and position of each line changes depending on the viewpoint. Consequently, recognizing each contour line from frame to frame is essential to be able to stylize individual lines.

1) Line tracking: 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 $s^{k-1} \neq e^{k-1}$

Additionally, we sample a number of equidistant points along each line and check the similarity of the matched lines using the Hausdorff distance:

$$h(A,B) = \max_{a \in A}(\min_{b \in B}(d(a,b)) \tag{10}$$

$$H(X,Y) = max(h(X), h(Y)) \tag{11}$$

With X and Y being the sets of sampled point of the two lines being checked.

Therefore, in order to prevent mismatches we add the following condition:

$$H(X,Y) < \delta \tag{12}$$

With δ being a distance threshold.

As previously stated, we decided to segment the lines at inflections points. This actually corresponds to dividing the lines as they cross the boundaries between *Gaussian curvature areas* in object-space. Therefore, we can optimize the tracking process by checking only the lines contained in the same *Gaussian curvature area*.

2) Tracked areas: As the camera position changes, view-dependent lines move across the surface until they eventually merge with other lines or completely disappear. Consequently, not all contour lines can be tracked at every viewpoint. It is then essential to be able to store and retrieve their references after arbitrary camera movements. Since contour lines can only appear inside a limited area, we store their references in a color-coded texture map containing the regions of the surface where each line has previously appeared (Fig. 5). We call these regions the *tracked areas*. It should be noted that the *tracked areas* can be considered as subsets of the *Gaussian curvature areas* (Fig. 6).

For any given line, we retrieve the reference from the texture map only if both of its terminal points and sampled internal points are in the same *tracked area*. The reference can then be used to recover the line stylization from the properties table (Table I). In our algorithm, the purpose of line tracking is limited to building the *tracked areas* corresponding to each contour line. Since these areas can be used to identify each line, pre-computing all the *tracked*

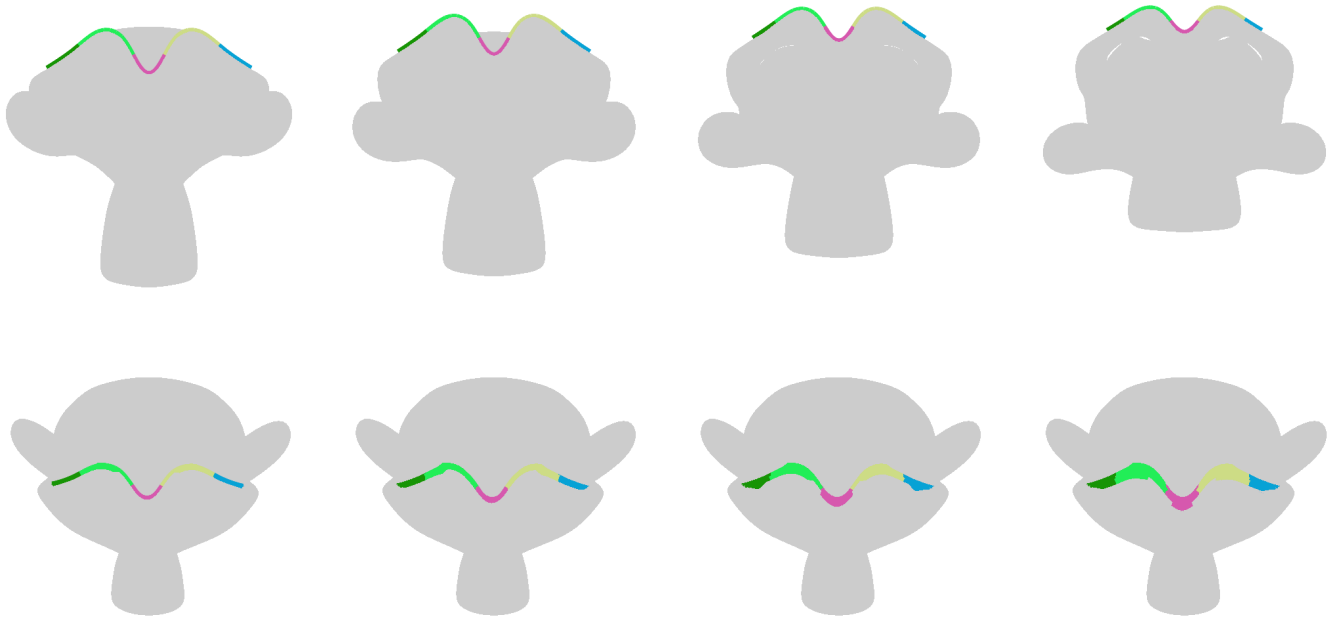

Figure 5. Five lines of a 3D model as the viewpoint changes (up) and the evolution of their tracked areas (down)

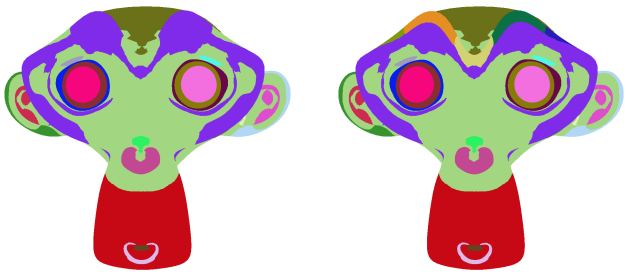

Figure 6. The original segmentation (left) and the tracked areas of 5 lines added as a subset of their corresponding Gaussian curvature area (right)

Reference	Line properties
0	ω, α_T, α_L, ...
..	..
..	..
..	..
n	ω, α_T, α_L, ...

Table I
THE PROPERTIES TABLE INDEXED BY EACH LINE REFERENCE

areas would allow us to just retrieve the reference directly from the color-coded texture map, making further tracking unnecessary.

C. Line rendering

We chose a commonly used method for line rendering that extrudes triangles strips along the zero-crossings in the normal direction. As opposed to other line primitive based methods, this rendering method enable us to change the width along lines as much as we want. We also added length dependent width and tapering[21] using the same modified Sigmoid function defined in (7) of section II-A2. We can obtain a variety of line shapes by changing the α coefficients of the Sigmoid functions. We can then calculate the width at a certain position of a line as follows:

$$W = \omega S_{\alpha_L}(L) S_{\alpha_T}(l) \tag{13}$$

With ω being the user-defined base width, L the total length of the line and l a scalar taking normalized values increasing as we move away from the nearest terminal point.

III. RESULTS

In our implementation, the user can select each line separately and change its properties or select a Gaussian curvature area and modify at once all the lines contained inside it. The process to stylize a line only needs to be done for one viewpoint because its properties are preserved as the camera moves. As result, our system not only frees the user from the tedious process of customizing all lines for every viewpoint and but also as opposed to previous stylization approaches[16], [17], we are able to stylize each line individually. For an experienced user, the whole process of stylizing the contour lines of the models shown in Fig. 7, 8 and 9 can be done in a few minutes. Additionally, we showed how unwanted areas caused by numerical instability can be

eliminated and also compared two smoothing methods for highly detailed models.

IV. Conclusion

We have proposed a method for *localized stylization* of *occluding contours* that preserves the properties of the lines as the viewpoint change. However, future approaches should implement more robust tracking methods to prevent incorrect matches. Our system only provides control over the line shape and therefore we should extend the range of customizable properties. In addition, given that different lines may appear in common regions, we should add support for overlapping *tracked areas*. We should also note that our algorithm partially supports *localized stylization* of *suggestive contours* but it is still lacking compared to *occluding contours*. Finally, the segmentation of contour lines may be extended to consider other important points like the extremas of contour points as possible candidates for terminal points.

References

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

[2] Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. Where do people draw lines? *Commun. ACM*, 55(1):107–115, January 2012.

[3] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. *SIGGRAPH Comput. Graph.*, 24(4):197–206, September 1990.

[4] Philippe Decaudin. Cartoon looking rendering of 3D scenes. Research Report 2919, INRIA, June 1996.

[5] A. Hertzmann. Introduction to 3d non-photorealistic rendering: Silhouettes and outlines. *Non-Photorealistic Rendering. SIGGRAPH*, 99, 1999.

[6] Yunjin Lee, Lee Markosian, Seungyong Lee, and John F. Hughes. Line drawings via abstracted shading. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

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

[8] 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.

[9] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph.*, 23(3):609–612, August 2004.

[10] Tilke Judd, Frédo Durand, and Edward Adelson. Apparent ridges for line drawing. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[11] Robert D. Kalnins, Philip L. Davidson, Lee Markosian, and Adam Finkelstein. Coherent stylized silhouettes. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 856–861, New York, NY, USA, 2003. ACM.

[12] Pierre Bénard, Forrester Cole, Aleksey Golovinskiy, and Adam Finkelstein. Self-similar texture for coherent line stylization. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, NPAR '10, pages 91–97, New York, NY, USA, 2010. ACM.

[13] Bert Buchholz, Noura Faraj, Sylvain Paris, Elmar Eisemann, and Tamy Boubekeur. Spatio-temporal analysis for parameterizing animated lines. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, NPAR '11, pages 85–92, New York, NY, USA, 2011. ACM.

[14] Kevin Karsch and John C. Hart. Snaxels on a plane. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, NPAR '11, pages 35–42, New York, NY, USA, 2011. ACM.

[15] Pierre Bénard, Jingwan Lu, Forrester Cole, Adam Finkelstein, and Joëlle Thollot. Active strokes: coherent line stylization for animated 3d models. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, NPAR '12, pages 37–46, Aire-la-Ville, Switzerland, Switzerland, 2012. Eurographics Association.

[16] 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.

[17] 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.

[18] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. A benchmark for 3d mesh segmentation. *ACM Trans. Graph.*, 28(3):73:1–73:12, July 2009.

[19] 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.

[20] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 317–324, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[21] Suguru Saito, Akane Kani, Youngha Chang, and Masayuki Nakajima. Curvature-based stroke rendering. *Vis. Comput.*, 24(1):1–11, November 2007.

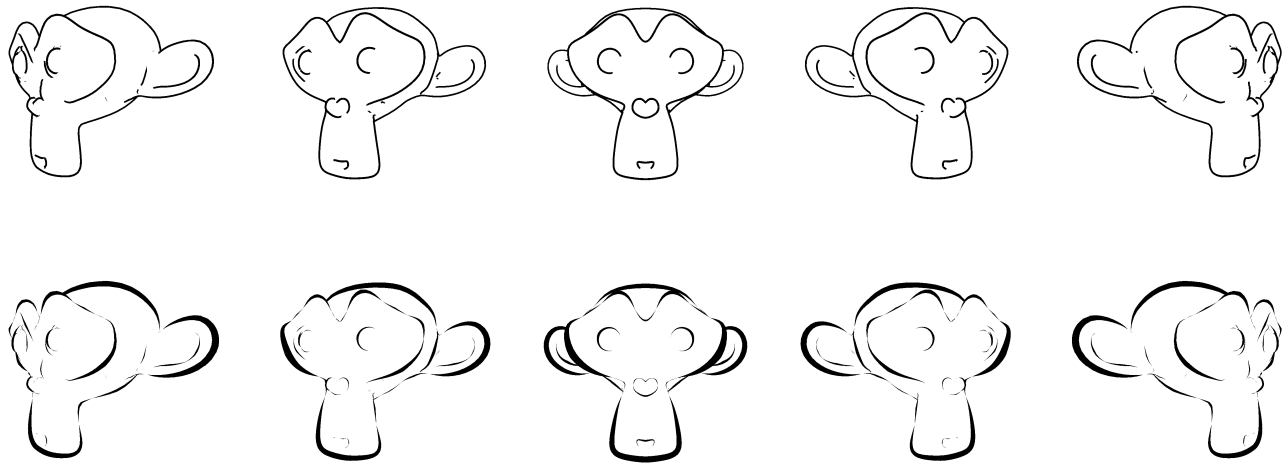

Figure 7. A monkey 3D model without stylization (up) and with *localized stylization* (down) as the viewpoint changes

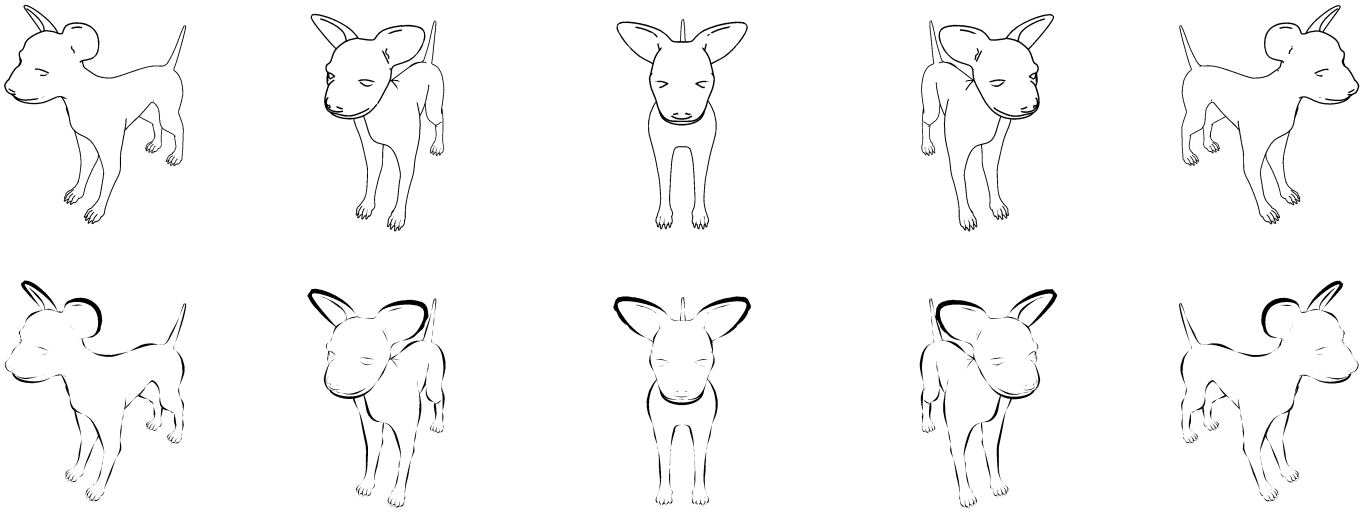

Figure 8. A dog 3D model without stylization (up) and with *localized stylization* (down) as the viewpoint changes

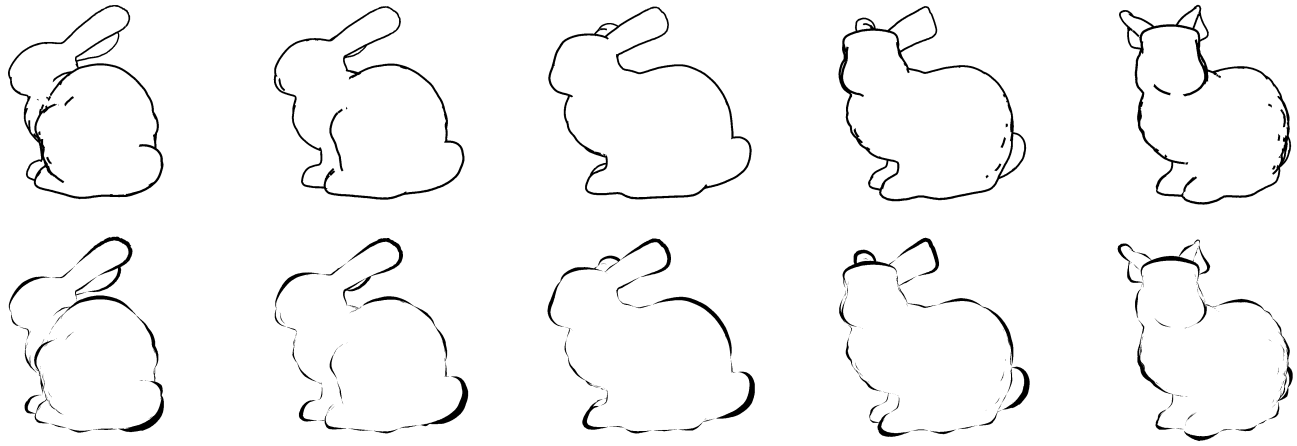

Figure 9. Stanford bunny without stylization (up) and with *localized stylization* (down) as the viewpoint changes