Coherent Stylized Silhouettes

Robert D. Kalnins Philip L. Davidson Lee Markosian Adam Finkelstein

Princeton University

Abstract

We describe a way to render stylized silhouettes of animated 3D models with temporal coherence. Coherence is one of the central challenges for non-photorealistic rendering. It is especially difficult for silhouettes, because they may not have obvious correspondences between frames. We demonstrate various coherence effects for stylized silhouettes with a robust working system. Our method runs in real-time for models of moderate complexity, making it suitable for both interactive applications and offline animation.

Keywords: Silhouettes, animation, non-photorealistic rendering.

1 Introduction

Silhouettes play a critical role in our visual interpretation of 3D shape. Artists and designers therefore often emphasize silhouettes, either by drawing them explicitly or by contrast enhancement across silhouette boundaries. Indeed, the most minimal hand-drawn image of a 3D scene will often be composed largely of marks representing silhouettes and sharp features.

Researchers have developed a number of algorithms to find and render silhouettes from 3D models. One branch of this research addresses *stylized* silhouettes, which are drawn with strokes that wiggle, vary in width and texture, or resemble in other ways strokes drawn by hand with natural media. Stylization is characteristic of non-photorealistic rendering (NPR) applications, for which many effective general arguments have already been made (e.g., [Durand 2002; Gooch and Gooch 2001; Strothotte and Schlechtweg 2002]). Stylized silhouettes can, for example, suggest surface texture, give an organic feeling to an overly-mechanical shape, or simply annotate features of a model such as the hidden silhouettes denoted by dashed lines in Figure 1b.

A key challenge for non-photorealistic rendering is to provide frame-to-frame coherence, so that strokes adapt smoothly over time to changes in the animated scene (Figure 1d). Temporal coherence is especially challenging for silhouettes because they may not have obvious inter-frame correspondence as they evolve over time. Furthermore, the goal of providing coherence on the 3D shape conflicts in general with another goal for stylized strokes: to look hand-drawn, stylization should map onto strokes parameterized by 2D arc-length. For example, it wouldn't do if the brush wiggles were all squished into one end of the stroke. A final challenge is to make these methods work at interactive rates, enabling real-time applications such as interactive illustrations and games.

Permission to make digital/hard copy of part of all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
© 2003 ACM 0730-0301/03/0700-0856 $5.00

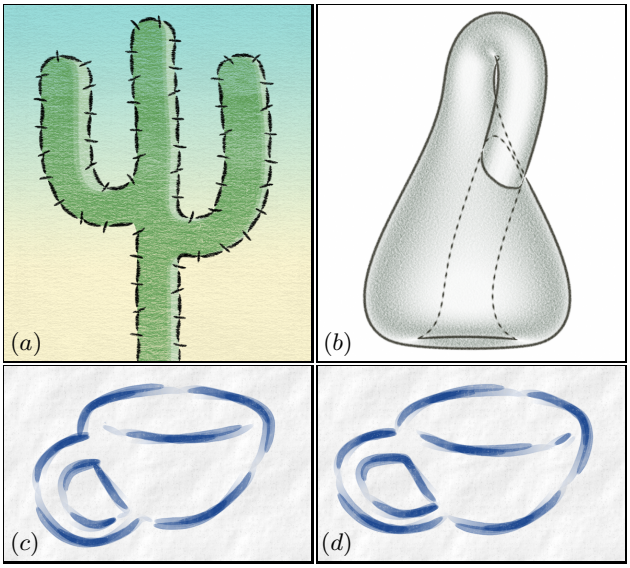


Figure 1: Stylized silhouettes can (a) suggest geometric detail, (b) reveal hidden features, and (c) impart a rich hand-drawn look. In panel (d), strokes from (c) adapt coherently to the new view.

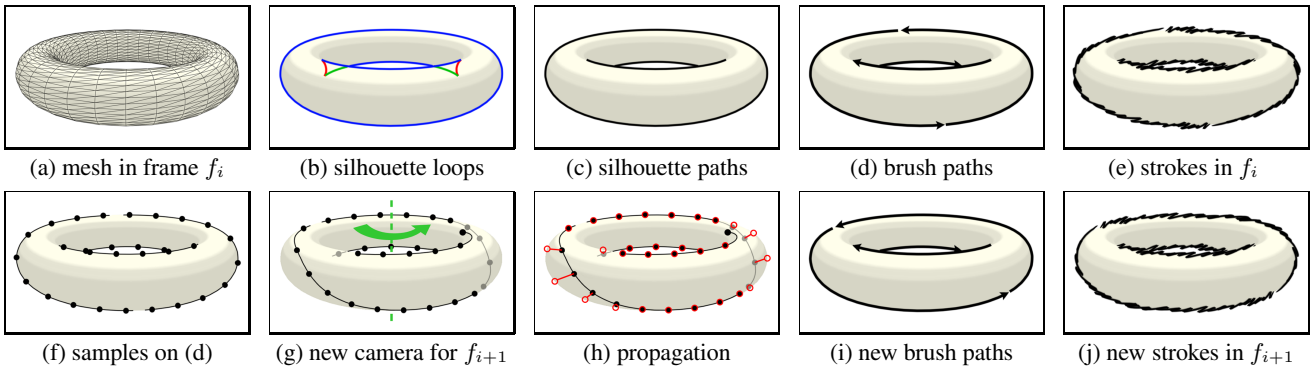
In this paper we explain how to interactively render stylized silhouettes with robust frame-to-frame coherence. We address the coherence challenge by describing a new algorithm for propagating the parameterization from strokes in one frame to strokes in the next. The method allows a designer to balance between the goals of coherence on the 3D shape and 2D arc-length parameterization.

2 Related Work

A number of researchers have developed strategies for explicitly drawing stylized silhouettes from a 3D model [Zeleznik et al. 1996; Masuch et al. 1997; Markosian et al. 1997; Bremer and Hughes 1998; Gooch et al. 1999; Hertzmann and Zorin 2000; Lake et al. 2000; Northrup and Markosian 2000; Isenberg et al. 2002]. Other methods leverage graphics hardware to render silhouettes without handling them explicitly [Gooch et al. 1999; Raskar 2001], but these admit only minimal stylistic control.

Little attention has been given to the temporal coherence problem for stylized silhouettes. Masuch et al. [1998] describe a solution to this problem for the setting when natural silhouette arc-length parameterization remains consistent, but this applies only under simple conditions (e.g. rotating about a cylinder).

A more general approach to this problem is given by Bourdev [1998]. He proposes to parameterize silhouettes based on parameter samples from nearby strokes in the previous frame. A simple version of Bourdev's scheme is implemented by Kalnins et al. [2002]. Inspired by Bourdev's sampling method, we present a new approach that provides significant improvements in robustness and offers control over the tradeoff between arc-length parameterization in 2D and coherence on the 3D shape.



(a) mesh in frame f_i (b) silhouette loops (c) silhouette paths (d) brush paths (e) strokes in f_i

(f) samples on (d) (g) new camera for f_{i+1} (h) propagation (i) new brush paths (j) new strokes in f_{i+1}

Figure 2: Overview of process. *Upper row:* stylization of silhouettes in frame f_i. *Lower row:* propagation to subsequent frame f_{i+1}.

3 Overview

This section describes some of the design goals for rendering stylized silhouettes with coherence and provides an overview of the key steps in our system. Our implementation works on triangle meshes, but the algorithms introduced here should work for any surface representation on which silhouettes can be found.

We assume that each mesh is assigned a stylization based on application-specific aesthetic concerns. The stylization is defined either as a set of brush marks or as a texture image. The mesh may be separated into patches, each of which receives a single stylization that is applied uniformly to all silhouettes. Optionally, invisible silhouettes and creases (or other hand-tagged features) may each receive their own stylization to augment the rendering.

Before delving into our design goals, we define a few terms used throughout the paper. First, *silhouette loops* separate front- and back-facing portions of the mesh; they consist of polylines that either form complete loops on the mesh or begin and end on mesh creases or boundaries (Figure 2b). Second, *silhouette paths* are visible or invisible portions of silhouette loops, depending on which we are interested in rendering (Figure 2c). Finally, *strokes* are the rendering primitives used in our system (see Section 3.2).

3.1 Design Objectives

The primary goal of our system is to draw stylized silhouettes using temporally coherent strokes. Strokes have two properties that affect coherence: the *path* in the image plane over which the stroke is applied, and the *parameterization* that defines how stylization (e.g., wiggles or texture) is mapped onto that path. Typically, silhouette paths in the image plane enjoy a natural coherence. Therefore, the key to achieving our goal is to provide coherence for the *parameterization* of silhouettes. There are two questions we must address: first, what is the unit of stylization to be parameterized? Second, how do we assign coherent parameterization to those units? This paper offers answers to these questions that make it possible to render stylized silhouettes with robust coherence. These answers are the major contributions of our work.

There is a seemingly-obvious answer to the first question: one can parameterize individual silhouette paths. Both Bourdev [1998] and Masuch et al. [1998] investigated this approach independently. This strategy works well for some sequences of frames. However, coherence will be broken whenever two silhouette paths in one frame merge into a single path in the next frame. There will be a visual "pop" if single parameterization is assigned, unless by chance the two paths have matching parameterization where they abut. To avoid this, Masuch et al. caused the two abutting paths to match in limited cases by enforcing coherence along the underlying silhouette loop. While this strategy prevents some popping, in general it exacerbates a problem arising whenever a single parameterization is maintained for long silhouette paths. Changes in foreshortening on one area of the model can influence other areas far away, leading to "swimming" artifacts wherein the stylization appears to slide along the visible silhouette.

To address the popping problem, we do *not* require a single parameterization per silhouette path. Instead, we enforce coherence for separate portions of the silhouette path that we call *brush paths* (Figure 2d). Intuitively, a brush path may be thought of as the path of a single stroke, though in practice many strokes may be applied in concert to a single brush path (e.g. for dashed lines). Section 4 provides a detailed description of how silhouette paths are divided into brush paths. Using brush paths also ameliorates the swimming problem because stylization is more localized.

Having identified brush paths as the unit of coherence, we now address the second question above: how do we parameterize them? Section 1 mentions two competing goals: stroke coherence on the 3D shape versus uniform 2D arc-length parameterization. In any particular scene the semantics of the stylization governs the relative priority of these two goals. Thus, we would like to provide the designer with controls for balancing between them. Bourdev [1998] also identifies this objective and offers a technique based on weighted averages to address it. However, his approach suffers from visual artifacts we will describe in Section 4.2. Section 5 provides a detailed discussion of our strategy for balancing between these goals in a robust, controllable manner.

3.2 The System

To find the silhouettes, determine visibility, and render strokes we largely follow procedures outlined by Kalnins et al. [2002]. We now briefly recap the key steps. (See Figures 2a-e.)

Extraction. Any of several methods can be used to extract silhouettes from 3D models with the goal of drawing strokes along them [Isenberg et al. 2002; Northrup and Markosian 2000; Hertzmann and Zorin 2000]. Isenberg et al. [2003] provide an excellent overview of the tradeoffs involved in choosing among the various silhouette extraction techniques.

Visibility. We determine visibility of silhouettes using an *ID image*, as described by Northrup and Markosian [2000]. The same ID image plays a role in the propagation of parameter samples from one frame to the next (described in Section 4). Regardless of the chosen visibility algorithm, the result of this phase should be a set of 3D silhouette paths, processed for visibility and resampled to screen-space resolution. The in-frustum occluded paths may optionally be drawn in some special style as in Figure 1b. However, this involves additional framebuffer techniques for constructing the ID image [Rossignac and van Emmerik 1992].

Rendering. A brush path may be rendered with multiple strokes, for example to produce a dashed line or to depict the spikes of a cactus (Figure 1). We use either of two mechanisms to apply strokes over brush paths. One method constructs a single stroke as a triangle strip following each brush path, and applies to it a (periodic) texture map suggesting one or more marks. In this case, the texture coordinate along the strip is given by the parameterization of the brush path. This strategy is useful for dotted or dashed lines and large paint brush strokes. The second method is to programmatically construct one or more strokes over the brush path, using triangle strips with geometric offset lists as described by Kalnins et al. [2002]. This strategy permits finer control over qualities such as tapering of individual strokes. In either case, the stroke pattern is repeated if needed to cover long brush paths.

4 Brush Paths

During rendering, stylization is mapped onto the brush path via a parameter t. To achieve temporal coherence, we propagate samples of t from one frame to the next. This section describes how we generate new brush paths from the samples. Section 5 describes how these new brush paths are parameterized from the samples. An overview of this process is shown in the lower row of Figure 2.

4.1 Sample Propagation

At each frame f_i, we take a set of evenly spaced samples from each brush path. Each sample contains a triplet: its location in 3D on the mesh, its brush path ID, and the parameter t that we wish to propagate. We record the 3D position as a barycentric location on a mesh triangle in order to track the shape during animation. In subsequent frame f_{i+1}, we seek to register the samples from f_i against the silhouette paths of f_{i+1} to propagate parameterization.

Our goal is that a sample should arrive at a silhouette path that is nearby in screen space and nearby on the mesh. Of course, from f_i to f_{i+1} there may be an arbitrary camera motion or mesh animation. However for ordinary coherent animation these motions will be moderate. As illustrated in Figure 2g, our first step is to project each sample from its 3D location to the image plane given the camera of f_{i+1}. Call its projection x.

To find a brush path near the sample in f_{i+1}, we search in the ID image near x. The ID image (Figure 3) is an item buffer that encodes individual IDs of silhouette loops and faces of the mesh. We check a one-pixel-radius neighborhood of x and step repeatedly by one pixel along the screen-space projection of the original surface normal until we find the ID of a silhouette path or else exceed a distance threshold. We search along the projected normal because this direction tends to point toward the silhouette. If more than one silhouette path is found, we choose the path intersection closest to the original location in 3D. We observe that because silhouettes move coherently across the mesh, most searches succeed immediately in the neighborhood of x, or else within just a few pixels. (This is always true for creases, because they remain fixed on the model.) A threshold of 6 pixels was found sufficient at 640x480 to propagate parameter samples under most camera changes. In practice, when the camera changes are extreme enough as to thwart propagation, we find that any coherence, or lack thereof, is imperceptible anyway. Finally, when a silhouette path is found, we record a *vote* (s, t) – the association of the parameter sample t with the arc-length parameter s on the silhouette path found by the search. The next two sections describe how these votes are used to generate and then parameterize brush paths.

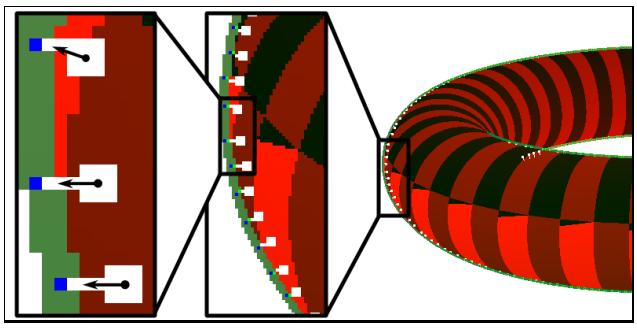


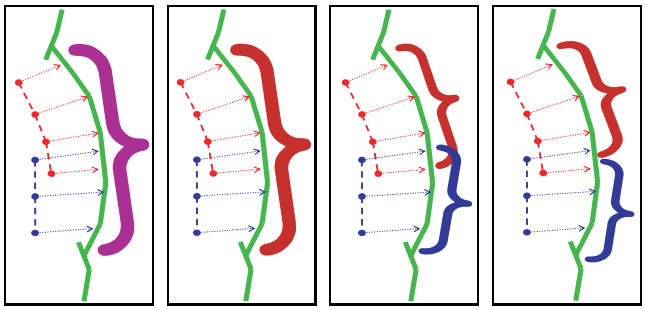
Figure 3: Sample propagation shown in successive enlargements of the ID image for a torus. Silhouette path IDs are green shades. Mesh faces are (striped) shades of orange. Search paths are marked in white. Blue pixels mark the successful termination of searches.

4.2 Brush Path Generation

We generate one or more brush paths along each silhouette path P by taking into account the votes it receives. Votes registered on P are first sorted into *vote groups* $\{(s_i, t_i)\}$ based on their brush path ID. The votes from a given brush path typically reach the same silhouette path in f_{i+1} en masse. Nevertheless, it frequently happens that some votes end up on silhouette paths that are also populated by votes from other brush paths. It can also happen that votes arrive out of order. To address such problems we process the vote groups as follows. We sort each group in order of increasing s_i. Wherever a large gap is found between s_i and s_{i+1} or t_i and t_{i+1} (relative to the mean) we split the group. Next we discard any groups that are (1) smaller than three votes, (2) ordered in reverse, (3) very sparse (i.e., the gaps in t_i are larger than anticipated), or (4) covering less than 5% of silhouette path length. When fewer than two vote groups survive on P, we assign a single brush path. For two or more surviving vote groups, we consider several policies for covering P with brush paths (Figure 4):

Mixed. We generate a single brush path covering P and parameterize it by mixing information from *all* the votes. This is the policy described by Bourdev [1998]. One problem with this policy is that it mixes data from different sources in a way that is not meaningful (effectively throwing away information). In addition, it suffers from the same drawback as the next policy.

Majority. This policy assigns a single brush path to P by choosing the vote group with the largest number of samples and ignoring the others. Unfortunately, this often yields poor temporal coherence. For example, in Figure 4b, all of P is parameterized consistently with the "red" vote group. The parameterization in the region of the "blue" votes is extrapolated from the red ones, resulting in a "pop" where the red parameterization replaces blue. (See the accompanying video.)



(a) mixed (b) majority (c) 1-*to*-1 (d) trimmed

Figure 4: Four policies for generating brush paths on a silhouette path given the same two vote groups. See Section 4.2.

1-*to*-1. This policy promotes temporal coherence by generating one brush path for each vote group. If any part of P remains uncovered, the nearest brush paths are extended to cover the gap. While coherent, this approach can lead to cumulative fragmentation over time, with a profusion of short brush paths covering P. Furthermore, the brush paths can overlap arbitrarily, leading to an accumulation of visual "clutter."

Trimmed 1-*to*-1. We favor this policy which eliminates the overlaps of 1-*to*-1. Each vote group is assigned a confidence based on its size. We assign brush paths as in 1-*to*-1 coverage, then trim away overlapping portions, leaving only those of highest confidence. While avoiding clutter, this approach still suffers from cumulative fragmentation over time. However, we can counteract this effect by merging abutting brush paths via a "healing" process as described at the end of the next section.

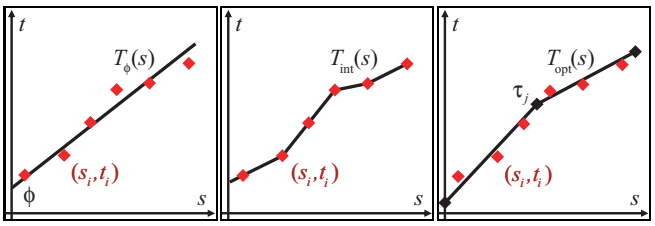
5 Brush Path Parameterization

Once the brush paths are generated it remains only to parameterize them. In the absence of any votes, we parameterize brush paths by a *scaled* arc-length (following Kalnins et al. [2002]). The scale factor ρ is a ratio of the model's initial size (recorded when the stylization is first applied) to its current size. We measure size as the screen-space diameter of the model's bounding sphere. The basic parameterization is thus $T(s) = \rho s$, where s is the arc-length parameter along the brush path. This scaling results in a more natural effect when the camera is zoomed in or out.

We now describe two schemes for parameterizing a brush path when votes are present. The schemes optimize for the competing objectives of uniform 2D arc-length parameterization *vs.* coherence on the 3D shape. Finally, we present a combined scheme that can strike a desired balance between the two extremes.

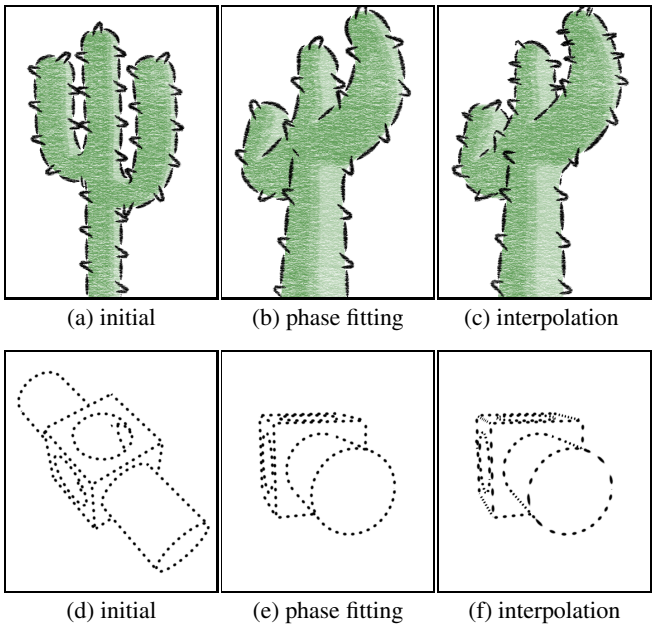
Phase fitting (Figure 5a): This method computes a uniform 2D arc-length parameterization $T_\phi(s) = \rho s + \phi$ by solving for the phase ϕ that best fits the votes (under least-squares). This works well for panning or zooming, but under changes in foreshortening it leads to a "swimming" effect wherein strokes appear to slide along the silhouette. This is undesirable for stroke styles meant to suggest 3D shape. Consider the cactus shown in Figure 6a. One expects the thorns to remain relatively fixed on the model, yet with phase fitting the spines flow around the top of the right arm.

Interpolation (Figure 5b): This method promotes coherence on the 3D shape by simply interpolating the votes: $T_{int}(s)$ is taken to be the polyline connecting the samples $\{(s_i, t_i)\}$). Under this policy, the spines of the cactus in Figure 6c appear fixed on the model, as expected. However, this effect is not desirable for styles that are more purely 2D (e.g. dotted lines). As shown in Figure 6f, this scheme leads to distortion in the dot spacing, whereas phase fitting maintains the expected even 2D spacing. Furthermore, interpolation can lead to progressive distortion over time (e.g. parameterization "piling up" in a region of brush path).



(a) phase fitting (b) interpolation (c) optimization

Figure 5: Three schemes for assigning parameterization $T(s)$ to a brush path given vote group $\{(s_i, t_i)\}$ (red points). See Section 5.



(a) initial (b) phase fitting (c) interpolation

(d) initial (e) phase fitting (f) interpolation

Figure 6: Behavior of strokes should depend on the semantics of the stylization. (See Section 5 and the accompanying video.)

Optimization (Figure 5c): This method balances the competing goals of coherence on the 3D shape and uniform 2D arc-length parameterization. We separate the second goal into two parts: globally matching arc-length scaled by ρ, and locally minimizing deviation from uniform arc-length. We then compute an optimal parameterization $T_{opt}(s)$ by minimizing a total energy defined as a weighted sum of energies corresponding to the different goals:

$$E = E_{votes} + \omega_\rho E_\rho + \omega_b E_{bend} + \omega_h E_{heal}. \qquad (1)$$

The first three terms of this energy correspond, respectively, to how well the parameterization (1) fits the votes, (2) matches scaled arc-length, and (3) avoids local distortion. The fourth term relates to the goal of "healing," described below. The non-negative weights ω_ρ, ω_b, and ω_h let a designer balance between these goals.

We find that taking t to be a piecewise-linear function of s is sufficient to fit the votes well without producing visual artifacts at tangent discontinuities. Thus, we take $T_{opt}(s)$ to be a polyline interpolating a vertex sequence $\{\sigma_j, \tau_j\}$, where the m knots σ_j are evenly-spaced arc-length parameters over the brush path.[1] Given the set of incoming votes and chosen weights, the optimization solves a system of equations to find τ_j's minimizing E. Because the system is linear, it can be solved efficiently (which is important since there can be many brush paths each frame).

The first term of the energy equation (1) measures the least-squares difference between T_{opt} and the votes:

$$E_{votes} = \frac{1}{n}\sum_{i=1}^{n}[T_{opt}(s_i) - t_i]^2 \qquad (2)$$

Each term $T_{opt}(s_i)$ depends only on the segment j of the polyline in which s_i falls. It is linear in the two free variables τ_j and τ_{j+1}.

[1] In our implementation, typical knot spacing is about 18 pixels in the ID image, compared to a spacing of 6 pixels for the parameter samples.

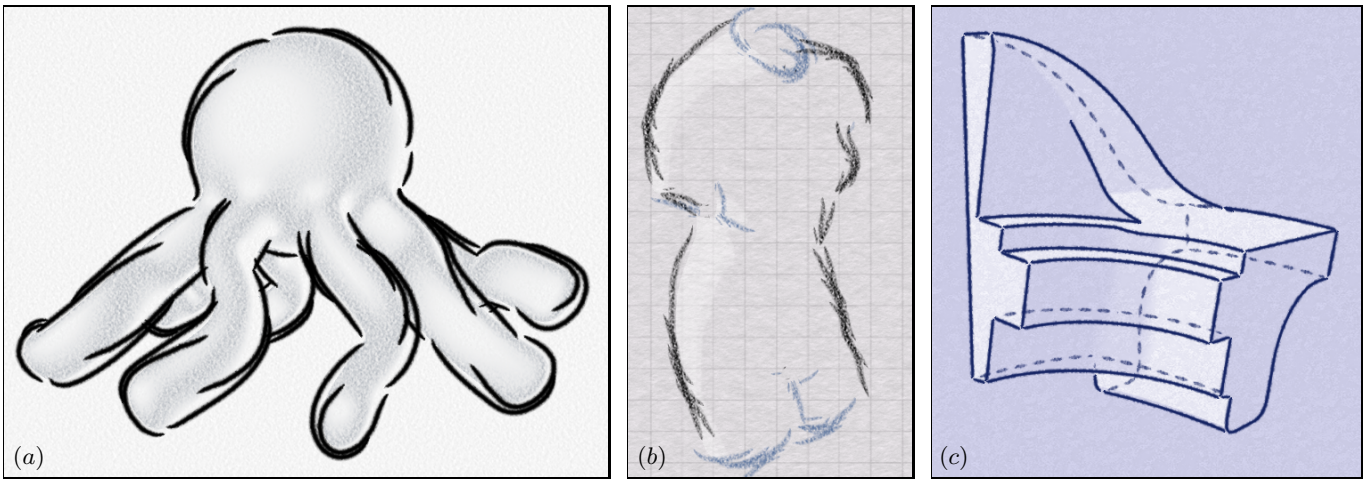


Figure 7: (a) Simple octopus model in a sketchy style. (b)"Venus" rendered with animated silhouettes. (c) Visualization of a mechanical part.

The next energy term is most easily expressed if we "normalize" the τ_j's by taking $\hat{\tau}_j \equiv \tau_j - \rho\sigma_j$. The energy measures how the τ_j's deviate from ρ-scaled arc-length, or simply how the *normalized* $\hat{\tau}_j$'s deviate from their average $\hat{\tau}_{ave}$:

$$E_\rho = \frac{1}{m}\sum_{j=1}^{m}[\hat{\tau}_{ave} - \hat{\tau}_j]^2 \tag{3}$$

The third term measures a "thin plate" energy that discourages the polyline from bending:

$$E_{bend} = \frac{1}{m}\sum_{j=1}^{m-2}[\tau_j - 2\tau_{j+1} + \tau_{j+2}]^2 \tag{4}$$

As mentioned at the end of Section 4.2, two abutting brush paths that share a silhouette path may be merged when their parameter discontinuity is small. We would like this *healing* to happen often to combat fragmentation. Whenever the parameter discontinuity is small (but not small enough to merge), we coerce the parameterizations on either side of the break to approach their average parameter value t_{ave} by creating an additional *healing vote* (s_k, t_{ave}) at each abutting endpoint s_k of the brush paths. Like the vote-fitting equation (2), these healing votes contribute to the overall energy via a sum of squares (with 0, 1 or 2 terms):

$$E_{heal} = \sum_k [T_{opt}(s_k) - t_{ave}]^2 \tag{5}$$

To minimize the total energy E, we solve the system of equations $\partial E/\partial \tau_j = 0$ using LU decomposition [Press et al. 1992]. The system is non-singular as long as there is at least one vote and ω_ρ is nonzero. All of the results demonstrated in the accompanying video were produced by parameterizing brush paths using T_{opt}, with the particular weights chosen according to the needs of the given stylization.

Figure	Faces	Brush Paths	Frames/Sec
video: jumping cactus	4k	10	30
7a: octopus	14k	19	24
7c: machine part	13k	50	20
8-left: not-still life	60k	70	20

Table 1: Frame rates and sizes of representative models.

6 Results

We have implemented the algorithms described in this paper and used them to render a variety of 3D models with stylized silhouettes, as demonstrated in Figures 1, 2, 6, 7 and 8. The accompanying video shows these and other examples in motion. We have observed the resulting stylized silhouettes to be temporally coherent across a wide range of styles and camera motions, for both static and animated geometry. The weights described in Section 5 offer practical control over the *kind* of coherence deemed appropriate by the scene designer. For the extreme examples in Figure 6 we selected weights to favor either 2D or 3D coherence, while a more even balance was appropriate for the intermediate examples of Figures 1cd, 7ab and 8-left. Furthermore, we observed that the behavior of the system varies smoothly and predictably as the designer adjusts these weights, and thus does not require careful tuning to achieve the desired effect.

Our system runs at interactive rates for scenes of moderate complexity on a 2.4Ghz Intel P4 CPU with nVidia Geforce Ti4200 GPU. For all the scenes shown in this paper and accompanying video, the frame rate ranges from 15 to 30 fps. Typical rates are reported in Table 1. For scenes under 100K polygons, the major factors affecting frame rates are the number and lengths of strokes rendered, due to visibility testing and computation of brush path parameterizations. The read-back of the ID image is a limiting factor that prevents the frame rate from exceeding 30 fps, even for simple scenes. This shortfall is not fundamental however as it stems from temporary limitations of GPU drivers and PC bus speeds.

With control over temporal coherence one can deliberately break coherence in controlled ways, as shown in the accompanying video. Figures 7b and 8-left show frames from animations wherein lines are drawn in deliberately noisy styles inspired by "Squigglevision" (the effect in the hand-animated TV show "Dr. Katz") and the "Loose and Sketchy" NPR animation of Curtis [1998]. For these effects, our system cycles the consecutive frames through a small set of differing stylizations to yield a jittery look. The temporally coherent underlying parameterization supports this effect, as well as a number of others based on animating the stroke stylizations.

While we have emphasized visible silhouettes in this discussion, these algorithms may be adapted for hidden silhouettes (Figures 1b and 7c), for creases and other features that remain fixed on the model (Figures 6d, 7c, and 8-left), and even for suggestive contours [DeCarlo et al. 2003]. Figure 8-right and the accompanying video demonstrate that these algorithms also work for animated meshes.

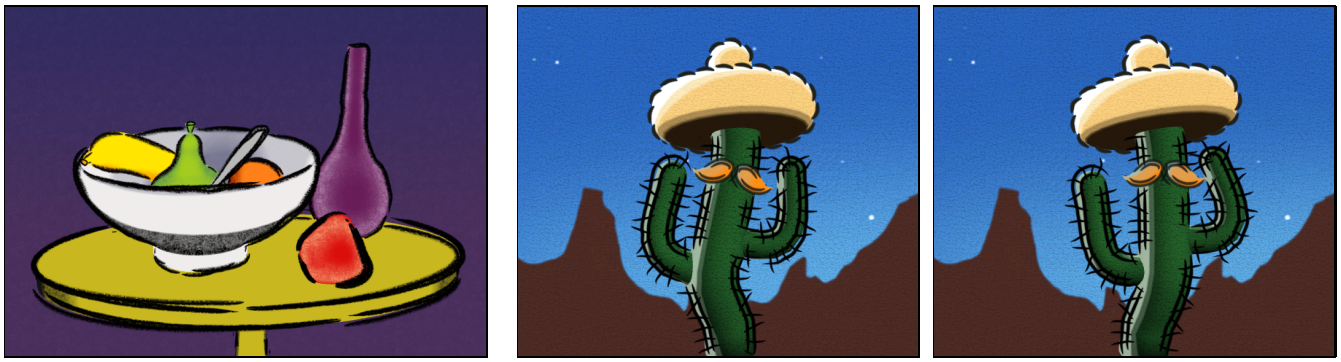


Figure 8: *Left:* "Not-so still life," with animated silhouettes. *Right:* "Boogie cactus," with stylized silhouettes on an animated figure.

7 Conclusion and Future Work

We have presented a way to render coherent stylized silhouettes of animated 3D models and demonstrated its effectiveness for a variety of scenes. With the ability to make silhouette stylizations coherent, we take a significant step toward increasing the expressive power of 3D computer graphics. This work suggests several areas for future exploration.

Localized stylization. Our approach allows stylization to be applied uniformly across all silhouettes of some *patch* of the mesh. However, the stylizations of each patch do not interact in any way. An open question is how can silhouettes transition smoothly between styles as they cross patch boundaries? One challenge is the UI: how would the designer specify what areas should have what style? Another challenge is how to better support styles that suggest 3D details on the surface, such as fur or leaves. Our current algorithm does not work for stylizations requiring a special orientation in 3D, such as tufts of hair that hang down. If we tried that in our current system, we could make the hair point down on one side of the model, but on the other side it would stick up.

Simplification. When an object at a distance appears small, human-crafted illustrations typically omit lines to reduce the image complexity. An interesting question is how to omit (or merge) strokes in such scenes automatically, and to do so with temporal coherence as the camera zooms in or out. The same mechanism could be useful for meshes that contain many small surface details, leading to silhouettes that are fragmented into many tiny loops. The resulting silhouette paths could benefit from simplification into longer, smooth connected paths.

Acknowledgments

We thank Grady Klein for the animated cactuses, Michael Kowalski for production help, and Doug DeCarlo and Szymon Rusinkiewicz for helpful discussions. This research was supported by Intel Labs.

References

BOURDEV, L. 1998. *Rendering Nonphotorealistic Strokes with Temporal and Arc-Length Coherence*. Master's thesis, Brown University. http://www.cs.brown.edu/research/graphics/art/bourdev-thesis.pdf.

BREMER, D. J., AND HUGHES, J. F. 1998. Rapid Approximate Silhouette Rendering of Implicit Surfaces. In *Proc. of Implicit Surfaces*, 155–164.

CURTIS, C. J. 1998. Loose and Sketchy Animation. *Technical sketch, SIGGRAPH 98*. Also see: http://otherthings.com/uw/loose/.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive Contours for Conveying Shape. In *Proceedings of ACM SIGGRAPH 2003*.

DURAND, F. 2002. An Invitation to Discuss Computer Depiction. *Proceedings of NPAR 2002*, 111–124.

GOOCH, B., AND GOOCH, A. 2001. *Non-Photorealistic Rendering*. A. K. Peters.

GOOCH, B., SLOAN, P.-P. J., GOOCH, A., SHIRLEY, P. S., AND RIESENFELD, R. 1999. Interactive Technical Illustration. In *1999 ACM Symposium on Interactive 3D Graphics*, 31–38.

HERTZMANN, A., AND ZORIN, D. 2000. Illustrating Smooth Surfaces. In *Proceedings of ACM SIGGRAPH 2000*, 517–526.

ISENBERG, T., HALPER, N., AND STROTHOTTE, T. 2002. Stylizing Silhouettes at Interactive Rates: From Silhouette Edges to Silhouette Strokes. *Computer Graphics Forum (Proc. of Eurographics) 21*, 3.

ISENBERG, T., FREUDENBERG, B., HALPER, N., SCHLECHTWEG, S., AND STROTHOTTE, T. 2003. A Developer's Guide to Silhouette Algorithms for Polygonal Models. *IEEE Computer Graphics and Applications 23*, 4 (July/August). To appear.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. WYSIWYG NPR: Drawing Strokes Directly on 3D Models. *ACM Transactions on Graphics 21*, 3, 755–762.

LAKE, A., MARSHALL, C., HARRIS, M., AND BLACKSTEIN, M. 2000. Stylized Rendering Techniques for Scalable Real-Time 3D Animation. In *Proceedings of NPAR 2000*, 13–20.

MARKOSIAN, L., KOWALSKI, M. A., TRYCHIN, S. J., BOURDEV, L. D., GOLDSTEIN, D., AND HUGHES, J. F. 1997. Real-time Nonphotorealistic Rendering. *Proceedings of SIGGRAPH 97*, 415–420.

MASUCH, M., SCHLECHTWEG, S., AND SCHÖNWÄLDER, B. 1997. daLi! – Drawing Animated Lines! In *Proceedings of Simulation und Animation '97*, SCS Europe, 87–96.

MASUCH, M., SCHUMANN, L., AND SCHLECHTWEG, S. 1998. Animating Frame-to-Frame Coherent Line Drawings for Illustrative Purposes. In *Proceedings of Simulation und Visualisierung*, SCS Europe, 101–112.

NORTHRUP, J. D., AND MARKOSIAN, L. 2000. Artistic Silhouettes: A Hybrid Approach. *Proceedings of NPAR 2000*, 31–38.

PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., AND VETTERLING, W. T. 1992. *Numerical Recipes: The Art of Scientific Computing*, 2nd ed. Cambridge University Press.

RASKAR, R. 2001. Hardware Support for Non-photorealistic Rendering. In *Proc. of SIGGRAPH/Eurographics Workshop on Graphics Hardware*.

ROSSIGNAC, J., AND VAN EMMERIK, M. 1992. Hidden Contours on a Framebuffer. *Proc. of 7th Workshop on Computer Graphics Hardware*.

STROTHOTTE, T., AND SCHLECHTWEG, S. 2002. *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufman.

ZELEZNIK, R., HERNDON, K., AND HUGHES, J. F. 1996. SKETCH: An Interface for Sketching 3D Scenes. *Proc. of SIGGRAPH 96*, 163–170.