Active Strokes: Coherent Line Stylization for Animated 3D Models

Pierre Bénard[1,2] Jingwan Lu[3] Forrester Cole[4] Adam Finkelstein[3] Joëlle Thollot[1,2]
[1]Grenoble University, LJK [2]INRIA [3]Princeton University [4]MIT CSAIL

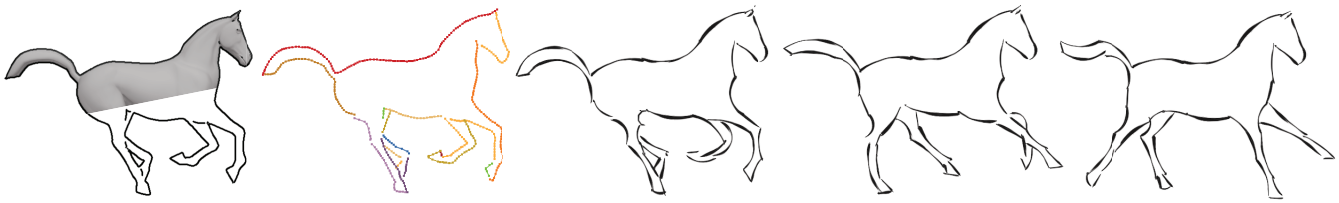


Figure 1: *Stylized animation of a galloping horse. From left to right: line samples are extracted from a 3D model; active strokes track the samples; brush paths are attched to the strokes and stylized as circular arcs; two more frames of animation exhibit temporal coherence.*

Abstract

This paper presents a method for creating coherently animated line drawings that include strong abstraction and stylization effects. These effects are achieved with *active strokes*: 2D contours that approximate and track the lines of an animated 3D scene. Active strokes perform two functions: they connect and smooth unorganized line samples, and they carry coherent parameterization to support stylized rendering. Line samples are approximated and tracked using active contours ("snakes") that automatically update their arrangment and topology to match the animation. Parameterization is maintained by brush paths that follow the snakes but are independent, permitting substantial shape abstraction without compromising fidelity in tracking. This approach renders complex models in a wide range of styles at interactive rates, making it suitable for applications like games and interactive illustrations.

Keywords: NPR, line drawings, temporal coherence, snakes

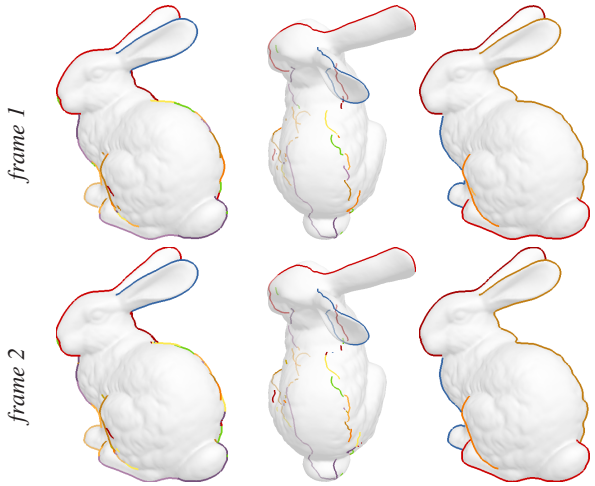
1 Introduction

Line drawings created by artists often include stylistic effects such as textured brushes and graceful, curved strokes. These effects reflect the motion of an artist's tool, and as such are fundamentally 2D in appearance. The main challenge in recreating such effects for computer animation is to create lines that smoothly track the 3D scene while maintaining the desired 2D texture and shape. This paper presents a method to create such a set of temporally coherent, stylized lines from an animated 3D model. We term these lines *active strokes*, because they adapt and extend active contours ("snakes") [Kass et al. 1988] to stylized rendering. Active strokes automatically update their topology and shape as the underlying 3D scene animates, and provide coherent parameterization for texture and shape abstraction (Figure 1). Our approach performs at interactive framerates, making it suitable for artistic exploration, games, interactive visualizations and illustrations.

Stylization effects such as texture and shape abstraction require that lines be represented as connected curves. Suitable curves can sometimes be extracted and stylized by traversing the 3D scene in object space [Northrup and Markosian 2000; Grabli et al. 2010]. In order to produce smooth, coherent animation, several approaches have been proposed to preserve the parameterization of object-space lines over time [Kalnins et al. 2003; Bénard et al. 2010; Buchholz et al. 2011; Karsch and Hart 2011]. However, for models of moderate complexity like the Stanford bunny shown in Figure 2, object-space line extraction may produce many noisy, short segments that confound stylization with spatial and temporal coherence. Moreover, object space lines lack natural level of detail (LoD) control, and offer no straightforward mechanism to depart from the underlying surface in support of 2D shape abstraction effects.

Active strokes avoid these issues by operating in image space, using line samples output by filter-based line extractors (e.g. [Saito and Takahashi 1990; Raskar and Cohen 1999; Lee et al. 2007]). Since no natural connectivity exists between these samples, active strokes first vectorize the line samples using snakes. Like traditional snakes, our snakes conform to the changing shape of the features in the scene. Unlike traditional snakes, they also need to adapt their topology and overall arrangement during animation. As the snakes split and merge to track the line samples, their parameterization may change discontinuously. Active strokes therefore carry independently parameterized brush paths on top of the snakes to maintain consistent stretches of parameterization from one frame to the next (Figure 3). Since the snake handles tracking of motion in the scene, the brush paths may deviate substantially from the position of the underlying line to support shape abstraction.

The key innovation of our method is the introduction of an independent 2D entity, the active stroke, that approximates lines extracted from the 3D model and persists from frame to frame. We present new mechanisms to add, remove, trim, extend, split and merge these strokes to match animated feature lines. Our method provides level of detail control, operates on line samples from any type of line extractor, and supports new stylization effects such as shape abstraction for lines. We demonstrate our method with a line rendering system that achieves interactive frame rates and provides a broad range of styles.



(a) *silhouettes in 3D* **(b)** *side view of (a)* **(c)** *snakes*

Figure 2: *Silhouette coherence. Frame 1: (a) Silhouettes in object space appear to be continuous in the image, but (b) are extracted as many disconnected parts. (c) Snakes recover 2D connectivity. Frame 2: (a) The silhouette is visually coherent with frame 1, but is (b) composed of a different set of disconnected parts. Nevertheless, the snakes (c) provide coherence.*

2 Related Work

Line stylization. Image processing is the simplest, most efficient way to extract lines from 3D models. The seminal work of Saito and Takahashi [1990] filters depth and normal maps to extract contours and creases. It produces line drawings with natural coherence and LOD in 2D, but does not support much stylization or abstraction due to the raster nature of the output. Generalizing the method of Lee et al. [2007], Vergne et al. [2011] combine 2D local fitting with spatially-varying convolution to produce more complex styles. But their implicit definition is without line parameterization, preventing the application of stroke texture or abstraction. Our approach supports image-based line extraction, but recovers connectivity and parameterization in support of such effects.

Object space methods extract connected paths on the surface of the model. These paths have obvious parameterization for each individual frame [Grabli et al. 2010], but the parameterization can change abruptly during animation due to changes in line topology. Kalnins et al. [2003] proposed the first complete method to enforce coherent parameterization by using the connectivity of smooth silhouettes [Hertzmann and Zorin 2000] to propagate stylization. This approach works well for simple models with few lines, where the mapping between object and image space is almost one-to-one. Models of moderate complexity like the Stanford bunny, however, produce many, tiny line fragments that collide in image space, thwarting effective propagation (Figure 2). The method of Bénard et al. [2010] enforces a common parameterization for nearby lines (thereby making short segments behave as parts of a longer line) but this solution is generally suitable only for nearby lines that are also parallel. The method of Karsch and Hart [2011] tracks snake-like *snaxels* on the 3D object rather than re-extracting at each frame, avoiding propagation of parameterization. Because the snaxels live in object-space, however, they cannot easily provide LoD control or abstraction, and do not appear to address the aforementioned line fragmentation problem.

Another contribution of Kalnins et al. is the use of multiple brush paths per silhouette. The parameterization of each brush path is independently optimized to find a compromise between uniformity in screen-space and in object space. Our brush paths behave similarly, with a few differences. Their brush paths both track 3D motion and transmit parameterization from frame to frame. Our snakes handle the former role, so that our brush paths can be abstracted, deviating far from the snakes without compromising the quality of tracking (Figure 3). In addition, we employ the *self-similar line artmap* (SLAM) textures of Bénard et al. [2010] to allow extra flexibility in the brush path parameterization.

To date most line stylization approaches operate in an online fashion, i.e., without knowledge of the future positions of the lines. Buchholz, et al. [2011] present a method for parameterizing lines across an entire animation, allowing lines to anticipate topology changes and merge smoothly. While presented in the context of object-space occluding contours, thus sharing the difficulties of other object-space approaches, the spatio-temporal concept is independent and could be applied to our active strokes, though we have not explored that option here.

Active contours in NPR. Introduced by Kass et al. [1988], snakes are widely used in computer vision and medical imaging applications because of their robustness to noise. They have also been applied in several NPR contexts: line extraction [Karsch and Hart 2011], painterly rendering [Hertzmann 2001], and video stylization [Agarwala 2002; Agarwala et al. 2004]. As mentioned, the snaxels approach of Karsch and Hart works in object space, where the topology of the contours generators forms a set of closed loops. On the other hand, methods that use snakes to delineate image space region boundaries need to deal with a network of open curves, and thereby take on the need to deal with more complex topology changes over time. The systems of Agarwala et al. segment and track color regions in videos with B-spline snakes for coherent stylization. The user manually draws the initial outlines of the contours and the outlines are propagated to track features in the video. These systems were designed for processing video offline and can therefore rely on heavy-weight optimization and manual correction. In contrast, our approach is designed to track motion field in animated 3D scenes where the camera path is not known in advance, for example games. Therefore it must be interactive, fully automatic, and cannot rely on "future" frames. On the other hand, we benefit from the quality of our input motion – the motion field extracted from 3D is generally less noisy than optical flow from video.

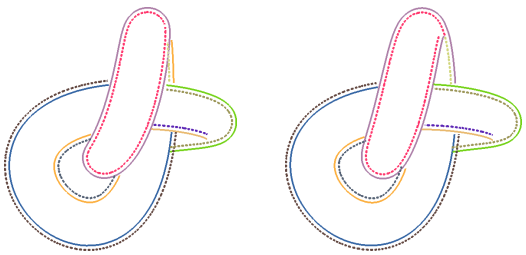


Figure 3: *Each brush path (dotted) is defined relative to a snake (solid), but can deviate substantially to support abstraction effects as shown in Figure 1. As the knot turns, the snake topology evolves; for example the orange snake merges with the purple snake (top). Coherent arc-length parameterization is preserved by maintaining the beige brush path over the purple snake.*

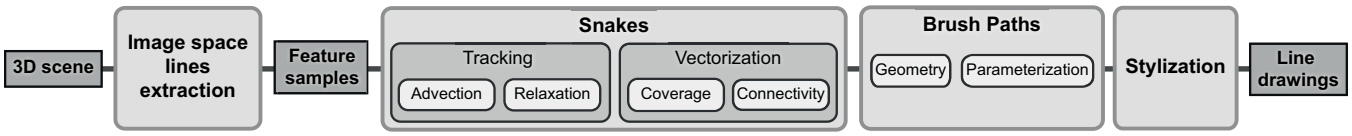


Figure 4: *Stages of our line drawing pipeline, an overview of which is given in Section 3 below.*

Line drawing LOD. Line level of detail (LoD) control is necessary to produce clean, abstract drawings from complex models. Image-space approaches provide a natural limit on line complexity (image pixels), but it is often desirable to further reduce density. Object-space lines allow arbitrary line density, but provide opportunities for simplification both during stylization [Grabli et al. 2004; Cole et al. 2006] and as a post-process [Barla et al. 2005; Shesh and Chen 2008]. A common approach is to extract lines, measure their density in screen space, then either omit or merge lines in overly dense areas. Grabli et al. [2004] and Barla et al. [2005] found that good results could be achieved by removing only overlapping lines that also align in orientation. We adopt a similar, simple definition of the line coverage that allows efficient processing and explicit control over spatial and directional proximity. Finally, by generating brush paths that are independent of the feature lines, we can easily merge lines rather than simply omitting them, unlike approaches such as Cole et al. [2006].

3 Overview

The key idea of our approach is create an independent set of curves (*active strokes*) that track and approximate unorganized feature line samples and provide a coherent parameterization for stylization. Active strokes use snakes in order to track and vectorize the input feature samples. On top of the snakes an active stroke carries a set of brush paths that propagate coherent parameterization and give the final shape of the strokes. Figure 4 illustrates our pipeline.

Our method takes as input a set of feature samples with 2D position and 2D tangent, and for animated scenes, 2D velocity. These samples can be extracted by any method that provides an orientation at each pixel (e.g. [Lee et al. 2007; Vergne et al. 2011]), but the smoother and more coherent the input samples, the better the result. We have found that lines extracted with steerable quadrature pair filters [Freeman and Adelson 1991] provide especially good input. We describe our application of these filters in Appendix A.

For the first frame only, we construct a set of snakes conforming to these feature samples via a stochastic process described in Section 5.

For successive frames, the snakes track and vectorize the input samples. To obtain a good tracking, we maintain two properties for our snakes: *coherence*, meaning snakes should evolve continuously from frame to frame as they track the features of the model; and *accuracy*, meaning snakes should faithfully represent the shape of the underlying linear features extracted from the scene. For vectorization, we maintain the following properties: *coverage*, each feature line should be covered by one snake and each snake should cover a feature; *simplicity*, snakes should maintain simple, smooth and clean topology; *length*, snakes should be as long as possible while respecting the previous criteria.

To achieve *coherence*, snakes from the previous frame are advected to follow the motion field in the scene (Section 4.1). Next the snakes are relaxed to conform to the features in the current frame (Section 4.2) in support of *accuracy*. After advection and relaxation, the goals of *coverage, simplicity* and *length* will generally have been violated. The ideal solution would involve a complex non-linear optimization, difficult to achieve in a system designed for interactive frame rates. Instead, we adopt a greedy approach and update each snake individually, using heuristics to move towards a better solution (Section 5). We obtain efficient vectorization by beginning with a good guess adapted from the previous frame.

Finally, we build on top of the snakes a set of brush paths that provide a coherent parameterization across frames (Section 6.2). By relying on the tracking behavior of the snakes, the brush paths can deviate widely from the underlying feature lines to achieve stylization and abstraction goals (Section 6.1). The brush paths are then textured and drawn to produce the final illustration.

4 Tracking Feature Lines

The first step of our approach is to track the feature lines available in each frame of the animation using snakes: 2D polylines that act like elastic strands, simultaneously attempting to satisfy external attraction forces and internal smoothing forces. In our approach the snakes are attracted to feature lines and track them from frame to frame. At each frame, we move the snakes according to the motion field in the scene (Section 4.1) and then relax them to conform to the feature lines in the current frame (Section 4.2).

4.1 Advection

At initialization, each snake vertex is associated with a 2D feature sample (Section 5). During advection, the snake vertex is moved according to the projected 3D motion field of the scene. The 3D motion field can be computed either by back projecting the 2D samples in the camera at frame $i-1$, reprojecting these 3D positions into the camera at frame i and computing the difference or by using an optical flow computation when the geometry is not available.

After advection, lines from frame $i-1$ are nearby (but not perfectly aligned with) features in frame i. At this stage, the method of Kalnins et al. [2003] performs an explicit search to associate line samples with features in frame i. Instead, we use relaxation to attach the advected snakes to features in frame i, as described next.

4.2 Relaxation

As described by Kass et al. [1988], snakes are parametric 2D curves $\mathbf{v}(s) = (x(s), y(s)),\ s \in [0,1]$ that minimize the energy E, defined as the sum of smoothing and data terms E_{int} and E_{ext}:

$$E = \int_0^1 E_{int}(\mathbf{v}(s)) + E_{ext}(\mathbf{v}(s))\, ds$$

The internal energy E_{int} ensures continuity (first-order membrane term weighted by α) and *smoothness* (expressed as second-order thin-plate term weighted by β):

$$E_{int}(v(s)) = \frac{1}{2}(\alpha(s)|\mathbf{v}'(s)|^2 + \beta(s)|\mathbf{v}''(s)|^2)$$

with $\mathbf{v}'$ and $\mathbf{v}''$ the first and second derivatives. In practice, we use constant α and β parameters (both equal to 0.001 in our experiments).

Addressing the *accuracy* goal, the external energy E_{ext} attracts the snake to areas of interest – in our case, feature lines that are projected and rasterized into a binary image I. This image is filtered by a Gaussian kernel whose gradient is the attraction field:

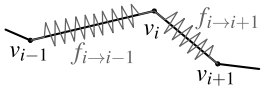
$$E_{ext}(v(s)) = -|\ \nabla\ [G_\sigma(\mathbf{v}(s)) \star I(\mathbf{v}(s))]\ |^2$$

The size σ of the filter kernel controls the width of the attraction field around the contour. A larger filter makes relaxation more robust to errors in advection, at the price of accuracy in tracking. In our experiments we use $\sigma = 8$px for screen-resolution images.

Following the approach of Kass et al. we optimize the energy E by gradient descent. The continuous snake $\mathbf{v}(s)$ is discretized as n vertices v_i and the energy is iteratively minimized by a semi-implicit Euler scheme. At each frame, a matrix of internal forces is constructed and applied iteratively, updating the contours positions and their associated external forces at each iteration. This scheme is simple and fast so long as the advection step provides a good starting condition. But when advection fails to produce a good initial guess it is likely due to extreme motion of the model. In this case, temporal coherence may not be necessary or even expected.

To maintain good sampling, especially while zooming, the snake resolution is updated after several iterations of relaxation. Via the approach of Delingette et al. [2000] we ensure an almost uniform sampling of the curves, with a maximum distance of $\delta_{\mathsf{max}} = 6$px.

Snakes have a natural tendency to shrink like a rubber-band due to their internal forces. This shrinking tends to produce inaccurate results where snakes round off corners or fail to reach line endpoints. To overcome this behavior, we apply spring forces along each snake edge so that they maintain almost constant length if no tangential external force is applied. The spring forces are implemented as an extra term in the external energy, similar to the approach of McInerney et al. [1995]:



$$f_{i\to i+1} = k\ (L_i - \|v_{i+1} - v_i\|)\frac{v_{i+1} - v_i}{\|v_{i+1} - v_i\|}$$

and the symmetric case for $f_{i\to i-1}$ where k is the stiffness parameter of the spring ($k = 1$ by default) and L_i is the rest-length equal to $\|v_{i+1} - v_i\|$ at the previous frame.

5 Vectorizing Feature Lines

During animation snakes must be created, deleted, or modified to preserve a good approximation of the underlying feature lines. Constructing snakes from the feature samples is equivalent to vectorizing line art. This is a difficult problem even for still images, and our scenario also requires that the vectorization be temporally coherent. Fortunately, we can exploit the smooth tracking behavior of the snakes to ease the vectorization problem. Snakes from the previous frame that have been advected and relaxed provide an excellent starting point for vectorizing the samples for the current frame, because the features they track are coherent in image space.

However, starting with these snakes, we still need to address two challenges. The first challenge is that after advection and relaxation we generally have not met our unit coverage goal. Several snakes may now overlap in image space where a single snake would suffice. Also, new lines may have appeared in regions where no snake existed before, or conversely a snake may remain where feature lines no longer exist. The second challenge is snake topology. Unlike our scenario, typical formulations for parametric snakes need not address the issue of automatically inferring their connectivity, usually by relying on user input. Starting from this initial connectivity, some approaches have been proposed to modify topology of the contours during segmentation tasks [McInerney and Terzopoulos 2000; Delingette and Montagnat 2000; Ji and Yan 2002]. We take inspiration from these methods to define a set of heuristic rules that allow the snakes to change their topology and overall arrangement during animation so as to match the underlying feature lines.

We organize these heuristics as six operators – delete, split, trim, extend, merge, insert – which are applied sequentially in a greedy fashion on the vertices of the snakes, until they can no longer modify the snakes. The order in which the operators are applied is important. Deletion and splitting occur first, so as to allow subsequent trimming to produce clean junctions. Extension also benefits from following splitting, and must precede merging to allow snakes to join across a gap. Insertion of new snakes occurs to address features without coverage after all the other operations have had the opportunity to adjust the snakes (especially extension). So in summary, we delete as much as possible, trim as much as possible, and so on; and finally insert new snakes where extra coverage is needed.

On the other hand, the order in which the snakes are processed for each operation has relatively less influence on the quality of the final result. Nevertheless, we process snakes in order from longest to shortest, except for the trim operator for which we use the opposite order, with the goal of maximizing the length of the longer snakes.

5.1 Operations

Our goal is to cover each feature line with exactly one snake. Recall that each feature line is actually composed of an unorganized set of feature samples $\mathcal{S}$ that are typically sparse in the image. We define *coverage* to mean that a feature sample s_i with tangent $\vec{T_i}$ has a snake vertex v_j nearby whose local orientation $\vec{t_j}$ is similar:

$$\mathsf{covered}(s_i) \equiv \exists v_j : \begin{cases} \|s_i - v_j\| & < & r_{\mathsf{cover}} \\ |\vec{T_i} \cdot \vec{t_j}| & > & \theta_{\mathsf{c}} \end{cases}$$

The constants r_{cover} and θ_{c} control, respectively, the apparent line density and the tendency for snakes to match the orientation of the underlying features. A related notion is $\mathsf{footprint}(v_j)$ which describes the set of feature points covered by v_j:

$$\mathsf{footprint}(v_j) = \left\{ s_i \in \mathcal{S} \mid (\|s_i - v_j\| < r_{\mathsf{cover}}) \wedge (|\vec{T_i} \cdot \vec{t_j}| > \theta_{\mathsf{c}}) \right\}$$

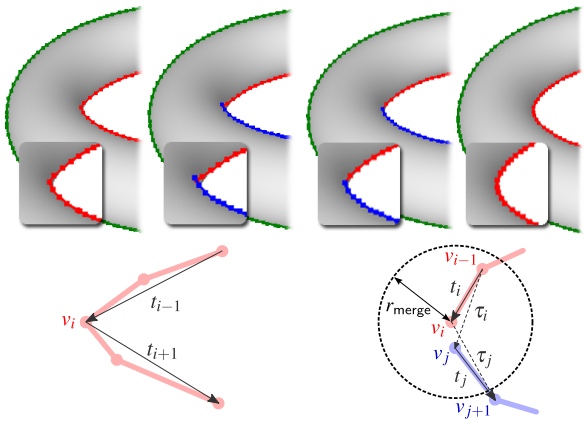
Initialization. For the first frame of an animation we start by creating new snakes as follows. We randomly pick an uncovered feature sample from which we grow a snake as far as possible using the extension operator described below. This process is repeated until it can no longer improve coverage. Though not providing optimality guarantees, this approach is simple and fast, and it offers a reasonable first guess for subsequent processing.

Delete. Snake vertices that are not covering any feature sample are deleted. This mechanism may remove an entire snake where a feature has become occluded, or may cut away portions of a snake in case of partial occlusion.

Split. (Figure 5a) To accurately track sharp features of a 3D model, a snake is split at vertex v_i if a sharp angle appears there, found by:

$$c_{\mathsf{split}} \equiv \overrightarrow{t_{i-1}} \cdot \overrightarrow{t_{i+1}} < \theta_\perp$$

where $\overrightarrow{t_{i-1}}$ and $\overrightarrow{t_{i+1}}$ are the tangents at two neighboring vertices and $\theta_\perp$ is a threshold constant for how much the snake is allowed to bend at a vertex.



(a) *Split* **(b)** *Merge*

Figure 5: *Split and merge operations modify snake connectivity. (a) When a snake bends too sharply around a corner it is split. (b) When two snakes abut smoothly they are joined.*

Trim. (Figure 6a) Having deleted portions of snakes that do not cover features, and split at sharp corners, we next trim snakes in regions where parallel lines provide more coverage than needed. Several strategies might be possible, for example attempting to merge lines where they nearly overlap. However, we have found a simple policy that trims lines at their endpoints to be effective for achieving unit coverage. Specifically, the three criteria for removing the endpoint v_i from a snake are that it is too close to a vertex v_j of a different snake, they are close to parallel, and that by removing v_i we will not leave a feature uncovered that would otherwise be covered by v_i:

$$c_{\text{trim}} \equiv \begin{cases} \|v_i - v_j\| & < & r_{\text{trim}} \\ |\vec{t_i} \cdot \vec{t_j}| & > & \theta_{/\!/} \\ \text{footprint}(v_i) & \subseteq & \cup_{k \neq i} \text{footprint}(v_k) \end{cases}$$

where $\vec{t_i}$ and $\vec{t_j}$ are their respective tangents, and r_{trim} and $\theta_{/\!/}$ are threshold constants for how close and parallel these parts of the snakes may become before trimming occurs.

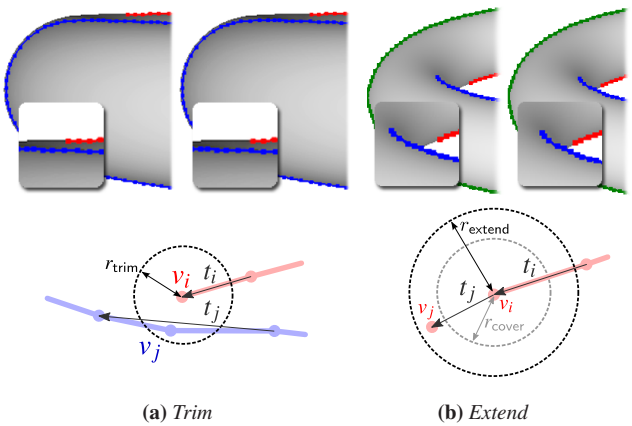
Extend. (Figure 6b) Next we extend snakes at every endpoint where doing so will offer coverage for otherwise uncovered feature points whose tangents align locally with the snake. Specifically, beyond the current endpoint v_i we will consider appending a new endpoint v_j at the location of every uncovered feature sample, based on four criteria:

$$c_{\text{extend}} \equiv \begin{cases} \neg \text{ covered}(v_j) & & \\ \|v_j - v_i\| & < & r_{\text{extend}} \\ \vec{t_i} \cdot \vec{t_j} & > & \theta_{/\!/} \\ |\vec{t_j} \cdot \vec{T_j}| & > & \theta_{/\!/} \end{cases}$$

where $\vec{t_i}$ and $\vec{t_j}$ are the tangents at the snake vertices, $\vec{T_j}$ is the tangent of the feature sample, and r_{extend} is a search radius.

In general there may be many candidate feature locations for which the extension criteria above are satisfied. Therefore, we compute a goodness score g_j that describes how well each candidate extends the snake and then choose the candidate with the highest score. The score favors aligning the tangents and a relatively short extension, as follows:

$$g_{\text{extend}} = \alpha \left(\frac{\min(\vec{t_i} \cdot \vec{t_j}, |\vec{t_j} \cdot \vec{T_j}|) - \theta_{/\!/}}{1 - \theta_{/\!/}} \right) + (1-\alpha) \left(\frac{r_{\text{extend}} - \|v_j - v_i\|}{r_{\text{extend}} - r_{\text{cover}}} \right)$$



(a) *Trim* **(b)** *Extend*

Figure 6: *Modifying snakes to improve coverage. (a) The endpoint of a snake is trimmed if it is too close and parallel to another snake. When multiple snakes overlap in screen space this operation moves towards unit coverage. (b) The end of a snake is extended when doing so adds coverage to otherwise uncovered features. In concert, the trim and extend operations tend to clean up T-junctions.*

where the two terms in parentheses have been normalized to the range $[0, 1]$ and α is a parameter that balances between them.

Merge. (Figure 5b) To simplify the snake topology, two snakes are merged if they are locally parallel and their endpoints are close:

$$c_{\text{merge}} \equiv \begin{cases} \vec{\tau_i} \cdot \vec{t_j} & > & \theta_{/\!/} \\ \vec{\tau_j} \cdot \vec{t_i} & > & \theta_{/\!/} \\ \|v_j - v_i\| & < & r_{\text{merge}} \end{cases}$$

These criteria may lead to multiple conflicting candidate mergers, for instance where three endpoints meet in a Y shape. Therefore, in a fashion similar to the extension process described above, we assign a goodness score that favors straightness and closeness:

$$g_{\text{merge}} = \alpha \left(\frac{\min(\vec{\tau_i} \cdot \vec{t_j}, \vec{\tau_j} \cdot \vec{t_i}) - \theta_{/\!/}}{1 - \theta_{/\!/}} \right) + (1-\alpha) \left(\frac{r_{\text{merge}} - \|v_j - v_i\|}{r_{\text{merge}} - r_{\text{cover}}} \right)$$

where v_i and v_j are the endpoints of two distinct contours, $\vec{t_i}$ and $\vec{t_j}$ their tangents, and $\vec{\tau_i}$ and $\vec{\tau_j}$ are tangents of hypothetical merger segments, shown in Figure 5b. We perform as many non-conflicting mergers as possible, in order of descending score.

Insert. This final operation works like the initialization step described above, except that it occurs after all the other operations. Its role is to insert new snakes where new features have appeared that have not been addressed by, for example, extending existing snakes.

5.2 Practical considerations

Threshold Constants. The scope of action of the four operations that trim, extend, split and merge the snakes are given by threshold constants for various radii and angles. Several relationships between them are important. In order for the snakes to establish proper coverage between samples: $r_{\text{cover}} > \delta_{\text{max}}/2$. In order to permit extension to occur: $r_{\text{extend}} > r_{\text{cover}}$. In order to enforce hysteresis between split and merge events: $\theta_{\perp} < \theta_{/\!/}$. Other constants trade off between accuracy and performance. In our experiments we have found the values effective for the full range of results shown: r_{cover} = 10px; r_{trim} = 5px; r_{extend} = 20px; r_{merge} = 20px; $\theta_{\text{c}} = \theta_{/\!/} = 0.85$; $\theta_{\perp} = 0.5$; $\alpha = 0.8$.

Acceleration structure. During advection and coverage updates, we often need to access the neighbors of snakes vertices or feature samples in the coverage radius. To expedite local search, we use the data structure of Delingette et al. [2000]: a randomized sparse regular grid (implemented by a hash table) containing all the snakes vertices and feature samples. With a grid cell size of 2 r_{cover}, we enumerate all points in a disc of radius r_{cover} by walking four cells.

6 Brush paths shape and parameterization

Snakes approximate and track the underlying feature lines from frame to frame. Based on this temporally coherent layer, we build a set of brush paths that are in charge of (1) providing the shape and position of the final strokes, and (2) of computing a temporally coherent parameterization.

6.1 Brush path geometry

We represent a brush path as a 2D polyline with vertices associated $1-1$ to a sequence of vertices on a single snake (Figure 3). The position of brush path vertices are described relative to their associated snake vertices to take advantage of the snake coherence during motion. The user controls a target length l_{max} (potentially infinite) for the brush path allowing for short brush paths if desired. In addition, brush paths can have "overshoot" that extends their end segments.

Smoothing in time During rapid camera movements, feature lines may appear or disappear abruptly. As snakes persist across frames, we can smooth out these events by assigning a lifespan to the snake vertices, as follows. Each snake vertex has a lifespan κ, initially null. At each frame, if a vertex persists, its lifespan increases: $\kappa = \max(\kappa + 1, \kappa_{\max})$ with $\kappa_{\max}$ a constant defined by the user (typically 3 in our experiments). Conversely, if a vertex disappears, its lifespan decreases ($\kappa = \kappa - 1$). This lifespan is transmitted to the brush path vertices and can be mapped to different style attributes, such as line opacity or thickness. That way, brush strokes fade in and out progressively during $\kappa_{\max}$ frames.

Shape Abstraction The shape of the brush path can deviate substantially from the shape of its corresponding snake, which allows highly abstract styles. As a proof of concept we show two extreme examples that have never been animated before: straight brush paths and circular arcs (Figure 7).

Drawing with straight brush paths implies to partition each snake in a set of approximately linear segments. We compute this partition using dynamic programming, similarly to the approach of McCrae and Singh [2009] for fitting sketched clothoid curves.

Between two frames, the initially straight brush paths are propagated following the motion of the underlying snakes. However the shape of the snakes is likely to have changed. Consequently, the brush paths may not be straight anymore, and we may even need to break a straight path into two or more segments to better match the new shape. Such a decision is taken using the same dynamic programming approach applied independently on each brush path. In addition, wherever the policy described in Section 6.2 might consider merging brush paths, we ensure that the dynamic programming solution would not immediately split the merged path.

Conveniently, the abstraction algorithm for straight brush paths is easily adapted for circular arcs. We use the "modified least squares" method of Umbach and Jones [2003] to determine a best arc and compute the fitting error for a sequence of vertices.

Slowing motion When using strong abstraction, temporal artifacts may still occur. For example a circular arc can suddenly switch from concave to convex, or a single segment can break into two. To soften these transitions, we provide the following optional smoothing mechanism. Suppose the smoothed position of a path vertex at frame i is s_i. At frame $i+1$, rather than sending the vertex to its target position t_{i+1}, we move it a step in that direction $\vec{d} = (t_{i+1} - s_i)$:

$$s_{i+1} = s_i + \mathsf{min}(1, d_{\mathsf{max}}/\|\vec{d}\|)\,\vec{d}$$

where d_{max} specifies a maximum step size. Note that s_i and t_i are represented as offsets relative to the corresponding snake vertex and not absolute positions. Thus our mechanism leaves unchanged the motion of the model while smoothing the abstracted paths. The smoothing is applied on the endpoints of segments, whereas every arc vertices are processed (meaning that the latter shape deviates from exact circles during smoothed transitions).

6.2 Brush path parameterization

We need to parameterize these paths with temporal coherence during animation. Snakes track features smoothly evolving in 3D. In absence of visibility events, their intrinsic parameterization correlates with the 3D motion of the feature lines. But it does not correspond to screen-space arc-length, which induces compression artifacts. Moreover, occlusions and disocclusions produce sliding and popping. To avoid such effects, we recompute the parameterization at each frame based on the previous one, and we use the *self-similar line artmap* (SLAM) textures of Bénard et al. [2010] to avoid compression when zooming. To handle topological events – merging, typically – we allow multiple brush paths per snake, and forbid merging of two brush paths with different parameterization.

Parameterization fitting. Similarly to the 2D coherence strategy of Kalnins et al. [2003], we assume that a brush path represents the interaction between a drawing tool and 2D paper. Thus it should be uniform in screen-space, implying that its parameterization T is linear with respect to arc-length s: $T(s) = \rho s + \phi$. Besides, T should evolve according to the motion of the underlying snake. To account for these two goals, we follow the approach of Kalnins et al., a linear least squares fit of T, based on the parameter values from the previous frame propagated to the brush path of this frame (via the associated snake). However, our approach offers an extra degree of freedom: SLAM textures allow us to handle gracefully variations in phase and slope while preventing sliding artifacts, especially when zooming. Also note that our method does not need a fitting algorithm robust to outliers, such as *RANSAC* used by Bénard et al. [2010], because we are not mixing the parameterizations of multiple brush paths.

Topological events. Brush paths rely on the connectivity of their associated snakes. When a snake is updated so are its brush paths. For example, snake resampling (Section 4.2) causes resampling of the paths. When a snake is split (Section 5.1), the brush paths covering the break are also split; likewise for trimming, extension and deletion. However, when two snakes merge, their brush paths are not merged automatically, as their endpoints may differ in position and parameterization. Instead, a new merging operator ensures that the paths are continuous spatially and in parameter value. If not, both paths are kept (Figure 3). Otherwise, the two brush paths are joined and their mutual parameterization becomes:

$$\tilde{T}(s) = \tilde{\rho} s + \phi_1 \;\text{ where }\; \tilde{\rho} = \frac{\rho_2 l_2 + \phi_2 - \phi_1}{l_1 + l_2}$$

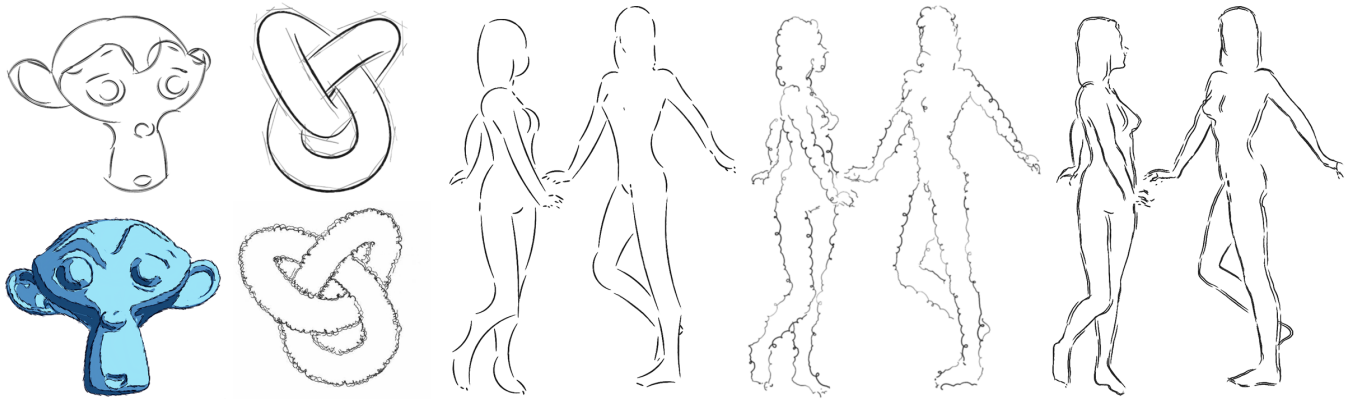
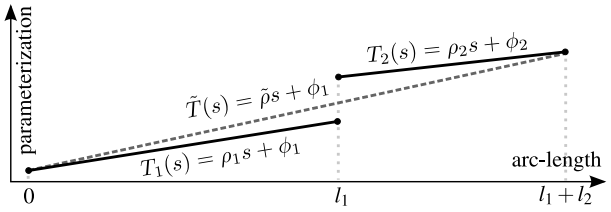


Figure 7: *Stylization examples, from left to right: monkey with circular arcs (above) and fuzzy SLAM texture over toon shading (below); knot with construction lines (above) and overdrawn wiggles (below); woman in two poses and three styles: arcs, loopy offsets, and overdrawn.*



To avoid brush path fragmentation over time, we also propose a *leveling* mechanism, similar in spirit to the "healing" process of Kalnins et al. [2003]. It progressively pushes the parameterization of two reasonably close brush paths to their common mean $\tilde{T}$, encouraging a fusion in subsequent frames. Note that we can explicitly deal with the periodicity of the parameterization during this process, which increases its efficiency.

7 Results

Snakes and their accompanying brush paths provide a robust, temporally coherent and evenly parameterized set of curves in screen space. Thanks to these qualities, we can explore styles that previous approaches were unable to apply on animated 3D scenes of moderate complexity (Figure 7 and Figure 11).

7.1 Style examples

We can produce standard drawing styles by using SLAM textures together with stroke attributes such as color and thickness. Tapering [Northrup and Markosian 2000] is provided and can be computed per brush path or per snake.

Offsets We can also apply displacement mapping to the vertices of each brush path, exploiting the fact that brush path geometry is persistent across frames. This mechanism supports, for example, the loopy effect added to the knot or the woman in Figure 7. Kalnins et al. [2002] synthesize screen space offsets using Markov random fields and apply them on the brush paths geometry. In contrast, we synthesize a texture of offsets using the SLAM synthesis approach of Bénard et al. [2010] which provides similar details at varying scales.

Overdraw The unit coverage computation at the snakes level not only prevents line clutter but also supports a form of "overdrawing," i.e. the overlap of a controlled number of brush paths on the same snake. The user specifies a target *overdraw* $\Omega_t \in \mathbb{R} > 1$. We measure the current overdraw Ω_c between two snake vertices as the number of brush paths overlapping this edge. For every brush path endpoint, we stochastically decide whether to extend, trim, or leave it unchanged, based on the probabilities:

$$\begin{aligned} P(\mathsf{extend}) &= \mathsf{max}(0, \gamma_{\mathsf{extend}}(1 - \Omega_c/\Omega_t)) \\ P(\mathsf{trim}) &= \mathsf{max}(0, \gamma_{\mathsf{trim}}(1 - \Omega_t/\Omega_c)) \end{aligned}$$

where γ_{extend} and γ_{trim} weight how rapidly the overdraw moves towards its target during animation. We can create even sketchier styles by combining overdraw with a small target length and random offsets per brush paths.

Speed lines Brush paths are persistent across frames and have their own linear parameterization. We take advantage of these two properties to produce speed lines (Figure 8) inspired by lines appearing in hand drawn comics.

Speed lines are created as new brush paths formed by chaining together copies of seed vertices over a series of frames. These seeds are chosen at locations undergoing noticeable motion, i.e. regions of brush paths where the velocity $\vec{v}$ has a strong component opposite to the projected surface normal. At every brush path vertex we compute $V = -\vec{v} \cdot \hat{n}$. We also compute the mean μ_{v} and standard deviation σ_{v} of V over all the vertices. A vertex is defined as a seed when $V > t\,\sigma_{\mathrm{v}} + \mu_{\mathrm{v}}$ with t a threshold that controls how much of the model receives these effects (we use $\frac{1}{2}$).

Speed lines can then be rendered as any other brush path using the effects described earlier. We generally find that it helps to trim the end of the speed lines after a given length.

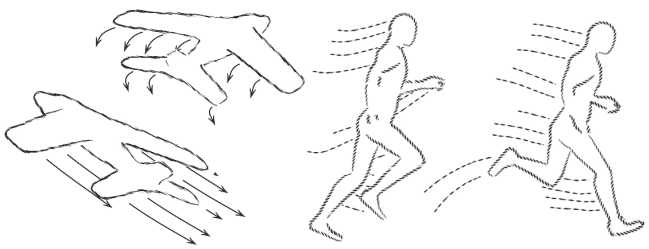


Figure 8: *Speed lines stylized with arrows and dots textures.*

(a) *Kalnins et al. [2003]* **(b)** *our approach*

Figure 9: *The Jot system of Kalnins et al. [2003] (a) generates many short strokes because the visible silhouettes produce short brush paths. Our approach (b) recovers connectivity using snakes, leading to fewer, longer strokes.*

7.2 Performance

Our implementation is primarily CPU-based, except for the 2D line extraction, blur for relaxation, and final rendering which rely on the GPU. The system performs at interactive rates on a recent machine (Intel Core 2 at 2.40 Ghz with 4 GB of RAM and a GeForce GTX 260) for models of moderate complexity (ranging from 20 to 30 FPS on the models shown in this paper). A major bottleneck is the readback from the GPU to CPU, which is performed at several stages. Techniques such as stream compaction [Horn 2005] could be used to reduce this cost. The sparse matrix inversion required for the relaxation step is performed efficiently using the CHOLMOD library [Chen et al. 2008].

7.3 Comparison with Previous Approaches

Our method is the first to allow a coherent parameterization of image-space feature lines. Implicit Brushes [Vergne et al. 2011] was the best available image space approach for line stylization. We obtain more complex styles with parameterization based effects (regular textures, tappering) and shape abstraction.

Previous object-space methods [Kalnins et al. 2003; Bénard et al. 2010; Buchholz et al. 2011; Karsch and Hart 2011] rely on visible stretches of object space feature lines as input, and their brush paths can not be longer than these features. The impact of brush path length on the final rendering can be seen in Figure 9. The short, unpredictable paths of the silhouette segments are clearly visible, while the snakes provide long brush paths that correspond with the major shape features of the model.

An alternative solution to our snake based approach could be to chain the small pieces of line together with an involved chaining algorithm (e. g., [Grabli et al. 2010]) before computing coherence using Kalnins et al. [2003] or Buchholz et al. [2011]. However, achieving interactive performances is very unlikely with such an approach. Moreover, vectorizing fully disconnected image space samples wouldn't be possible.

As a nice side effect, working in image space allows us to provide a LoD mechanism. While zooming in and out, the constant coverage radius ensures a unit line density for a given range of orientations (Figure 10). That way, brush paths appear and disappear naturally according to the camera motion as can be seen in the accompanying video.

Figure 10: *Zoom out on an octopus depicted with object space lines (left) and active strokes (right). The line screen space density varies with our approach avoiding line clutters (bottom).*

7.4 Limitations

Our method deals with image-space lines. We therefore have little knowledge of the 3D informations of the scene. It mostly has an impact on occlusions and disocclusions. For these visibility events a snake does not know what are the best feature samples to follow. It creates a tearing effect where during 2 or 3 frames the snake zigzags between two feature lines. This is the price to pay to have an image-based approach allowing for efficient frame-rates that would not have been possible to obtain with an object-space approach.

Another limitation of our approach is that for efficiency, decisions about snakes and brush paths are made locally. There is no explicit mid- or high-level understanding of the scene. For example, tracking T-junctions might be a way to improve the behavior at occlusions. As a result, our approach is well-suited for feature lines that are view-dependent and extracted from organic shapes, rather than fixed features such as those found in architectural scenes.

8 Conclusions and future work

This paper describes an approach for coherent evolution of feature line rendering for animated 3D scenes. Feature lines are approximated by snakes that advect with the motion of the scene. The snakes adjust their arrangement and topology using a new set of operators, in order to match the changing scene. Attached to the snakes, brush paths carry parameterization from one frame to the next with coherence, and can therefore be stylized and abstracted with temporal coherence. We introduce several new stylization effects shown in Section 7 and in the accompanying video.

Our pipeline is generic enough to be applied on video, by using vision algorithms to extract an approximated version of the data that we need: optical flow, feature lines, and depth. This endeavor would be particularly timely with the advent of commodity cameras that supply depth and optical flow along with color information.

Finally, while this system cannot optimize globally over an entire animation sequence (since we assume the frames are not known in advance) it might be possible to anticipate one or two frames in the future using something like a Kalman filter on the camera path and then use this information to broaden the scope of the optimization.

Acknowledgements

Thanks to Szymon Rusinkiewicz, Fredo Durand, Sylvain Paris, Pascal Barla and Aaron Hertzman for their insightful comments. Many of the images shown in this paper are rendered from 3D models courtesy of the Princeton Shape Benchmark, the Stanford Computer Graphics Laboratory, the Utah 3D Animation Repository, and the Computer Graphics Group at MIT. This work was sponsored in part by the ExploraDoc program of the region Rhône-Alpes, France.

Figure 11: *Animated models stylized with active strokes: running elephant rendered with circular acrs (left) and segments (middle); animated hand in three poses with SLAM texture and offsets.*

A Line Extraction with Steerable Filters

Freeman and Adelson [1991] describe an image contour extractor using steerable quadrature pair filters. The quadrature pair responds equally to step-edges and fine lines in the image, providing a consistent, intuitively appealing set of line samples. The steerable formulation provides smooth, precise estimation of orientation, which we use for the tangent at each contour pixel.

We apply the contour extraction to a shaded image, allowing us to capture lighting as well as shape effects (similar to [Lee et al. 2007]). The image is optionally augmented with depth information to ensure that depth discontinuities are captured by lines. The input to the method is a scalar image $I(x, y) = L(x, y) + \beta z$ where x and y are the screen space pixel coordinates, $L(x, y)$ is any shaded image, and β is a parameter that controls the influence of depth z. We apply to I the second-order quadrature filter pair contour detector of Freeman and Adelson [1991]. Finally, we perform non-maxima suppression using the estimated orientation to find the set of line samples. The parameters of the method are the filter kernel size, a noise threshold, and β.

The filter pair can be computed very efficiently on the GPU using the steerable formulation with seven basis filters. Each basis filter can be expressed as the composition of a Gaussian and a small derivative kernel, where the Gaussian can be computed with a separable formulation. We use this method for all the results with the exception of Figure 11.

References

AGARWALA, A., HERTZMANN, A., SALESIN, D. H., AND SEITZ, S. M. 2004. Keyframe-based tracking for rotoscoping and animation. *ACM Transactions on Graphics 23*, 3, 584–591.

AGARWALA, A. 2002. Snaketoonz : A semi-automatic approach to creating cel animation from video. In *Proceedings of NPAR 2002*, 139–146.

BARLA, P., THOLLOT, J., AND SILLION, F. X. 2005. Geometric clustering for line drawing simplification. In *Proceedings of EGSR 2005*, 183–192.

BÉNARD, P., COLE, F., GOLOVINSKIY, A., AND FINKELSTEIN, A. 2010. Self-Similar Texture for Coherent Line Stylization. In *Proceedings of NPAR 2010*, 91–97.

BUCHHOLZ, B., FARAJ, N., PARIS, S., EISEMANN, E., AND BOUBEKEUR, T. 2011. Spatio-temporal analysis for parameterizing animated lines. In *Proceedings of NPAR 2011*, 85–92.

CHEN, Y., DAVIS, T. A., HAGER, W. W., AND RAJAMANICKAM, S. 2008. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software 35*, 3, 1–14.

COLE, F., DECARLO, D., FINKELSTEIN, A., KIN, K., MORLEY, K., AND SANTELLA, A. 2006. Directing gaze in 3d models with stylized focus. In *Proceedings of EGSR 2006*.

DELINGETTE, H., AND MONTAGNAT, J. 2000. New algorithms for controlling active contours shape and topology. In *Proceedings of ECCV 2000*, 381–395.

FREEMAN, W. T., AND ADELSON, E. H. 1991. The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence 13*, 891–906.

GRABLI, S., DURAND, F., AND SILLION, F. X. 2004. Density measure for line-drawing simplification. In *Proceedings of PG 2004*, 309–318.

GRABLI, S., TURQUIN, E., DURAND, F., AND SILLION, F. X. 2010. Programmable rendering of line drawing from 3D scenes. *ACM Transactions on Graphics 29*, 2, 18:1–18:20.

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

HERTZMANN, A. 2001. Paint by relaxation. In *Proceedings of the International Conference on Computer Graphics*, 47–54.

HORN, D. 2005. Stream reduction operations for GPGPU applications. In *GPU Gems 2*, M. Pharr, Ed. ch. 36, 573–589.

JI, L., AND YAN, H. 2002. Robust topology-adaptive snakes for image segmentation. *Image and Vision Computing*, 147–164.

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.

KALNINS, R. D., DAVIDSON, P. L., MARKOSIAN, L., AND FINKELSTEIN, A. 2003. Coherent stylized silhouettes. *ACM Transactions on Graphics 22*, 3, 856–861.

KARSCH, K., AND HART, J. C. 2011. Snaxels on a plane. In *Proceedings of NPAR 2011*, 35–42.

KASS, M., WITKIN, A., AND TERZOPOULOS, D. 1988. Snakes: Active contour models. *International Journal of Computer Vision 1*, 4, 321–331.

LEE, Y., MARKOSIAN, L., LEE, S., AND HUGHES, J. F. 2007. Line drawings via abstracted shading. *ACM Transactions on Graphics 26*, 3, 18:1–18:5.

MCCRAE, J., AND SINGH, K. 2009. Sketching piecewise clothoid curves. *Computers & Graphics (Sketch-Based Interfaces and Modeling) 33*, 4.

MCINERNEY, T., AND TERZOPOULOS, D. 1995. Topologically adaptable snakes. In *Proceedings of ICCV '95*, 840–845.

MCINERNEY, T., AND TERZOPOULOS, D. 2000. T-snakes: Topology adaptive snakes. *Medical Image Analysis 4*, 73–91.

NORTHRUP, J. D., AND MARKOSIAN, L. 2000. Artistic silhouettes: a hybrid approach. In *Proceedings of NPAR 2000*, 31–37.

RASKAR, R., AND COHEN, M. 1999. Image precision silhouette edges. In *Proceedings of I3D 1999*, 135–140.

SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible rendering of 3-D shapes. In *Proceedings of ACM SIGGRAPH 1990*, vol. 24, 197–206.

SHESH, A., AND CHEN, B. 2008. Efficient and dynamic simplification of line drawings. *Computer Graphics Forum 27*, 537–545.

UMBACH, D., AND JONES, K. 2003. A few methods for fitting circles to data. *IEEE Transactions on instrumentation and measurement 52*, 6, 1881–1885.

VERGNE, R., VANDERHAEGHE, D., CHEN, J., BARLA, P., GRANIER, X., AND SCHLICK, C. 2011. Implicit brushes for stylized line-based rendering. *Computer Graphics Forum 30*, 513–522.