Volume 0 (1981), Number 0 pp. 1–13 COMPUTER GRAPHICS *forum*

Line-Drawing Video Stylization

N. Ben-Zvi[1,2], J. Bento[1,3], M. Mahler[1], J. Hodgins[1], A. Shamir[1,4]

[1]Disney Research, [2]The Hebrew University, [3]Boston College, [4]The Interdisciplinary Center,
(nirbenz@cs.huji.ac.il, bentoayr@bc.edu, moshe.mahler@disneyresearch.com, jkh@disneyresearch.com, arik@idc.ac.il)

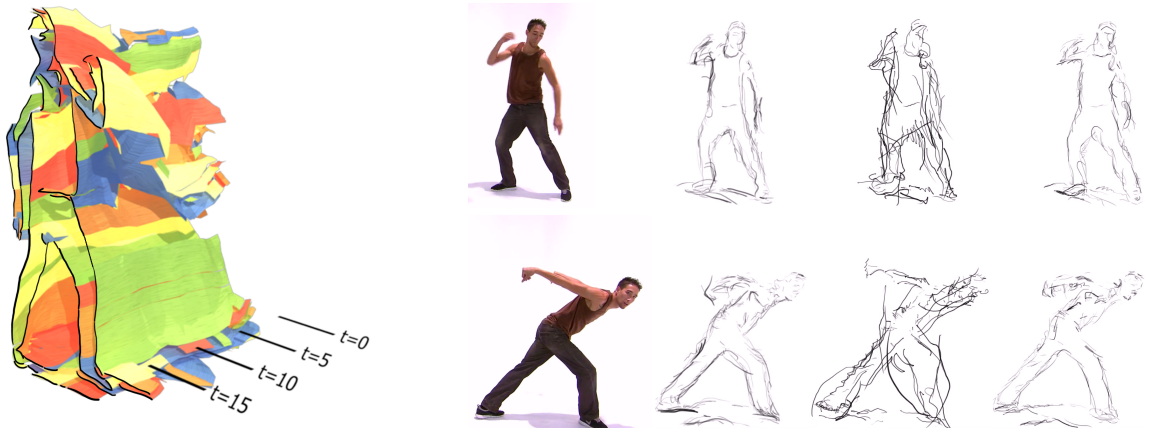


Figure 1: *Our algorithm creates spatio-temporal 3D sheets (left) from curves extracted from video, and uses these to render a line-drawing style animation following different artists' styles. Example of two frames in three styles are shown on the right.*

Abstract

We present a method to automatically convert videos and CG animations to stylized animated line-drawings. Using a data-driven approach, the animated drawings can follow the sketching style of a specific artist. Given an input video, we first extract edges from the video frames and vectorize them to curves. The curves are matched to strokes from an artist's library, while following the artist's stroke distribution and characteristics. The key challenge in this process is to match the large number of curves in the frames over time, despite topological and geometric changes, allowing to maintain temporal coherence in the output animation. We solve this problem using constrained optimization to build correspondences between tracked points and create smooth sheets over time. These sheets are then replaced with strokes from the artist's database to render the final animation. We evaluate our tracking algorithm on various examples and show stylized animation results based on various artists.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Although labor intensive to produce, hand-drawn animations have an appeal that has not yet been replicated by computerized rendering techniques. Some of the appeal of hand-drawn animation comes from the natural "noise" of the medium, including small frame to frame changes in the lines and silhouettes and even frame-to-frame imperfections in the drawings. This noise creates a sense of liveliness that computer-rendered animations sometimes lack because of their perfect and clean look.

Figure 2: *Three sets of curves in three consecutive frames: due both to motion and edge detection problems (e.g. note the blur in the example on the left) curves can appear, disappear, change their position, shape and topology. Our method succeeds in simultaneously tracking such complex curve-sets over an entire video.*

In this work, we concentrate on animations created as line drawings. We present a method to automatically convert live action footage or rendered animation (the "input video"), to what appears to be a hand-sketched animation. Furthermore, we match the sketch-style of a specific artist by taking as input a small set of static sketches by the artist that allow us to model the artist's style. Following the work of Berger et al. [BSM*13], we create a library of the artist's strokes and gather statistical measures. However, in contrast to Berger et al., we use them to render a *dynamic* animation as output (see Figure 1, right).

The biggest challenge in rendering such dynamic line-drawing animations is maintaining and controlling temporal coherence. Temporal coherence demands continuity in shape and movement throughout the animation. Without sufficient temporal coherence, the animation will become too "noisy", and may contain perceptually troublesome artifacts such as flickering and popping.

Our method first converts the frames of a given input video to vector curves by using edge detection and vectorization on each frame. Ignoring temporal coherence, each curve in each frame can be matched and replaced independently by strokes from the artist's library. However, this results in very noisy animation output (see examples in accompanying video). To create more coherent results, curves should be tracked across frames, and replaced by the same stroke from the library, applying local changes such as rotation and translation to fit the curve motion. However, over time, curves can also change shape and topology in an arbitrary manner – they can appear or disappear, as well as split and merge. Tracking a large set of curves over time becomes a serious challenge (see Figure 2).

We present a method that can simultaneously track a large set of curves over time. We convert the tracking problem to pairwise matching, and guide the matching of curves by matching points. The point matching problem is solved using constrained optimization. In addition, our approach offers control over the tradeoff between style and coherence by using a single parameter to determine the duration of typical tracked curves.

The contributions of this work are first, a matching algorithm for a large set of curves, that is used to construct long sequences of tracked curves through time, which we call *sheets* (see Figure 1, left). Second, a fully automatic method to create animations in line-drawing style from input videos, using a set of static sketches as examples and controlling the tradeoff between style and coherence (see Figure 1, right).

We evaluate our curve-set tracking algorithm on various inputs where movements range from simple and smooth to erratic, using a statistical measure – the histogram of the duration of the curves. We demonstrate the power of our video stylization approach by modeling the style of six different real artists and a few "artificial" ones (using strokes deliberately drawn in a unique manner) on a number of videos originating from both live footage and graphics rendering.

2. Previous Work

Motion pictures and short films have already used computer-assisted techniques for Rotoscoping (converting live action footage to animation, for example, in "A Scanner Darkly", "Waking Life", and "Le Congress"). The effect created is highly engaging and the approach allows modifying the original imagery as well as adding new visual layers. However, these hybrid techniques still demand extensive manual processing on the frames of the video themselves, while we target a more automatic stylization technique.

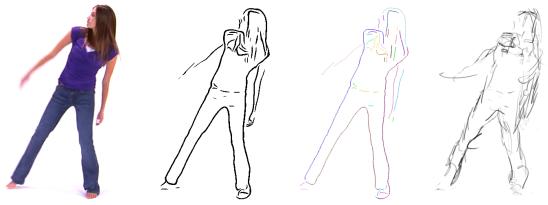
Our work falls into the broad category of video stylization [KCWI13], where one of the main challenges is temporal coherence. A good overview of the temporal coherence problem in various approaches for video stylization is given in [BTC13]. Several of these stylization techniques address rendering styles of the whole frame such as comic-style in [WXSC04, WCH*11, WOG06] or painterly rendering in [OH12].

Our technique can be considered as "segmentation" stylization as we first extract edges from the frames and then convert them to temporal sheets. Collomosse et al. [CRH05]

define a similar construct called "stroke surface" and use them for video stylization in Video Paintbox [CH05]. However, these surfaces are boundaries between segmented regions, and better methods for video segmentation exist such as [GKHE10]. Boundaries of video segmentation regions can be more temporally stable than edges extracted from frames, but they cannot be used for our purpose as they do not detect and cannot track open curves. Open curves constitute a large percentage of our sheets and are important for the style and appearance of our animations (see Section 7).

Other contour tracking methods that support open curve tracking in video exist [PM08, GS11]. However, these works usually track a single, or a small number of designated curves through time, while we have a large set of curves that form many-to-many relations and include many topological changes. Xu et al. [XSQ13] extract ridges, valleys and silhouette curves in 3D animations and optimize the matching between them. Their method relies on 3D geodesic distances of the curves (paths) on the objects, which cannot be simply extended to video curve tracking. Another possible approach is to use snakes [KWT88]. In SnakeToonz [Aga02], snake loops are defined over the video and tracked over time, arriving at a video segmentation that can then be used to create cartoon animations. However, such approaches rely on tracking of points over a motion field defined from the video, they often need manual intervention, and must include special handling for topological changes of the curves. In [BLC*12] real-time stylized rendering of 3D objects is presented. Line features are extracted from every frame and tracked using active strokes, which are snake-like curves. However, their method targets 3D scene rendering, and cannot handle the videos used in our examples, which are more complex and include "imperfect" lines that are less stable over time. There are other methods for tracking silhouettes in 3D scenes but they also rely on 3D information. Kalnins et al. [KDMF03] track 3D positions of silhouette samples and project them back on the next frame, while Buchholz et al. [BFP*11] build space-time surfaces from animated lines (curves) using a similar graph construct as ours but track topological events in 3D.

The closest method to our curve tracking is presented in [AHSS04]. Their approach adds automatic tracking to a user-driven keyframe system. User-defined curves on keyframes are interpolated to in-between frames by optimizing a space-time energy function containing both shape and image terms. The user can define constraints on the optimization by manipulating control points on any frame in the sequence. However, scaling such an optimization solution to handle our large sets of curves in a fully automatic manner would not be easy. More recently, BetweenIT [WNS*10] defined a method to match the curves, which are termed "strokes", in two line-drawing keyframes to create in-between strokes (curves). However, they match the strokes (curves) directly in a greedy manner based on length and proximity (the area between the strokes), and would have difficulty addressing the numerous topological changes occurring in our sets of curves. In their method, when a curve in one keyframe maps to two curves in the next, it must be duplicated. This method was used in [NSC*11] to control the level of noise in a sketchy animation. The tradeoff presented there between noise and coherence is similar to our tradeoff between style and coherence.



(a) (b) (c) (d)

Figure 3: *Converting a video frame to line drawing animation: (a) Original frame (b) Extracted edges (c) The set of curves in the frame (c) Final rendering of the strokes replacing the curves using a specific artist's stroke library.*

Apart from [BSM*13], all the above mentioned works demonstrate procedural stylistic effects and do not address the question of replicating a specific artist's stroke style. There are other works that learn specific styles of strokes in drawings such as [FTP03] and [LYFD12] for line drawing, or [KNBH12] for hatching. These works use data-driven approaches, but build a procedural or parametric model that is used when rendering the output. Similar to Berger et al. [BSM*13], we use the strokes of artists directly and statistics learned from their drawings, allowing us to mimic the actual style of an artist by reusing his or her strokes in a new image.

3. Overview

Drawing a video in the style of a line-drawing animation by hand is a painstaking process. The key idea of our work is to use some sample drawings of an artist to automatically convert a video to an animation resembling the line-drawing style of that artist. We utilize the data-driven approach presented by Berger et al. [BSM*13], where sample drawings of an artist are used to build a library of strokes defining the line-drawing style of the artist.

Berger et al. collected a database of portrait sketches from several artists by displaying a reference photograph of a face to the artists, and asking them to sketch the portrait digitally using a stylus pen. All sketches were captured using a Wacom tablet, allowing the artist to modify the brush parameters but not allowing erasing or undoing strokes. Each stroke was captured as a parameterized directed curve along with the pen parameters, and stored as a transparent bitmap. In our work, we use their stroke libraries and statistics along

with new data defining some "artificial" styles to render our animations.

Given an input video of length T frames, we first convert each frame f_t at time t $(1 \leq t \leq T)$ to a set of curves $C_t = \{c_{t,i}\}$ using edge detection and vectorization. We use the methods of [KLC07] for edge detection, and [NHS*13] for vectorization, although other methods could be used as well (see supplemental video for examples). To render a line-drawing of frame t, each curve $c_{t,i} \in C_t$ can be converted to a set of strokes based on the characteristics of a given artist and using his/her strokes library (see Figure 3). Converting the set of curves of each frame to strokes separately leads to noisy and incoherent animation results. To promote coherence, the same strokes must be used over time by adapting them to replace similar curves in a continuous sequence of frames. Therefore, our main challenge is to define a tracking algorithm for a large set of curves over time.

Our method (Section 4) constructs long sequences of tracked curves over time by building a match between the sets of curves in pairs of consecutive frames. Each tracked curve creates a *sheet* in a 3D space-time cube. However, instead of matching curves between frames, we match the points that define the curves, while constraining only contiguous parts of the curves at $t-1$ to be mapped to contiguous parts of the curves at t (see Figure 4). This step is formulated as a constrained optimization problem between every pair of frames (Section 4.1). We take advantage of motion continuity by predicting where the points of frame $t-1$ will move to in frame t (Section 4.2). Instead of the actual points, we match the prediction of the points of frame $t-1$ to the points in frame t, resulting in a more accurate matching that relates curves and sub-curves through time.

To construct the sheets (Section 4.3), the curve-to-curve relations can be read from the point-to-point matchings found. First, the point matches are used to construct a trellis graph of the whole video sequence. Second, the graph is converted to a set of simple paths, each one defining a sheet, which represents one tracked curve through time. To render an animation (Section 5), each sheet is replaced by a stroke (or a set of strokes) from a designated stroke library, adapting it through time, thus maintaining temporal coherence and creating a smoother animation. Our method also allows control over the amount of coherence vs. style of the animation allowing to create different stylistic effects.

4. Tracking Sets of Curves

To track a large set of curves $C_t = \{c_{t,i}\}$ extracted from every time frame t separately, we transform the problem into one of matching. We only match curves between pairs of consecutive frames, although we also utilize motion dependencies based on previous frames. We compose these pairwise matches together to track curves along the whole sequence. There are two competing principles at play in this task. On the one hand, to create coherent results, we want

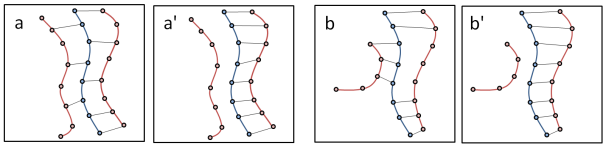


Figure 4: *Given two sets of curves (red and blue in these examples), matching only based on point distances (see Eq. 1) can create non-consecutive mapping as in (a) and (b). Our constrained optimization (see Eq. 2) prefers continuous solutions as in (a') and (b').*

to match as many curves as possible. On the other hand, matching curves that are too different in shape means that the strokes used for rendering might not fit the different curves well. Curves can change dramatically between frames because, in addition to geometric and shape changes, a curve can undergo topological changes when moving from one frame to the next (see Figure 2):

1. A curve can appear or disappear.
2. A curve can split into several curves.
3. Several curves (or parts of curves) can merge into one curve.

The problem is to find a valid mapping between the set of curves (and sub-curves) of C_{t-1} and the set C_t. This map must cover as many curves as possible and must map curves to similar curves (or parts of curves) undergoing various topological changes.

At first glance, this problem appears to be a variant of the perfect matching problem with the additional requirement that we can have many-to-one and one-to-many matchings. Using a distance measure between curves, we could find a matching that minimizes the average distance between the two sets of curves. The many-to-many mapping could be solved, for instance, using multiple copies of each element on both sets (see [AKJO07]). However, such a solution is not sufficient for our application because solving the matching problem at the level of curves will not tell us how sub-curves match to each other, or how to split them when needed. Complex cases for mapping sub-curves often occur when topological changes take place (see example in Figure 6d). Moreover, the definition of a good curve-distance is by itself a challenge, especially when curves can undergo various deformations through time.

Our key idea is to formulate this problem as a point matching problem instead of a curve matching problem, but leverage the curve structure: the neighbors-relation and ordering of points both spatially on the curves, and temporally across frames. This formulation allows continuous sub-curves to be matched and eliminates the need for a curve distance measure.

4.1. Two-Frame Point Matching

We solve the point-to-point matching problem by formulating and solving a minimization problem. Each curve in C_t and C_{t-1} is defined by a set of sample points. We need to produce a matching between the individual points in frame $t-1$ and the individual points in frame t. However, such a match must follow certain guidelines to produce a good matching between curves later. First, we want to match points in the two frames whose spatial position is close. Second, we want to match as many points as possible and prefer a one-to-one mapping because each point represents a part of a curve. Last, but not least, we want to promote matching consecutive points belonging to the same curve in frame $t-1$ to (consecutive) points belonging to the same curve in frame t (Figure 4), because we want to bias the solution towards matching full curves (or sub-curves).

Without loss of generality, assume that the matching is between frame f_1 and frame f_2. f_1 has a set of n_1 points $\{p_1, p_2, ..., p_{n_1}\}$ and f_2 has a set of n_2 points $\{q_1, q_2, ..., q_{n_2}\}$. We write $i \in f_1$ ($j \in f_2$) when we want to refer to the point in f_1 (f_2) with index i (j) and coordinate p_i (q_j). These points are arranged into two sets: C_1 of m_1 curves in f_1 and C_2 of m_2 curves in f_2. For simplicity of notation, we assume each curve c in C_1 (C_2), is represented by an ordered set of indices that index contiguous points in the frame f_1 (f_2). For example, $c = \{5,6,7,8\}$ can define the curve passing through points $\{p_5, p_6, p_7, p_8\}$.

The point-to-point matching problem can be defined as an optimization problem on a bi-partite graph. The nodes on each side of the graph represent the points in the two frames respectively. The weighted edges of the graph connect any pair of points $(i,j), i \in f_1, j \in f_2$ with a $\text{cost}(i,j) = \|p_i - p_j\|$. For efficiency, we remove edges whose $\text{cost}(i,j) >$ **MaxDist**, for some **MaxDist** > 0 (we use 5% of the diagonal of a frame). The optimization is defined as an integer program over three sets of binary variables $\{w_{(i,j)}\}_{(i,j)\in\{0,1\}}$, $\{e_i\}_{i\in\{1,...,n_1\}}$ and $\{d_j\}_{j\in\{1,...,n_2\}}$. $w_{ij} = 1$ indicates that i^{th} point in f_1 is matched to the j^{th} point in f_2, and it is 0 otherwise. Variable $\{e_i\}$ ($\{d_j\}$) is 1, if point i (j) in f_1 (f_2) remains unmatched, or 0 otherwise. Note that $w_{ij} = 1$ implies that $e_i = 0$ and $d_j = 0$.

We use a matrix notation to describe our optimization problem. We define a cost matrix $\mathbf{M} \in \mathbb{R}^{n_2 \times n_1}$ whose entries at position (i,j) are $\text{cost}(i,j)$ if $\text{cost}(i,j) <$ **MaxDist**, or ∞ otherwise, a matrix $\mathbf{W} \in \mathbb{R}^{n_1 \times n_2}$ whose entries are the variables $w_{(i,j)}$, and vectors $\mathbf{e} \in \mathbb{R}^{n_1}$ and $\mathbf{d} \in \mathbb{R}^{n_2}$ whose entries are the variables e_j and d_i respectively. Using these notations we define an integer program as follows:

$$\begin{aligned} &\text{Minimize}_{\mathbf{W},\mathbf{d},\mathbf{e}}\{\text{tr}(\mathbf{MW}) + \alpha\|\mathbf{e}\|_1 + \alpha\|\mathbf{d}\|_1\} \\ &\text{such that } \mathbf{W}\vec{\mathbf{1}} = \vec{\mathbf{1}} \text{ and } \mathbf{W}^\top\vec{\mathbf{1}} = \vec{\mathbf{1}} \end{aligned} \tag{1}$$

The first term minimizes the cost of the edges used for matching. The second and third terms make sure we match as many points as possible (otherwise matching none would be optimal). In this case, integer solutions can be found by solving a relaxed version obtained by replacing the binary constraint on the variables $[...] \in \{0,1\}$ by inequality constraints $1 \geq [...] \geq 0$. Note that, even without rounding values, the relaxed optimization problem produces only $\{0,1\}$ solutions because the constraints are totally unimodular. However, this solution only provides the least costly assignment between two sets of points with no relation to the curves. Points in one frame will match their closest point in the other frame, disregarding the curve structures (see Figure 5).

Following the above guidelines, we add additional terms and (soft) constraints so that neighboring points on a curve in f_1 will match neighboring points on the same curve in f_2. We further define two point membership matrices $\mathbf{B_1} \in \mathbb{R}^{n_1 \times m_1}$ and $\mathbf{B_2} \in \mathbb{R}^{n_2 \times m_2}$ whose entries $b_{1(i,k)} = 1$ if point i in frame f_1 belongs to curve k in C_1, and $b_{2(j,k)} = 1$ if point j in frame f_2 belongs to curve k in C_2, or 0 otherwise. Note that the product of $\mathbf{WB_2}$ produces a matrix in which entry (i,k) is 1 if point i in frame f_1 connects to curve k in frame f_2, and 0 otherwise. Similarly, $(\mathbf{B_1}^\top\mathbf{W})$ is a matrix whose entry (k,j) is 1 if point j in frame f_2 connects to curve k in frame f_1, and 0 otherwise.

Lastly, we define block-structured matrices $\mathbf{H_1} \in \mathbb{R}^{n_1 \times n_1}$ and $\mathbf{H_2} \in \mathbb{R}^{n_2 \times n_2}$, where for each set of k points defining a curve we have a block of dimensions $k \times k$ of the form:

$$\begin{pmatrix} 1 & -1 & 0 & \dots & 0 & 0 \\ 0 & 1 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & -1 \\ 0 & 0 & 0 & \dots & 0 & 0 \end{pmatrix}.$$

Multiplying a matrix with matrices $\mathbf{H_1}$ and $\mathbf{H_2}$ will take consecutive differences between rows that belong to the same curve. Hence, if $\mathbf{W}$ is a matching matrix, $w_{i,j} \in \{0,1\}$, then the product $\mathbf{H_1WB_2}$ is a matrix with the following properties. (a) The r^{th} row is $\vec{0}$ if point r and point $r+1$ on the same curve in f_1, match the same curve in f_2 or no curve at all. (b) The r^{th} row has exactly one 1 and one -1 (all other entries being zero) if the points r and $r+1$ match to different curves in f_2. (c) The r^{th} row has only a single ± 1 (all other entries being zero) if one of the two points has a match and the other does not. Therefore, the L_1-norm of each row of $\mathbf{H_1WB_2}$ is either 0, 1 or 2, and is only 0 when all consecutive points on the same curve in f_1 match the same curve in f_2, or both match to nothing. If we use the notation $\|\cdot\|_1$ on matrices to mean the sum of the absolute values of all entries instead of the regular L_1-norm, then $\|\mathbf{H_1WB_2}\|_1$ is a proxy for the number of times that curves in frame f_1 switch their matching to curves in frame f_2. Similarly, $\|\mathbf{B_1}^\top\mathbf{WH_2}\|_1$ is a proxy for the number of times that curves in frame f_2 switch their matching to curves in frame f_1. We now write our full

optimization problem in its relaxed form:

$$\begin{aligned} &\text{Minimize}_{\mathbf{W},\mathbf{d},\mathbf{e}}\{\text{tr}(\mathbf{MW}) + \alpha\|\mathbf{e}\|_1 + \alpha\|\mathbf{d}\|_1 + \\ &\qquad \beta\|\mathbf{H_1 W B_2}\|_1 + \beta\|\mathbf{B_1}^\top \mathbf{W H_2}\|_1\}, \\ &\text{subject to } \mathbf{W} \geq 0,\ \mathbf{e} \geq 0,\ \mathbf{d} \geq 0, \\ &\qquad \mathbf{W}\vec{\mathbf{1}} + \mathbf{e} = \vec{\mathbf{1}},\ \mathbf{W}^\top\vec{\mathbf{1}} + \mathbf{d} = \vec{\mathbf{1}}. \end{aligned} \tag{2}$$

The fourth and fifth terms penalize matching consecutive points in f_1 (f_2) to points on different curves in f_2 (f_1). The first three constraints force the relaxed variables to be positive and the last two constraints make sure that, in its $0/1$ representation, each point is matching to a single point (note that some points can still remain unmatched in both frames). α and β scale the terms to be of the same magnitude and can be used to weigh the different terms. This relaxed problem can be reduced to linear programming (see Appendix A), but we must round the variables in the end to 0 or 1. This scheme produces a solution that is suboptimal, but in practice works well for our application. Note that although we can compute the cost coefficients using different metrics, it is important to use the L_1-norm in the last two terms, instead of L_2, as it concentrates the error in a small number of places instead of distributing it. In our case, this term limits the switches in matching to different curves to a small number of places.

4.2. Leveraging Curve Dynamics

Because the points we match belong to curves that move and deform through time, we can leverage the curve dynamics similar to the use of Kalman filters [GA93] to enhance the matching and to account for unmatched points. Instead of matching the points of f_{t-1} to points of f_t, we extrapolate the curves of f_{t-1} based on k previous frames to predict how these curves will move in the next timestep t, and then match the *predicted* positions of points of frame f_{t-1} to the actual points of frame f_t.

For most points $p_i \in c$ in each curve $c \in C_{t-1}$, we can fit a polynomial $P(\cdot) \in \mathbb{R}^2$ through the sequence of k points that were matched up until point p_i (we use k=5). Using this polynomial, we can predict p_i', the position of point p_i at frame t, and compute the vector $\Delta p_i = p_i' - p_i$. Next, we define the predicted position of point p_i as $\tilde{p}_i = p_i + \mu\Delta p_i$, where $\mu \in [0,1]$ is a weighting factor pre-specified by the user (we use $\mu = 0.5$). Applying the same procedure for all points in curve $c \in C_{t-1}$ that have previous matches, we get a prediction of where the curve c would move at frame t and can use it to better match the points along c to points in C_t. We can also predict the positions of missing points on c (for example, points that did not have previous matches), by interpolating their position using the points on c that have a predicted value.

We use this procedure to find Δp_i for all curves that contain more than 30% matched-points. For all other curves, we move their points using the average delta vector of all curves in the frame. Finally, instead of computing the pairwise matching between frames $t-1$ and t as described in Subsection 4.1, we compute a matching between $\tilde{C}_{t-1}$, the *predicted* curves of frame $t-1$ and C_t. Because there is a one-to-one mapping between the points of $\tilde{C}_{t-1}$ and C_{t-1}, this match also provides the matching between C_{t-1} and C_t.

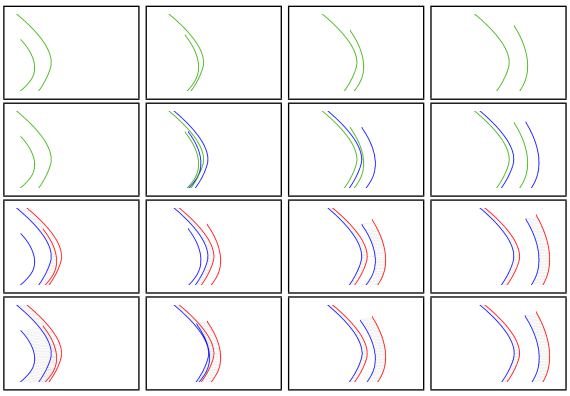


Figure 5: *Tracking sets of curves leveraging curve dynamics and curve structure. In the top row are four frames of two curves moving through time, where one curve is faster and crosses the other. The curves in each consecutive pair of frames need to be matched. The actual matching uses the predicted positions of the curves, shown in blue in the second row based on our method. Note how this prediction brings the curves closer to the position of the curves in the next frame (depicted in red in the last two rows). Setting $\beta = 0$ (which amounts to using only Eq. 1 for optimization) creates the least costly assignment between the two sets of points disregarding the curves structure (third row). Note that the blue curves are not in the same position because the matching also affects the predicted extrapolation. Our full optimization (Eq. 2) creates better matching and better predictions, allowing correct tracking of the curves (fourth row).*

An example can be seen in Figure 5. The green curves are the original curves, C_{t-1}, and the blue curves are their predicted positions $\tilde{C}_{t-1}$ based on our optimization. These blue curves are matched to the red curves, which are the curves at time t. Compared to simple point-based matching (using Eq. 1), our full optimization (using Eq. 2) creates a match between the points that better preserves the curve structures, and also supports a better prediction of the curves' motion. This feedback is a key characteristic of our method: better matching supports better prediction, and better prediction supports better matching.

4.3. Sheets Construction

Using the point correspondences, we define a trellis graph where each slice is a frame containing one node for each curve in that frame (Figure 6(a)). Two nodes in two consecutive slices are connected if they have a large enough matching overlap (5% of their length) that is long enough (10 pixels). Our goal is to convert this graph into a set of simple

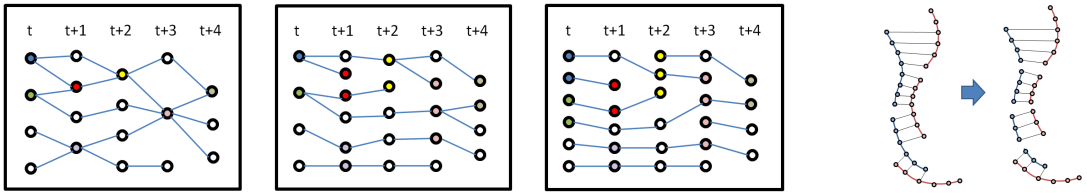


Figure 6: *Converting the curves-matching trellis graph to simple paths using a two-pass greedy approach. We traverse the graph (a) first from left to right (b), and then from right to left (c). In each pass, we split nodes (and their respective curves) that have more than one incoming edge. Because our matching optimization prefers consecutive matchings, each curve can be split into simple consecutive parts as in (d).*

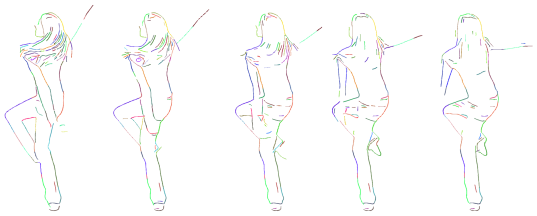


Figure 7: *Visualization of sheets: each sheet is displayed in a consistent color through time.*

paths of connected nodes through time. Each path will define one sheet, which is a tracked curve through time.

Because of our optimization constraints, we can split curves at time t to sub-curves that match a single sub-curve at $t+1$, and vice versa (Figure 6(d)). Consequently, we can use a simple greedy approach to convert the graph into simple paths. We traverse the graph first from left to right, and then from right to left. In each pass, we either split the nodes that include more than one incoming edge or remove an edge (Figure 6(b-c)). Every node split imposes a split on its respective curve between the two points where the matching changes. We split a node only if the sub-curves created are long enough (10 pixels) and constitute a large enough part of the curves (again, 5% of their length).

Once we have simple paths, we can define each graph path as a sheet (Figure 1, left), which spans a continuous subset of frames. Figure 11 shows the distribution of sheet duration in the male dancer movie example. We re-parameterize all curves in a sheet by sampling them using cord-length to have the same number of points and a one-to-one mapping between the points. The intersection of a sheet with a specific frame it spans is a curve belonging to that frame that should be rendered (Figure 7).

5. Line Drawing Stylization

To create line-drawing animation in a given style, we replace the curves found in the previous steps by a set of strokes from a specific artist's library. We also follow the statistics gathered from the artist's drawings on average curve length and amount of overlap between strokes to fit his/her drawing style. To achieve temporal coherence, curves belonging to the same sheet will use the same set of strokes.

We use the curve-skeleton of the strokes to match the curves in the frame. The matching descriptor vector combines the curve length, a histogram of the shape context [BMP02], and a histogram of the curvature along the curve. Each curve is sampled uniformly (we use 400 sample points, unless the distance between two points is smaller than a pixel). The shape context descriptor is calculated for each sample point, using 12 logarithmic bins for distances and 12 bins for orientation. All shape context descriptors of all points are combined to a normalized 2D histogram. The curvature of all sample points is calculated and gathered into a histogram with 12 bins. The length of the stroke is calculated by summing up the lengths of the segments between each two sample points.

Matching the first curve in each sheet to a stroke and fitting this stroke to the sheet over time (e.g. using ICP and rigid transformations) can achieve high temporal coherence (as the same stroke is rendered over time), but will not fit well the curves' shapes over time. Hence, each sheet is replaced with several strokes: first, a stroke is matched not only to the first curve but also to other curves in the sheet (while still fitting them to all curves), and second, the curves of the sheet are segmented to sub-curves, and each sub-curve is replaced with a stroke. We also expand or reduce the sub-curves (using extrapolation or interpolation), so that they overlap a little. The sub-curve lengths, extrapolation and amount of overlap are based on the statistics gathered for the chosen artist. These enhancements create a richer rendering style that mimics the look-and-feel of hand drawn sketches.

The greedy approach for curve splits defined in Section 4.3, favors sheets that have a longer duration in time at the cost of shorter length in space. Because the longer the duration of the sheets is, the more splits will occur in curves to conform to these sheets. Shorter curves are also simpler, and are replaced by simple strokes, jeopardising the richness of an artist's style. Therefore, using the simple greedy approach promotes temporal coherence over artistic style. The

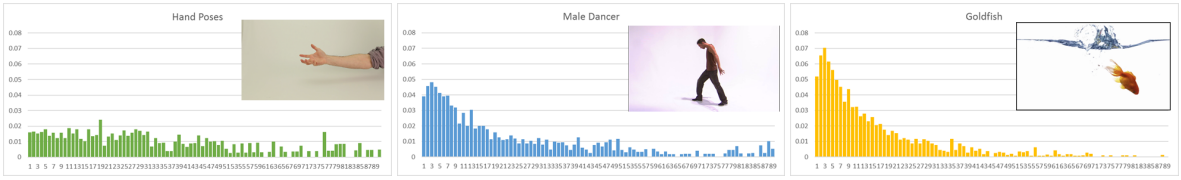


Figure 8: *Examples of sheet length histograms on three different movies (histograms are clamped at 90 frames). From left to right: as tracking is more difficult, the percent of short duration sheets grows.*

naive approach of rendering each frame separately, can be interpreted as the other extreme – promoting artistic style over temporal coherence.

To create results between these two extremes, we impose a restriction on the duration of the sheets. We use a threshold θ and use it as a soft limit for the duration of sheets. During the graph traversal stage of our algorithm, we do not extend a sheet if its current duration is longer than $\theta \pm r$, where r is some randomization factor per query (usually $0 \leq r \leq 0.15\theta$). This constraint maintains spatially longer curves, that are replaced by more characteristic strokes of the artist (see Figures 11 and 12 and discussion in next section).

6. Evaluation

This section focuses on the curve tracking part of our algorithm, before presenting our final results. We first evaluate the robustness of our curve tracking with respect to the speed of curve movement and the complexity of the scene. More complex scenes will include a larger number of curves as well as greater and more erratic movements. Next, we present the effect of the θ parameter that governs the tradeoff between temporal coherency and style preservation. Lastly, we compare our edge-tracking approach to region-tracking based on video segmentation.

Tracking Robustness. We conduct several experiments to evaluate and demonstrate our curve tracking algorithm. We use several examples demonstrating different conditions using different parameter settings and apply a statistical measure to examine the characteristics of the algorithm (all movie examples can be found in the supplemental material). Ground truth data is unavailable because matching two given sets of curves is an ill posed problem: curves could be matched in many ways, especially if there are topological differences between the sets including birth, death and splitting of curves.

The statistical measure we chose to examine is the histogram of curves belonging to sheets in a given duration in a video. Histograms with larger portions of long-duration sheets means that more curves are tracked over time and better temporal coherency could be achieved. Histograms with larger portion of short-duration sheets implies high birth/death rate of sheets, and therefore less success in temporal tracking. Figure 8 demonstrates this trend on three of our example videos (see Table 1 and accompanying video): a hand poses video that does not contain large movements and curves can be tracked over longer periods creating a rather uniform histogram of sheet durations, while a fish swimming video contains a large number of short period curves that split and merge at the water's edge, creating a histogram of many short duration and very few long duration sheets. A video of a male dancer lies between these two extremes.

Movement Speed and Complexity. There are two key factors that affect the robustness of our (and any other) tracking algorithm: the speed of movement and the complexity of the movement. To factor out complexity and illustrate how speed can affect the results, we create an artificial movie example where the bunny model is rendered moving from left to right in the frame. We use three speeds of movement and the results are shown in Figure 9. At slow speed (M1) the curves are tracked more successfully and long sheets are created. As speed increases (triple the speed at M3 and six times the speed at M6), fewer curves are tracked successfully and only short sheets are created.

The effect of unmatched curves can be diminished in our algorithm using the **MaxDist** parameter. By expanding the search range for matching points, more distant curves are taken into account while solving the global optimization. This strategy achieves better tracking at the cost of longer computation times. In Figure 9, when increasing **MaxDist** to 25% of the diagonal of a frame (instead of 5%), longer sheets are created again even when using the bunny movie at six times the original speed.

Complexity of movement is more challenging to evaluate. We have used three examples of more complex videos to demonstrate our algorithm. The first movie contains a scene with many edges where the background is moving because of camera movement, and the foreground is moving in a different manner (a man walking). Figure 10 (top) shows the histogram of this movie using three different frame rates (which is equivalent to speeds of movement). As can be seen, speed is a factor, but the complexity of movement is more of a challenge. By examining the tracking results one can see that the background smooth movement contains most of the longer sheets, while the foreground complex walk contains most of the short sheets. A similar trend is shown in the city example of Figure 10 (middle) where movements of different objects are combined in one scene. The histogram

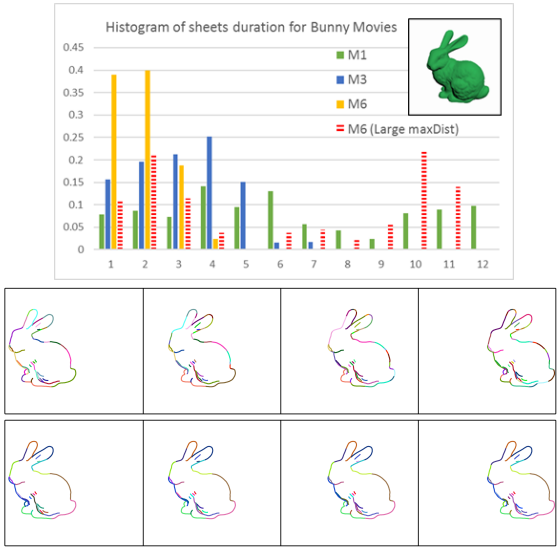


Figure 9: *The effect of speed on tracking: faster moving objects are harder to track and fewer long sheets are created (compare M1 to M3 or M6). When increasing* ***MaxDist****, tracking is better at the cost of longer computation times. Compare the curve matching colors of four frames between M6 (top row) and M6 using large* ***MaxDist*** *(bottom row).*

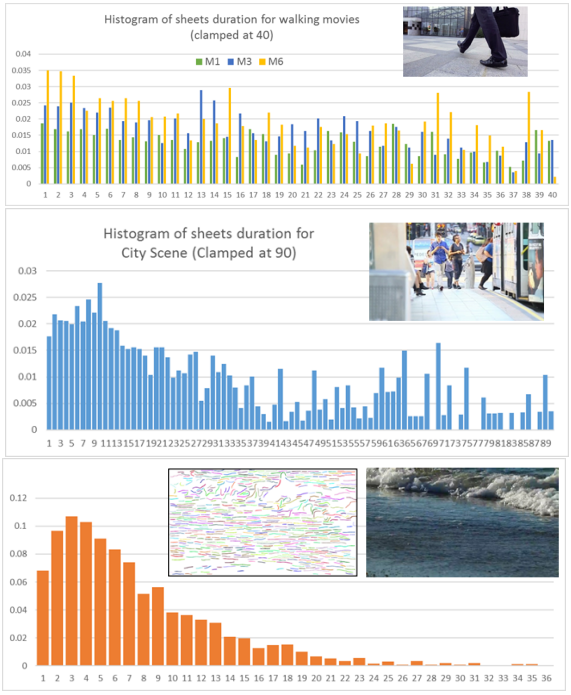


Figure 10: *Examples for complex movement tracking: camera movement along with object movement (top), different object movements (middle), and erratic movement of edges (bottom). Note that our algorithm succeeds in tracking the smoother frame movements - creating long sheets, but creates only short sheets for erratic movements.*

contains longer sheets for static edges (window frames) or smoother movements (tram), and short sheets for more complex movements (walking pedestrians). Lastly, in Figure 10 (bottom) we use an extreme example of a water waves movie where the extracted edges are all erratic and very challenging to track. As can be seen in this case, most curves are tracked for only a few frames and almost no sheets longer than 30 frames are created.

The θ Parameter. As described earlier, our algorithm tracking depends on the parameter θ that governs the tradeoff between temporal coherency and style. Figures 11 and 12 illustrate the effect of the θ parameter on the distribution of curve length and sheet duration on the male dancer example. When $\theta = \infty$, there is no constraint on the duration of sheets, we get longer sheet duration, and the distribution of duration is wide (this setting is the default used in all previously described experiments). On the other hand, the curves' lengths tend to be shorter. When we constrain $\theta = 15$ or even $\theta = 5$, we get shorter duration sheets but longer curve lengths. Therefore, this parameter, θ, can be used to govern the tradeoff between style and temporal coherence (see examples in the accompanying video).

Edges vs. Boundaries. Video segmentation has been suggested as an alternative to edge detection, which tends to be more stable over time on videos (see Collomosse et al. [CRH05]). To illustrate why this is not the case, we compare our edge detection algorithm to boundaries of segments using a newer algorithm for video segmentation. Using the same input video, we use our algorithm to extract edges, and use Grundmann et al. [GKHE10] algorithm for video segmentation. We define each boundary between two segments found using Grundmann et al. as an edge. Let $B(i)$ be the set of pixels belonging to such boundaries in frame i, and $E(i)$ the set of pixels belonging to some edge using our algorithm (see Figure 13 for an example of one frame). We define $C(i)$ as the set of common pixels that belong to both $B(i)$ and $E(i)$. However, we do not simply use $C(i) = B(i) \bigcap E(i)$. Instead, a pixel p is defined as a common pixel if $p \in B(i)$ and there is a pixel $q \in E(i)$ in its k-neighborhood of pixels, or if $p \in E(i)$ and there is a pixel $q \in B(i)$ in its k-neighborhood of pixels. We used $k = 1, 2, 3$ in our experiments.

For $k = 1$, the average of the ratio $C(i) : B(i)$ is 0.50, and $C(i) : E(i)$ is 0.22. For $k = 2$, the averages are 0.67 and 0.29, respectively, and for $k = 3$, the averages are 0.78 and 0.33. These numbers mean that even when we allow a large tolerance, there are boundaries between segments that are not edges and there are many edges that are not boundaries. Boundaries of segmentation in $B(i)$ must be closed and therefore contain parts that are not at all edges such as the boundaries inside the hands and legs in Figure 13(d). On

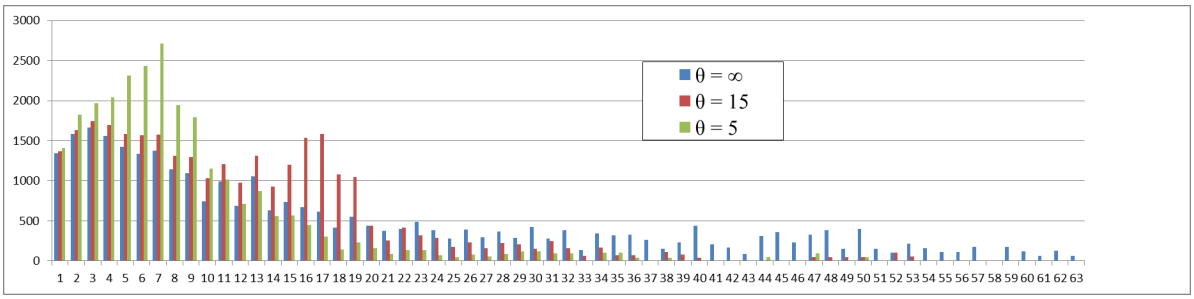


Figure 11: *The number of curves belonging to sheets of a given temporal duration. The larger* θ *is, the longer the average durations of a sheet is.*

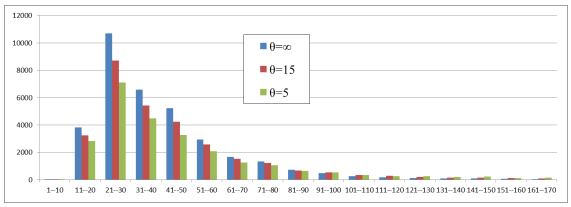
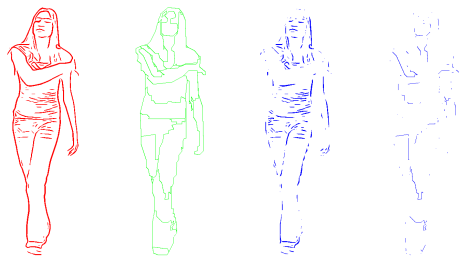


Figure 12: *The number of curves in a given spatial length (clamped at 170). As* θ *increases, more splits occur and spatially shorter sheets (and curves) are created.*



(a) (b) (c) (d)

Figure 13: *Comparing the edges extracted in a movie frame (a) to boundaries of region segmentation (b) clearly shows that there are edges that are not caught by the boundaries (c), as well as boundaries that are not at all edges (d).*

the other hand, the edges in $E(i)$ contain many curves that are not closed such as the folds of the shirt in Figure 13(c). Note that we specifically tried to create segments that are as large as possible. When using more accurate segmentation, many more boundaries between segments appear that are not edges. Therefore, using video segmentation may allow higher temporal coherency, but can destroy the style of line-drawing animation.

7. Results

We have tested our algorithm on six artists using the artists libraries from Berger et al. [BSM*13]. We also created four "artificial" styles by capturing a set of strokes in a prescribed manner (wiggly, curly, zigzag and straight) and building a strokes library from them. We chose several different input videos: two dance video sequences, two videos of animal movements: a fish and a butterfly, a hand wave video sequence, a moving train sequence and a computer rendered jump sequence. Table 1 summarizes some statistics on these sequences and provides timing results for the various stages of the algorithm. The primary factor governing the processing time is the number of edges in the sequence. The reason is that the pairwise matching optimization stage, which is the most costly stage in the algorithm, depends on the number of curves in the frames.

The best manner to view our results is to refer to the accompanying video. For all examples we use the same set of parameters $\beta = \textbf{MaxDist}$, $\alpha = \frac{1}{2}\beta$. We demonstrate a few examples of simple outputs of our algorithm, and some artistic stylization examples using our algorithm such as combining several artists output together, compositing the strokes with the original video, and overlaying the result on a static background. We also compare results with different settings of θ. A few examples of static frames are shown in Figure 14.

Recognition Study. To assesss our ability to mimic specific styles, and measure the effect of our θ parameter, we conducted an informal recognition experiment. We wanted to test if people can correctly match the style of an animation video to a static sketch. This task is a very difficult as the mediums are different and style can be very subtle. We created animations of the male dancer in five different styles of real artists in three different settings. First, we used $\theta = \infty$ for maximum coherence. Second, we used $\theta = 15$ to allow more style preservation (while reducing coherence). Third, we used the naive approach of rendering each frame separately to create no-coherence maximum style results. In total, we had 15 different movies: three types of movies for each of the five artists. Examples of the three types of movies can be seen in the supplemental materials.

We separated the experiment into three batches according to the different θ values. We ran the experiments on Mechanical Turk using 40 participants for each batch (participant ages range from 19 to 78, with a similar number of

Sequence	No. of frames	Res.	Avg. No. Curves	No. of Sheets	Avg. Duration	Edge time	Vect. time	Match time	Track. time	Rend. time	Total. time
Dance (F)	240	1080p	70	5283	5.2f	5m	33m	0:57h	5m	0:38h	2:13h
Dance (M)	273	1080p	50	3499	4.9f	6m	47m	1:39h	5m	1:44h	4:15h
Fish	401	720p	124	13238	5.2f	6m	103m	25:00h	7m	2:16h	29:06h
Butterfly	225	1080p	40	1979	4.6f	2m	29m	0:03h	3m	0:19h	0:54h
Hand	273	1080p	57	1341	18.9f	5m	38m	0:58h	7m	1:54h	3:37h
Train	254	1080p	161	16796	3.2f	7m	75m	27:00h	9m	1:47h	30:11h
Jump	600	1080p	25	2250	13.4f	4m	52m	1:04h	4m	0:22h	2:22h

Table 1: *Statistics and timing results of all our examples. From left to right: the number of frames in the sequence (we are using 24fps for all our examples), the resolution, the average number of curves per frame, the number of sheets of length > 1 created, and the average duration of each these sheet. Next, the breakdown of the running times of the algorithm: edge detection time (minutes), vectorization time (minutes), point matching optimization time (hours), tracking, or converting the graph time (minutes), average rendering time (hours), and total time to process the sequence.*

male and female participants). We presented the viewer with one movie at a time, and below displayed five face sketches from the five different artists. The order of movies of different artists was randomized as well as the order of sketches presented in each question. We asked the viewer to choose the one that resembles most the style shown in the movie. The recognition rates were 40.5% for the max-coherence study, 65.75% for the naive (no-coherence) study, and 61.4% for the more style preservation study. For all three studies, classification is well above the accuracy expected by chance, which is 20%. Moreover, these results support the conjecture that higher coherence can jeopardize style.

8. Discussion

The method we present converts video sequences to line-drawing animations, mimicking the specific style of an artist or some "artificial" style. Our goal was to retain the appeal of hand-drawn animation in a fully automatic method. As in other animation techniques, the key factor in this process is temporal coherence. Our algorithm uses a single parameter to control the tradeoff between coherence and style.

There are several limitations to our approach. First, not every video is suited to this type of stylization. Videos with motion blur where edges are not clear will challenge the edge detection stage of the algorithm (see Figure 2). Videos where differences between frames are too large, videos with a large amount of noise, or videos with too many edges will challenge the tracking stage. Our method is more suitable to clean, clear motion (although the stochastic movement like the water in our fish example also produces an interesting effect). Our approach first extracts edges from frames and then tries to match them to create sheets, a possible alternative would have been to devise a type of 3D edge detection, but such an approach remains a future challenge. Another serious limitation are the timing results. The largest factor in the running time is the point-matching algorithm, which was implemented in MatLab, and could be optimized to reduce its running time considerably.

Our sheet construction algorithm is greedy in a sense that we use only splits of nodes (curves). A more global solution may allow merging of nodes (and curves) when converting the graph to simple paths. In the rendering stage, we fit the strokes to the sheet curves using rigid transformation. We experimented with more advanced deformations for fitting, such as Affine transformations, but found that such transformations can blur the bitmaps of the strokes. Building a parametric model of the artist's strokes instead of using a direct data-driven approach, would allow more complex deformations to better fit the curves in the sheets. Another option is to try and define an "average" curve (see [HPR11]) from all curves in the sheet instead of fitting individual curves, but averaging tends to create curves that are too smooth.

Future Directions. The stroke libraries that we used originated from sketches of faces. We found the richness of strokes sufficient for general videos but possibly having a more general library of strokes could benefit the final results. Alternatively, one could imagine creating a specific stroke library for a given animation by drawing directly some of its frames. Although our goal was to produce a fully automatic method, one could imagine providing the artist with more control over the final results up to the level of replacing or modifying individual strokes. Our tracking method could also be used to create in-betweening of frames and in other applications such as tracking objects in videos.

Appendix A:

To see how to reduce our optimization problem from Eq. 2 to a linear program, notice that the objective function is simply a sum of terms that are the absolute value of linear combinations of the variables. For illustration purposes, a term can be of the form $c \cdot |w_{12} + w_{22} - w_{13} - w_{23}|$ for some $c \in \mathbb{R}$. Each of these terms can be replaced by a new variable in the objective and a new constraint. In our example, we can replace the above term by a new variable q and a new constraint that $c \cdot |w_{12} + w_{22} - w_{13} - w_{23}| \leq q$. Finally, any constraint of this form can be represented as a

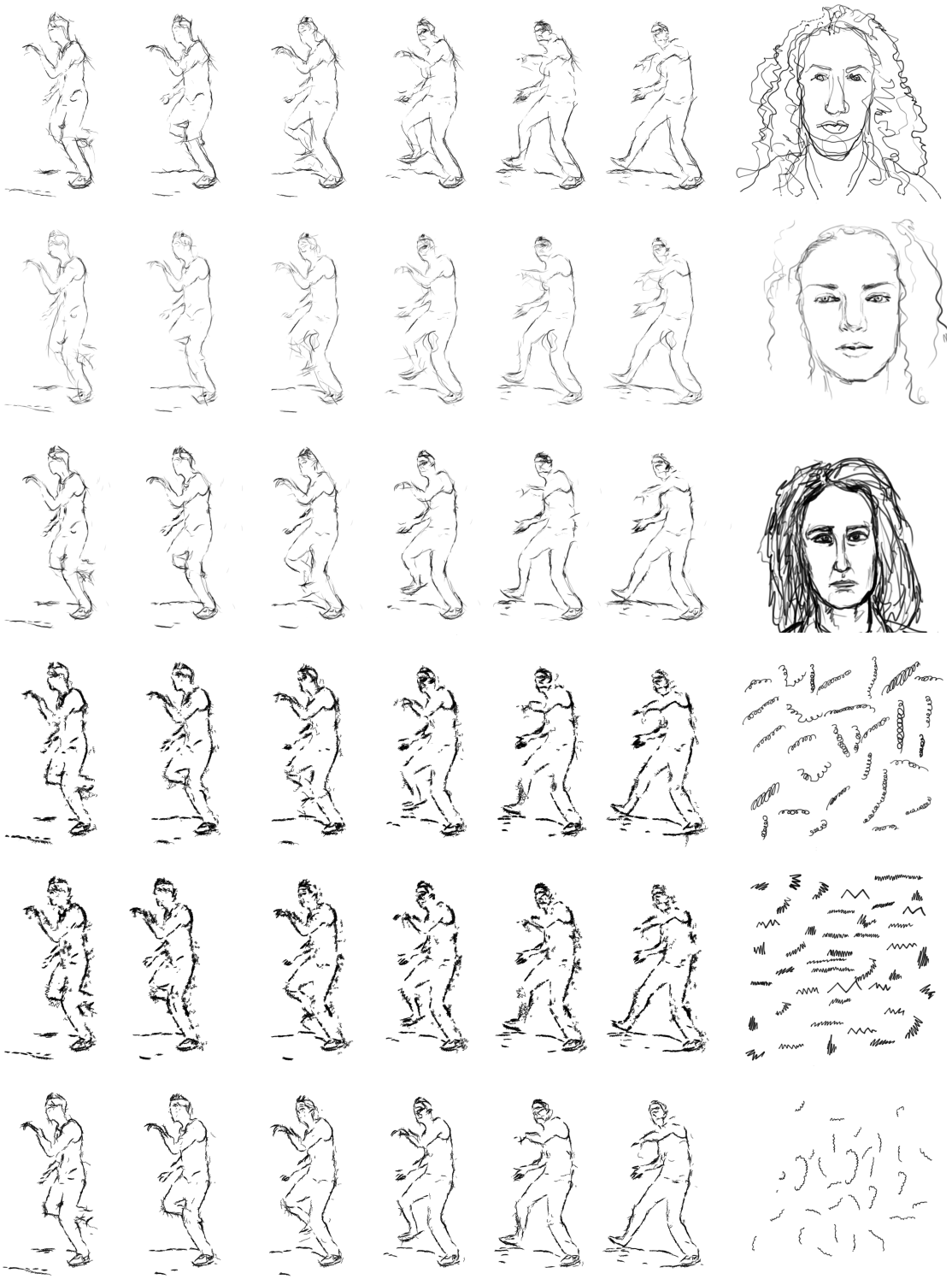


Figure 14: *Three examples of real artists' stylization of animation and three examples of "artificial" animation styles. On the right: the three faces are examples from [BSM*13] data-set (with permission), the three bottom stroke collections are examples from libraries created by choosing a specific manner to draw strokes. Please refer to supplemental material for the animations.*

set of linear inequalities. In our example, we can convert the new constraint to $c(w_{12}+w_{22}-w_{13}-w_{23}) \leq q$ and $-c(w_{12}+w_{22}-w_{13}-w_{23}) \leq q$. Because all the other constraints in Eq. 2 are already linear, this representation reduces the problem to a linear objective and a set of linear inequality constraints.

References

[Aga02] AGARWALA A.: Snaketoonz: A semi-automatic approach to creating cel animation from video. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering* (2002), NPAR '02. 3

[AHSS04] AGARWALA A., HERTZMANN A., SALESIN D. H., SEITZ S. M.: Keyframe-based tracking for rotoscoping and animation. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 584–591. 3

[AKJO07] AHUJA R. K., KUMAR A., JHA K. C., ORLIN J. B.: Exact and heuristic algorithms for the weapon-target assignment problem. *Operations Research 55*, 6 (2007), 1136–1146. 4

[BFP*11] BUCHHOLZ B., FARAJ N., PARIS S., EISEMANN E., BOUBEKEUR T.: Spatio-temporal analysis for parameterizing animated lines. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering* (2011), NPAR '11, pp. 85–92. 3

[BLC*12] BÉNARD P., LU J., COLE F., FINKELSTEIN A., THOLLOT J.: Active Strokes: Coherent line stylization for animated 3D models. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering* (2012), NPAR '12. 3

[BMP02] BELONGIE S., MALIK J., PUZICHA J.: Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 4 (2002), 509–522. 7

[BSM*13] BERGER I., SHAMIR A., MAHLER M., CARTER E., HODGINS J.: Style and abstraction in portrait sketching. *ACM Transactions on Graphics 32*, 4 (2013), Article No. 55. 2, 3, 10, 12

[BTC13] BÉNARD P., THOLLOT J., COLLOMOSSE J.: Temporally coherent video stylization. In *Image and Video-Based Artistic Stylisation*, Rosin P., Collomosse J., (Eds.), vol. 42 of *Computational Imaging and Vision*. Springer, 2013, pp. 257–284. 2

[CH05] COLLOMOSSE J., HALL P.: Video paintbox: The fine art of video painting. *Computers & Graphics 29*, 6 (2005), 862 – 870. 3

[CRH05] COLLOMOSSE J. P., ROWNTREE D., HALL P. M.: Stroke surfaces: temporally coherent artistic animations from video. *IEEE Transactions on Visualization and Computer Graphics 11*, 5 (2005), 540 – 549. 2, 9

[FTP03] FREEMAN W. T., TENENBAUM J. B., PASZTOR E. C.: Learning style translation for the lines of a drawing. *ACM Transactions on Graphics 22*, 1 (2003), 33–46. 3

[GA93] GREWAL M. S., ANDREWS A. P.: *Kalman Filtering: Theory and Practice*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993. 6

[GKHE10] GRUNDMANN M., KWATRA V., HAN M., ESSA I.: Efficient hierarchical graph-based video segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2010), pp. 2141–2148. 3, 9

[GS11] GAI J., STEVENSON R. L.: Robust contour tracking based on a coupling between geodesic active contours and conditional random fields. *Journal of Visual Communication and Image Representation 22*, 1 (2011), 33–47. 3

[HPR11] HAR-PELED S., RAICHEL B.: The frechet distance revisited and extended. In *Proceedings of the Twenty-seventh Annual Symposium on Computational Geometry* (2011), SoCG '11, pp. 448–457. 11

[KCWI13] KYPRIANIDIS J. E., COLLOMOSSE J., WANG T., ISENBERG T.: State of the "art": A taxonomy of artistic stylization techniques for images and video. *IEEE Transactions on Visualization and Computer Graphics 19*, 5 (2013), 866–885. 2

[KDMF03] KALNINS R. D., DAVIDSON P. L., MARKOSIAN L., FINKELSTEIN A.: Coherent stylized silhouettes. *ACM Transactions on Graphics 22*, 3 (July 2003). 3

[KLC07] KANG H., LEE S., CHUI C.: Coherent line drawing. In *Proceedings of the 5th International Symposium on Non-photorealistic Animation and Rendering* (2007), ACM, pp. 43–50. 4

[KNBH12] KALOGERAKIS E., NOWROUZEZAHRAI D., BRESLAV S., HERTZMANN A.: Learning hatching for pen-and-ink illustration of surfaces. *ACM Transactions on Graphics 31*, 1 (2012), 1:1–1:17. 3

[KWT88] KASS M., WITKIN A., TERZOPOULOS D.: Snakes: Active contour models. *International Journal of Computer Vision 1*, 4 (1988), 321–331. 3

[LYFD12] LU J., YU F., FINKELSTEIN A., DIVERDI S.: Helpinghand: Example-based stroke stylization. *ACM Transactions on Graphics 31*, 4 (2012), 46:1–46:10. 3

[NHS*13] NORIS G., HORNUNG A., SUMNER R. W., SIMMONS M., GROSS M.: Topology-driven vectorization of clean line drawings. *ACM Transaction on Graphics 32*, 1 (2013), 4:1–4:11. 4

[NSC*11] NORIS G., SÝKORA D., COROS S., WHITED B., SIMMONS M., HORNUNG A., GROSS M., SUMNER R.: Temporal noise control for sketchy animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering* (2011), NPAR '11, pp. 93–98. 3

[OH12] O'DONOVAN P., HERTZMANN A.: Anipaint: Interactive painterly animation from video. *IEEE Transactions on Visualization and Computer Graphics 18*, 3 (2012), 475–487. 2

[PM08] PAPADAKIS N., MÉMIN E.: A variational technique for time consistent tracking of curves and motion. *Journal of Mathematical Imaging and Vision 31*, 1 (2008), 81–103. 3

[WCH*11] WANG T., COLLOMOSSE J., HU R., SLATTER D., GREIG D., CHEATLE P.: Stylized ambient displays of digital media collections. *Compuers and Graphics 35*, 1 (2011), 54Ű–66. 2

[WNS*10] WHITED B., NORIS G., SIMMONS M., SUMNER R., GROSS M., ROSSIGNAC J.: Betweenit: An interactive tool for tight inbetweening. *Computer Graphics Forum (Proceedings of Eurographics conference issue) 29*, 2 (2010), 605–614. 3

[WOG06] WINNEMÖLLER H., OLSEN S. C., GOOCH B.: Real-time video abstraction. *ACM Transactions on Graphics 25*, 3 (July 2006), 1221–1226. 2

[WXSC04] WANG J., XU Y., SHUM H.-Y., COHEN M. F.: Video tooning. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 574–583. 2

[XSQ13] XU X., SEAH H. S., QUAH C. K.: Animated 3d line drawings with temporal coherence. *Computer Graphics Forum 32*, 7 (2013), 285–294. 3