Keyframe-Based Tracking for Rotoscoping and Animation

Aseem Agarwala[1] Aaron Hertzmann[2] David H. Salesin[1,3] Steven M. Seitz[1]

[1]University of Washington [2]University of Toronto [3]Microsoft Research

Abstract

We describe a new approach to rotoscoping — the process of tracking contours in a video sequence — that combines computer vision with user interaction. In order to track contours in video, the user specifies curves in two or more frames; these curves are used as keyframes by a computer-vision-based tracking algorithm. The user may interactively refine the curves and then restart the tracking algorithm. Combining computer vision with user interaction allows our system to track any sequence with significantly less effort than interpolation-based systems — and with better reliability than "pure" computer vision systems. Our tracking algorithm is cast as a spacetime optimization problem that solves for time-varying curve shapes based on an input video sequence and user-specified constraints. We demonstrate our system with several rotoscoped examples. Additionally, we show how these rotoscoped contours can be used to help create cartoon animation by attaching user-drawn strokes to the tracked contours.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques; I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Tracking

Keywords: rotoscoping, tracking, video editing, user-guided optimization, non-photorealistic rendering

1 Introduction

Rotoscoping, the process of manually tracing shapes through a captured image sequence, has become a central and critical part of creating computer-generated imagery (CGI). Nearly every modern film with special effects involves copious rotoscoping, often consuming up to twenty percent of the human time required for a CGI project [Goldman 2003]. Rotoscoping is used in multiple ways. Frequently, it is used to create mattes to place an actor into a different scene; conversely, it can be used to replace a real prop with a CGI element. Rotoscoped mattes can be used to apply image filters selectively over parts of a video frame. Rotoscoping can also be used to create 2D animation from captured video, as in the recent film, "Waking Life" [Linklater 2001]; indeed, rotoscoping was originally invented for just that purpose [Fleischer 1917].

Rotoscoping is still largely a manual process performed one frame at a time. The state of the art in CGI production uses simple keyframing: "roto-curves," or splines that bound the desired shape, are drawn by the animator at certain key frames in the animated sequence, and linear interpolation is used to generate roto-curves for the frames in between. Whenever the intermediate roto-curves appear too far from the shapes they are meant to trace, the animator adjusts certain control points by hand. These "pulled" points become new constraints, and the edit is propagated automatically to earlier and later frames. Unfortunately, simple linear interpolation fails to track any kind of interesting, complex motion very well, and so in practice a great deal of tedious, manual adjustment is required.

In contrast, the computer vision research community has developed a large body of work for the automated tracking of contours. Ideally, one would expect that such work would have utility in alleviating much of the tedium of state-of-the-art rotoscoping. Unfortunately, very little of this work is directly applicable. One reason for this is that virtually all previous work in tracking is purely "feed forward" — that is, a set of points, contours, or other features is initialized in one frame and automatically tracked forward in time. Such an approach is problematic in that, when the tracking goes wrong, it is not clear in which frame *exactly* the tracked curves went sufficiently astray to require correcting (Figure 1). Furthermore, edits propagate only forward and never backward, requiring a huge number of manual interventions. Fundamentally, the problem with applying traditional tracking techniques for rotoscoping is that they are not designed to make full use of user intervention. A final problem is consistency [Stewart 2003]: roto-curves that are created by stepping forward and editing frame-by-frame tend to "bubble" or "chatter" around the desired edge, while roto-curves that are interpolated from keyframes tend to move much more smoothly.

In this paper, we show how tracking can be reformulated as part of a user-driven keyframe system. This reformulation recasts tracking as a spacetime optimization that computes shape and motion simultaneously. Our approach combines the best features of user guidance and automated tracking: the rotoscoper can specify constraints by manipulating any roto-curve control point at any frame in the sequence; a spacetime optimization, computed using a standard nonlinear optimization technique, then finds the best interpolation of the roto-curves over time. The user can iterate by refining the results and restarting the optimization. Thus, the user can guide the automatic tracking in situations that are simply beyond the capabilities of state-of-the-art tracking, while the optimization can significantly reduce the amount of human effort involved.

In addition to rotoscoping, we show how a variant of the same formulation can be applied to the creation of 2D animation, based on the roto-curves. In our animation system, the animator begins with roto-curves that serve as a scaffolding for the animated brush strokes. The brush strokes, painted by the animator, are automatically associated with the roto-curves. The animator is free to make an edit to any brush stroke in any frame, and the edits are propagated forward and backward in time.

We demonstrate our approach with a number of rotoscoped examples, and we show how the resultant roto-curves can be used for a variety of animated effects.

1.1 Related work

Our work is a synthesis of visual tracking techniques and keyframe animation. As such, it is related to a wide body of work in computer

http://grail.cs.washington.edu/projects/rotoscoping/

Figure 1 *A comparison to forward tracking.* Seven frames of a twenty-five frame sequence. ***First row:*** forward tracking, in an approach similar to snakes (we use the same energy function described in Section 2.3 to perform forward tracking across two frames). Slight problems appear at frame 23, grow larger at 24, and become disastrous by 40. ***Second row:*** interactive editing to fix the problem. In a forward-tracking framework it is unclear at which frame to begin editing the curve; the error is perhaps too small to be concerned with in frame 23, so the user makes a correction at frame 24 and continues to track forward. This fixes the next few frames, but does not prevent the problem from eventually reappearing. ***Third row:*** keyframe-based tracking. The user edits the curve in frame 40, making it a keyframe, and the system simultaneously optimizes the curve shape in frames 16-39. Even though the same energy functions are used to track the contours, spacetime optimization outperforms the feed-forward approach.

vision, graphics, and image processing.

While there is a very large literature on tracking in the computer vision community, most closely related to our work is automated contour tracking (see Blake and Isard [1998] for an overview). We are particularly inspired by *snakes*, in which Kass et al. [1987] defined tracking and contour fitting as an energy-minimization problem where a manually initialized curve is automatically refined to fit edges in an image, using a combination of image gradient and shape regularization terms. The original snakes work incorporated user hints that help guide a contour to a better solution in a single frame. However, these hints are not propagated to the rest of the sequence, other than by providing a better initialization for the next frame. We also make use of ideas from patch tracking, in particular the classic work by Lucas and Kanade [1981]. However, in contrast to previous work on tracking, which is purely "feed forward," we optimize over all frames simultaneously to obtain a global solution that incorporates user constraints at any point in time.

More recently, a few authors have described methods for simultaneous motion estimation over an entire sequence, either to compute optical flow [Irani 2002] or for point-tracking [Torresani and Bregler 2002]. These methods are designed to work automatically, and make a number of important restrictions on the input. We extend this approach significantly by incorporating user interaction and applying it to contours.

While modern tracking techniques are increasingly robust and can track multiple hypotheses [Blake and Isard 1998], they are still limited in the range of motions that can be reliably tracked — certainly no tracking method works all of the time. In contrast, commercial rotoscoping tools like Pinnacle Commotion allow a user to track any sequence by intensive manual annotation; each contour is hand-positioned every few frames. While some video effects tools such as Apple's Shake and Adobe After Effects include basic motion tracking, they are generally limited to the forward tracking of individual control points. This is inadequate for rotoscoping, since information provided by the shape of the curve is ignored, and because the benefits of keyframing are lost.

A related problem to rotoscoping is *matte extraction*, i.e., separating foreground from background. Insofar as matte extraction yields silhouettes, it may be considered a form of rotoscoping. Blue-screen matting [Smith and Blinn 1996] is a common approach that requires the video to be shot with special backgrounds. Autokey [Mitsunaga et al. 1995] offers human-assisted rotoscoping and alpha matte extraction; however, their interface is designed around forward tracking (they suggest a keyframing approach as future work). Video matting [Chuang et al. 2002] shows how complex mattes can be generated, assuming successful motion estimation; they use a publicly-available optical flow algorithm [Black and Anandan 1996], which frequently fails for moderately difficult motions. One potential application of our system is generating robust trimaps for alpha matting of complex scenes.

Agarwala [2002] and Hoch and Litwinowicz [1996] both present rotoscoping systems that use contour tracking to generate animation from video; however, both use forward tracking. In contrast, Hall et al. [1997] describe a keyframed rotoscoping system that they call "active surfaces"; however, they require the user to select edge features every third frame, and show limited results on a single, fifteen-frame sequence. In a similar system, Luo and Eleftheriadis [1999] use forward and backward tracking of contours and then splice the two tracks together.

We take inspiration from systems that exploit vision techniques to assist interactive image editing; an overview of this area can be found in Mortenson [1999]. One common application is interactive segmentation of objects from an image, such as "intelligent scissors" [Mortensen and Barrett 1995]. Our work can be seen as extending intelligent scissors to video sequences.

Spacetime constraint systems for animation [Witkin and Kass 1988; Cohen 1992] allow a user to specify constraints and objectives over the length of an animation, and use optimization to generate motion paths. Though the domain of our problem is very different, we also use optimization over space and time to calculate motion according to user-specified constraints.

Our animation system builds on previous methods for creating animation with strokes. Painterly animation systems create animation from video sequences or 3D models [Litwinowicz 1997; Meier 1996], but do not allow users direct control over the motion of individual strokes. Our animation system is most directly inspired by the WYSIWYG NPR system of Kalnins et al. [2002] in which a user draws strokes over 3D models, which can then be propagated

to new frames.

Our application to animation is also related to keyframe animation tools using curves. Burtnyk and Wein's classic system [1976] allows an animator to attach drawings to polygonal skeletons with manual correspondences and warping. Hsu and Lee [1994] extend this system by automating and improving the polygonalization. Researchers have also addressed the challenging problem of automatically creating in-betweens of hand-drawn keyframes [Kort 2002].

2 Interactive tracking

We now describe our keyframe-based rotoscoping system for tracking the motion over time of a set of curves in an image sequence. Applications of this system are described in Sections 3 and 4. We first depict the interactive system from a user's point of view, in which a user specifies roto-curves to track, and their shapes in two or more keyframes. We then define an objective function for determining the motion of the curves over time. This objective function includes both temporal smoothness terms and image terms; hence, the problem is a spacetime optimization problem, constrained by the user-specified keyframes. We then give an optimization procedure that solves for the best curve sequence, optimizing over the whole image sequence at once. Once the optimization has completed, the user may refine roto-curves in any frame, and then rerun the optimization with these new constraints.

2.1 Interaction

A typical interaction with the system to track a set of curves is as follows:

1. ***Draw first keyframe***. The user begins by selecting a frame t_a in the video sequence, and drawing a set of curves in this frame. These curves specify the first keyframe. Curves are specified by placing control points of a piecewise cubic Bézier curve with C^0 continuity, to allow for sharp features.[1] The curves usually correspond to image features and contours, although they may be placed anywhere. These curves can be attached at endpoints to form *joints*, allowing the user to specify that certain curves should stay attached to each other throughout the sequence.

2. ***Draw second keyframe***. The user then selects a later frame t_b, and the system copies the curves from t_a to t_b. The user adjusts these curves to place them in their desired positions using a simple curve editing interface. Our interface supports curve rotation, translation, and scaling, and allows pulling on individual control points. The spacing of keyframes depends on the complexity of motion. In the examples in this paper, we usually placed keyframes every fifty frames.

3. ***Optimization***. The system then solves for the positions and shapes of these curves in the intermediate frames, as described in Sections 2.3 and 2.4.

4. ***Refinement***. Because of the difficulty of the tracking problem, there will often be unsatisfactory aspects of the estimated motion. The user may edit the resulting sequence to fix any errors produced by the optimization. To refine the rotoscoping result, the user visits selected in-between frames and pulls on control points. The user can then rerun the optimization. Every point that the user has ever pulled during editing becomes a hard constraint for the optimization.

Typically, for curves whose shape across time is unsatisfactory, the user pulls on a single control point whose position is furthest from the desired position, and then restarts the optimization. This human guidance is often enough to achieve the desired solution; during the optimization, the information propagates to neighboring points in the same frame and other frames. If not, the user may pull on additional control points, and iterate. As we show in our results, the user typically has to pull very few points, many fewer than would be required with current linear-interpolation-based rotoscoping tools.

This approach makes use of the best features of user guidance and automatic tracking: the user specifies a few constraints that are difficult to determine automatically, and automatic tracking fills in the rest, thus automating most of the work.

[1] Our system also allows curves to be drawn using "intelligent scissors" [Mortensen and Barrett 1995]. However, we have found the control point interface more useful, since the user is best able to choose a parameterization that is detailed enough for the purposes of the user, but has as few control points as possible.

2.2 Parameterization

Our goal is to optimize a set of roto-curves. Each roto-curve is parameterized by a set of control points; these are the unknowns in the optimization. Specifically, we write each curve as $\mathbf{c}_t(s)$, where t is an index over frames and s is the spatial parameterization of the curve in frame t. The user-drawn keyframe roto-curves are in frames t_a and t_b; thus the optimization variables are the positions of the control points for all frames $t_a < t < t_b$.

The mapping from control points to variables must be done carefully to take any constraints into account, since we use an unconstrained nonlinear optimization method. Control points of curves that are attached at a joint must map to the same variable (although we define the energy terms in Section 2.3 with respect to a single curve, we typically optimize multiple curves, attached at joints, simultaneously). Also, when the user pulls a control point, it is removed from the set of variables since its location becomes fixed.

The energy terms of the optimization are defined with respect to points sampled roughly one pixel apart along the keyframe curve in frame t_a. These sample positions are denoted in the parametric domain as $(s_1, s_2, ...s_N)$. Because of the way that the keyframe curves are created, we assume that they are in one-to-one parameteric correspondence, and use the same sample points in the remaining frames.

2.3 Energy function

We now define the objective function that is used to guide the tracking problem. The objective function consists of two types of energy terms, *image terms* and *shape terms*. The image terms prefer curves that closely follow the motion of image features or contours. The shape terms penalize quickly moving and deforming curves. These different objective functions are complementary — the image terms can "lock onto" deforming regions where pure shape interpolation would fail, and the shape terms can produce reasonable interpolations for slowly moving objects where image terms alone would fail (e.g. due to occlusions or clutter).

Our objective function is a linear combination of five weighted energy terms:

$$E = w_V E_V + w_L E_L + w_C E_C + w_I E_I + w_G E_G \qquad (1)$$

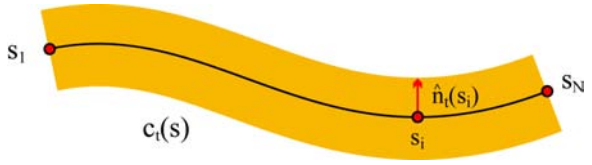

Figure 2 A thin ribbon around a curve $\mathbf{c}_t(s)$ forms the window of image data that we track. This window is parameterized by coordinates s,k where discrete samples s_i run along the curve and k marches off the curve in the direction of the unit normal $\hat{\mathbf{n}}_t(s_i)$.

The relative weights $w_V = 0.1$, $w_L = 500$, $w_C = 40,000$, $w_I = 1$, $w_G = 1,000$ have been determined experimentally, and remain fixed within the system; they do not need to be adjusted by the user, since, in our experience, the same set of weights works well for all of the sequences we have tried. The individual energy terms are described in the next two sections.

2.3.1 Shape terms

Our first two shape terms are based on the shape interpolation terms proposed by Sederberg et al. [1993]. The edge length term E_L penalizes the change over time of the length of the vector between adjacent samples of a curve:

$$E_L = \sum_{i,t} \left(\|\mathbf{c}_t(s_{i+1}) - \mathbf{c}_t(s_i)\|^2 - \|\mathbf{c}_{t+1}(s_{i+1}) - \mathbf{c}_{t+1}(s_i)\|^2 \right)^2 \quad (2)$$

The second shape term E_C measures the change in the second derivative of the curve over time, as an approximation to measuring the change in curvature:

$$\begin{aligned} E_C \quad = \quad & \sum_{i,t} \|(\mathbf{c}_t(s_i) - 2\mathbf{c}_t(s_{i+1}) + \mathbf{c}_t(s_{i+2})) - \\ & (\mathbf{c}_{t+1}(s_i) - 2\mathbf{c}_{t+1}(s_{i+1}) + \mathbf{c}_{t+1}(s_{i+2}))\|^2 \end{aligned} \quad (3)$$

These two terms alone constrain the shape of the curve but not its position or velocity. The following term is added to penalize fast motion:

$$E_V = \sum_{i,t} \|\mathbf{c}_t(s_i) - \mathbf{c}_{t+1}(s_i)\|^2 \quad (4)$$

2.3.2 Image terms

Most tracking algorithms assume that appearance changes slowly over time, and in particular that a small window of image data around a feature being tracked remains roughly constant in consecutive frames. We thus compare image data along corresponding curves in consecutive frames. Our tracking term E_I is an extension of Lucas-Kanade [1981] tracking; they compare square windows of image data between two frames, and optimize over the location of the window in the second frame to minimize the difference in the image data. We are interested in image data within a thin ribbon around each curve we are tracking (Figure 2). If a curve being tracked separates a foreground character from a background, the image constancy assumption only applies to the foreground; to account for this, we allow the user to indicate that only one side of a curve should be tracked.

The image term samples points along directions normal to the curve in frame t, and compares them with corresponding points in the next frame $t+1$, for all frames. Specifically, let $\hat{\mathbf{n}}_t(s_i)$ be a unit normal to the curve at point s_i at time t, that is created by rotating the tangent vector $\frac{d\mathbf{c}_t(s_i)}{ds_i}$ (computed analytically) by 90 degrees and then normalizing by its length. Then, the image term is:

$$E_I = \sum_{i,k,t} \|I_t(\mathbf{c}_t(s_i) + k\hat{\mathbf{n}}_t(s_i)) - I_{t+1}(\mathbf{c}_{t+1}(s_i) + k\hat{\mathbf{n}}_{t+1}(s_i))\|^2 \quad (5)$$

where $I_t(\mathbf{p})$ is the RGB color vector of image I_t at point $\mathbf{p}$, and k varies over a user-specified window (for the results presented in this paper, k varies from -5 to 5 for two-sided tracking, and 0 to ± 5 for one-sided tracking).

Many roto-curves lie on image edges; in this case, edges can be used as additional information to guide the curves. In addition, when curves separate foreground and background, the user generally selects only the foreground side of the curve to apply the image constancy term to. However, this makes it very easy for such a curve to drift toward the interior of the foreground shape. In this case, edge cues can rescue the track.

Our edge term E_G measures the magnitude of the image gradient at points along the curve. We write the gradient magnitude (the sum of the magnitudes of the image gradients in the three color channels) as $G(\mathbf{p})$ for an image point $\mathbf{p}$. Since we are minimizing energy terms, we subtract the gradient from the maximum possible gradient, K, and minimize this quantity $G'(\mathbf{p}) = K - G(\mathbf{p})$. Also, it is preferable to consider a curve sample's edge strength relative to how strong an edge that sample lies on in the keyframes; i.e., if the sample does not lie on strong edges in the keyframes, E_G at that sample should be weighted lightly. Thus, E_G is normalized at point s_i by the minimum, denoted M_i, of the gradients at the two keyframes:

$$E_G = \sum_{i,t} \left(\frac{G'(\mathbf{c}_t(s_i))}{M_i} \right)^2. \quad (6)$$

where $M_i = \min(G'(\mathbf{c}_{t_a}(s_i)), G'(\mathbf{c}_{t_b}(s_i)))$.

2.4 Optimization Method

We now give the method used to minimize the energy function E (Equation 1). Our goal is to solve for the control points that minimize E over all time frames, subject to the user-specified constraints. Notice that E has the general form of a nonlinear least squares (NLLS) problem, i.e., it can be written as $E = \sum_k w_k \|\mathbf{f}_k(\mathbf{x})\|^2$, where $\mathbf{x}$ is the vector of unknowns.

NLLS optimization is well-studied, and Levenberg-Marquardt (LM) [Nocedal and Wright 1999] is a popular solution. The LM algorithm requires that we can compute the Jacobian of each $\mathbf{f}_k(\mathbf{x})$. To obtain this Jacobian, we first define the Jacobian of each energy term with respect to all point samples $\mathbf{c}_t(s_i)$ in all time frames: $(\mathbf{J}_k)_j = \frac{\partial \mathbf{f}_k(x)}{\partial \mathbf{c}_j}$, where j indexes over all samples. We then use a change-of-basis matrix $\mathbf{B}$ to convert from curve samples to spline control points; the Jacobian of $\mathbf{f}_k(\mathbf{x})$ is thus given by: $\mathbf{J}_k\mathbf{B}$.

The number of variables in our optimization can be quite large — two times the number of curve control points in a frame times the number of in-between frames (e.g., Figure 4). Fortunately, the Jacobian matrix is very sparse, and thus we use a trust-region variant of LM developed by Steihaug [1983] for large, sparse NLLS problems; see Nocedal & Wright [1999] for details.

Determining the Jacobian for most of the energy terms is straightforward. For image terms that involve $I_t(\mathbf{p})$, the derivative of the

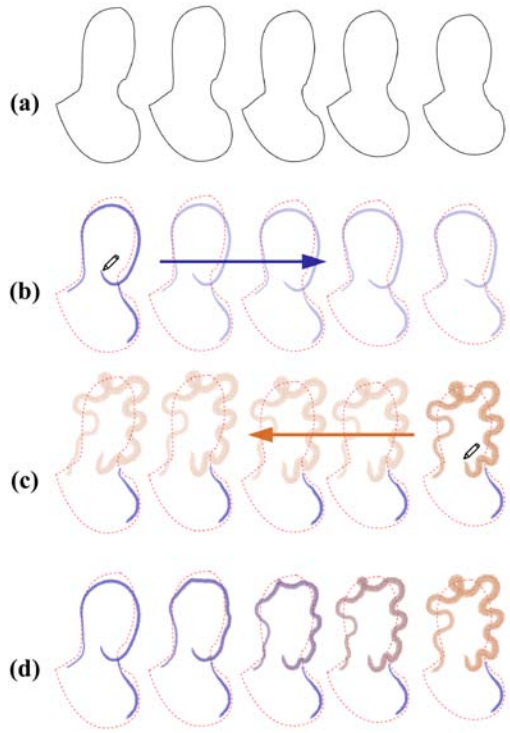

Figure 3 *Creating a rotoscoped animation.* The animator begins with a set of roto-curves (a), which serve as a scaffolding for the animated brush strokes. The animator then draws one or more brush strokes on an arbitrary frame of the sequence (b, leftmost image). The brush strokes are automatically associated with roto-curves, but the animator need not be aware of these associations. By following the roto-curves, these associations allow new strokes to be generated for all other frames in the sequence automatically, which the animator can view (b). The animator is free to make an edit to any brush stroke in any frame (c, rightmost image). These edits are propagated backward (c). Strokes of in-between frames are defined as linearly blended versions of the two propagated strokes (b,c) according to the distance from the drawn frames (d).

image is computed by convolution with a derivative-of-Gaussian filter, similar to the Lucas-Kanade method [1981]. The derivative of $G(\mathbf{p})$ is also computed this way. In order to compute the derivative of the unit curve normal $\hat{\mathbf{n}}_t(s_i)$, we need to take the normalization by its length (which makes it a unit vector) into account. We approximate this derivative by assuming that this normalization factor is constant; this approximation behaves reasonably well, since the edge length term E_L causes length to vary slowly from frame-to-frame. Thus, the derivative of the unit normal is approximated as the derivative of the unnormalized normal, $\mathbf{n}_t(s_i)$, which is simply a linear combination of four control points.

In addition, we apply the optimization in a coarse-to-fine fashion [Bergen et al. 1992] using four levels of a Gaussian pyramid, in order to avoid local minima. We use the hierarchical basis preconditioner of Szeliski [1990] to avoid problems from ill-conditioning.

3 Rotoscoped animation

One application of rotoscoping is to create cartoon-style animation from video. In this section, we describe the key aspects of our animation system, which makes use of the roto-curves.

Though it is possible to directly use the roto-curves themselves as animation primitives [Agarwala 2002], it is often preferable to modify them for animation, since good animation departs from realism in numerous ways.

In the rest of this section, we will refer to the animation curves as *strokes* to differentiate them from the roto-curves upon which they are based.

3.1 Interaction overview

We begin by describing a typical interaction to draw an animated character. Prior to drawing the animation, the animator creates a set of roto-curves in the scene that will serve both as visual reference and as a scaffolding for the animated brush strokes. Once the scene is rotoscoped, the video image itself becomes optional and can be kept as visual reference, or faded to show just the roto-curves. The animator can then begin to draw; strokes are propagated to other frames as described in Figure 3.

3.2 Technical approach

There are two key steps required to propagate drawn strokes to other frames so that they follow the motion of the roto-curves. First, we calculate a correspondence between each stroke and the roto-curves in a single frame. Then, we copy strokes to other frames and deform them so as to follow the motion of the corresponding roto-curves.

Calculating correspondences between curves is a classic problem that is typically solved using dynamic programming [Sederberg and Greenwood 1992; Geiger et al. 1995]; we extend this approach in Section 3.2.1.

One approach to deforming strokes would be to interpolate the deformation of the roto-curves over image space using a scattered data interpolation technique such as Beier-Neely interpolation [Beier and Neely 1992] or thin-plate splines [Litwinowicz and Williams 1994]. However, this may give undesirable results, as a drawn stroke would be influenced by unrelated parts of roto-curves, rather than just the parts it is attached to. We thus explore an alternate approach using offset curves in Section 3.2.2.

3.2.1 Stroke–roto-curve correspondence

The first step of our animation system is the automatic correspondence of strokes to roto-curves. The problem is complicated by the fact that there is not necessarily even a one-to-one correspondence between strokes and roto-curves: a short stroke may cover just a small portion of a roto-curve, or a long one might travel the length of several roto-curves. Also, the direction of travel may be opposite to the parameterization of the roto-curve.

To handle all of these cases, we extend the dynamic programming algorithm of Geiger et al. [1995], which matches a pair of contours. In our extended version of the algorithm, a single brush stroke $\mathbf{b}$ is matched against *all possible* roto-curves $\mathbf{c}^1, \ldots, \mathbf{c}^k$, as follows.

We represent drawn strokes as polylines, and also sample each spline-based roto-curve into a polyline (correspondence algorithms are simpler for polylines, and there is no reason to convert drawn strokes to splines). We then begin by creating a 2D table with the pixel-spaced samples of $\mathbf{b}$ on the x-axis and the pixel-spaced samples of all curves $\mathbf{c}^1, \ldots, \mathbf{c}^k$ stacked up, one above the other, on the y-axis. Using dynamic programming, each (x, y) cell is filled with the score of the best possible overall mapping that maps point x to point y. This score is computed by finding the lowest-cost path back

to any row of the leftmost column of the table. To make the algorithm faster (at the expense of necessarily finding the globally best solution), when filling in each cell (x,y), we only consider candidate pairs whose sample points lie within 10 pixels of each other. When the table is full, the best correspondence is found by finding the best score in any row of the rightmost column of the table. The result is a mapping from the samples of $\mathbf{b}$ to the samples of one or a series of curves in $\mathbf{c}^1,\ldots,\mathbf{c}^k$.

We use the same function to evaluate the quality of the mapping as Geiger et al. [1995], with one additional term. Since strokes are typically drawn near the objects they should follow, we also minimize the Euclidean distance from each stroke sample to its corresponding roto-curve sample.

3.2.2 Generating strokes as offsets from roto-curves

Once we know which portion of a roto-curve or roto-curves a given stroke $\mathbf{b}$ corresponds to, we must propagate deformations of that stroke to frames forward and backward in time, according to deformations of the corresponding roto-curve(s). For ease of explanation, let $\mathbf{b}$ correspond to exactly one roto-curve $\mathbf{c}$. For each sample $\mathbf{b}(r_i)$ along $\mathbf{b}$ and its corresponding sample $\mathbf{c}(s_j)$ along $\mathbf{c}$ we calculate a local coordinate system for $\mathbf{c}(s_j)$. This coordinate system is defined by the unit tangent and normal to $\mathbf{c}$ at $\mathbf{c}(s_j)$. The coordinates u,v of $\mathbf{b}(r_i)$ within this coordinate system are calculated.

Finally, to generate the deformed stroke $\mathbf{b}'$ at some different frame, we iterate over each sample $\mathbf{b}(r_i)$ of $\mathbf{b}$. We find the corresponding point $\mathbf{c}(s_j)$ and its corresponding point $\mathbf{c}'(s_j)$ along $\mathbf{c}'$ in the other frame. The local coordinate system of $\mathbf{c}'(s_j)$ is calculated, and we use the same coordinates (u,v) in this new local coordinate frame to generate sample $\mathbf{b}'(r_i)$ of $\mathbf{b}'$.

3.2.3 Improving generated strokes with optimization

Generating strokes as offsets from roto-curves does not always produce desirable results; the strokes can occasionally lack temporal coherency. We would like to enforce a similar condition to the one we enforce in rotoscoping — namely, that our strokes should be a reasonable shape interpolation, and change shape slowly over time.

Thus, we use a similar optimization process to improve generated strokes as we do to improve roto-curves, with a few significant changes. For one, since strokes are represented as polylines, no change of basis is required. Also, the image constancy and edge terms are not relevant; instead, we add a new term E_O that encourages the new, optimized deformed stroke $\mathbf{b}''$ to be similar to the deformed stroke $\mathbf{b}'$ generated through offsets:

$$E_O = \sum_{i,t} \left\| \mathbf{b}_t''(r_i) - \mathbf{b}_t'(r_i) \right\|^2 \tag{7}$$

We would like to incorporate the smooth shape interpolation terms as well, namely terms E_L, E_C, and E_V (equations 2, 3,and 4). However, since the new term E_O and shape terms E_V and E_C are linear, we instead use a modified version of E_L that is also linear, since this allows the solution to be computed exactly and quickly in one solution of a linear system. This modified E_L, which we denote E_L', compares tangent vectors over time directly rather than their squared length. When rotoscoping, comparing edge lengths is useful to avoid shortening [Sederberg et al. 1993]. This phenomenon is prevented by the term in equation 7 since the deformed stroke $\mathbf{b}'$ is typically of the correct length, so comparing tangents directly yields good results. The final energy function that is optimized to determine the shape of drawn strokes over time is thus a weighted sum of E_O, E_L', E_C, and E_V.

Sequence	user-edited points	total points	ratio
Figure 5	483	4230	11.4%
Figure 6	338	5204	6.5%
Figure 7	494	8606	5.7%

Figure 4 *User-interaction required.* We measure the effort of the user to rotoscope each sequence by comparing the number of user-edited control points to the total number of points across the image sequence. User-edited points are either drawn at keyframes, or pulled by the user during interactive refinement. The final column is the ratio of the two numbers, i.e., the percentage of total control points that were user-edited. Control points attached at a joint are counted as a single control point.

3.2.4 Interpolating edited strokes

If a propagated stroke is later edited by the user in a different frame, the edited stroke becomes a new keyframe. Strokes in between two keyframes are treated a little differently than those propagated from one keyframe. Instead of being propagated in one direction only, they are propagated both forward and backward from their surrounding keyframes. The two propagated strokes are then linearly blended according to their relative distance from each keyframe (Figure 3).

4 Results

We demonstrate our results by rotoscoping three sequences (Figures 5, 6, and 7). Our results are best seen in video form, though we show several frames of the roto-curves here.

It is difficult to objectively measure the success of our rotoscoping system, and the amount of user-interaction required. One possible approach, shown in Figure 4, is to count the number of user-edited control points, compared to the total number of control points. Notice that this ratio slowly increases with the difficulty of the sequence; the waving hand in Figure 5 moves and deforms rapidly, and thus requires more user-interaction.

We also demonstrate that our system can be used for both special effects and animation. For the former, we create simple mattes by grouping roto-curves into regions and filling them. Then, we apply filters selectively to different regions of the video to create a variety of effects, i.e., applying makeup, changing the color of a shirt, or applying a stylistic filter. We used our system to quickly create the mattes, and then used Adobe After Effects to apply effects.

Finally, we demonstrate several animations created using our system and the same roto-curves as the previous examples. Each source, animation, and processed video clip can be seen in the accompanying video.

5 Conclusion

In this paper, we showed how tracking can be reformulated as part of a user-driven keyframe system. There are many opportunities to improve our system. One current limitation is how we handle curve samples that leave the boundaries of the image; we simply drop the image terms for these samples. This approach works fairly well, but can introduce discontinuities that quadratic optimization techniques are not well-suited to handle; thus, control points that lie

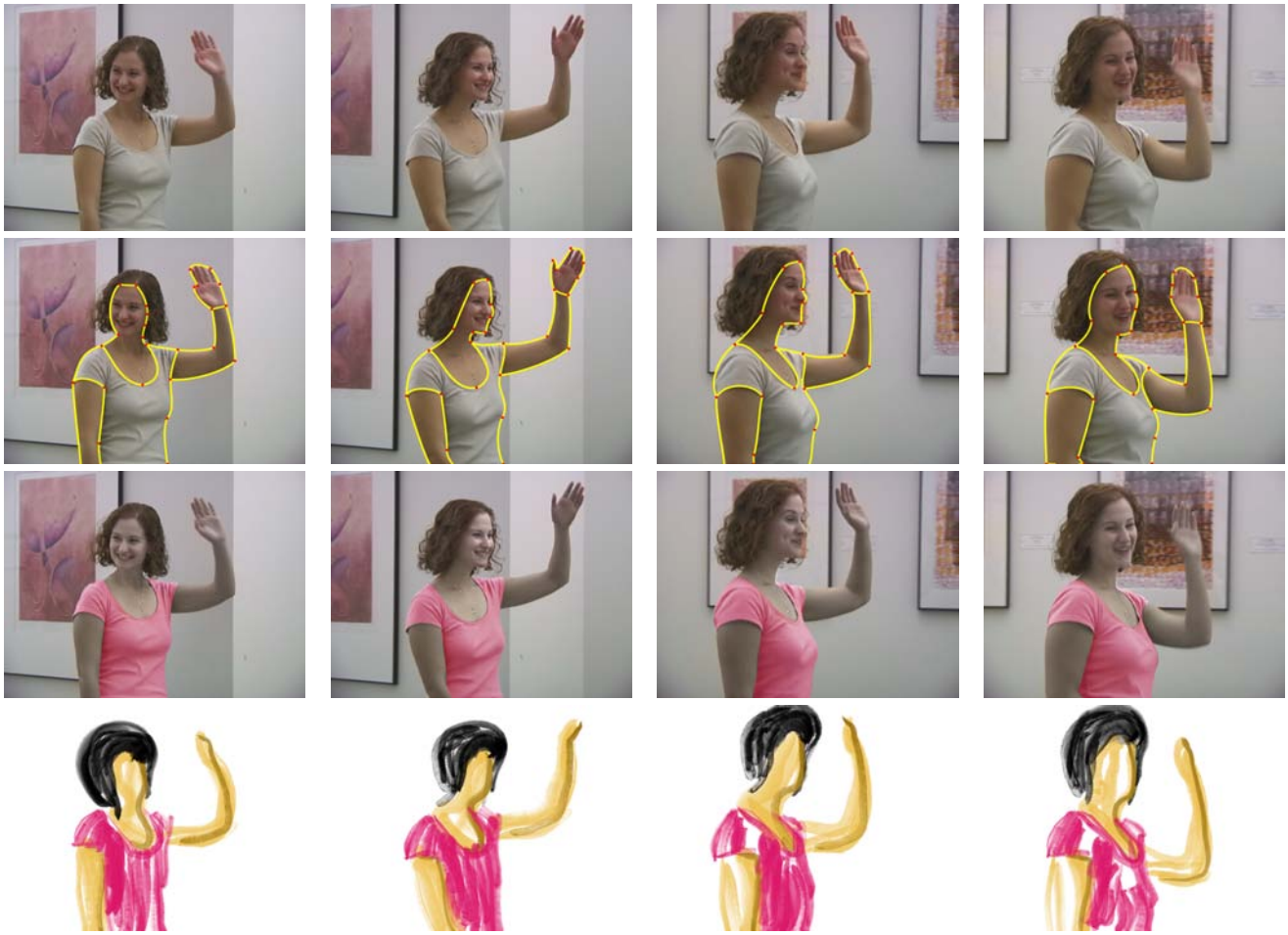

Figure 5 ***First row:*** four frames of a three-second sequence. ***Second row:*** the roto-curves. ***Third row:*** using regions bounded by these curves, we change the color of the shirt, simulate white makeup on the face, and desaturate the arms. ***Fourth row:*** an animation that follows the curves. To demonstrate the stroke propagation algorithm, the artist drew all strokes in the first frame; the later frames were generated automatically without any user editing.

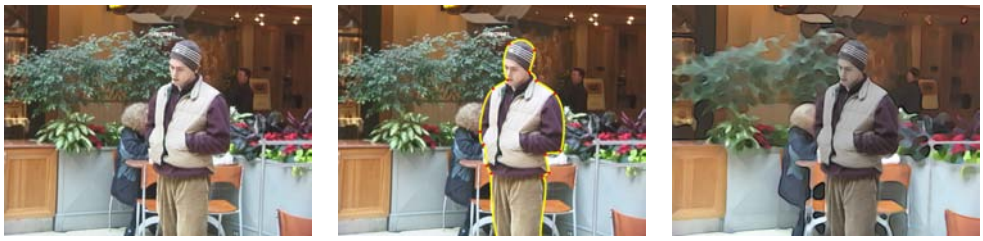

Figure 6 One frame of a six-second sequence. After rotoscoping to separate foreground and background layers, we apply a non-photorealistic filter to the background to place the character in a dream-like world. We also desaturate the foreground to increase contrast between the layers.

on or beyond the image boundaries typically need a little more user effort.

Our system employs spacetime optimization with energy terms that are based on well-known and classic approaches to motion estimation and shape interpolation. However, numerous alternatives formulations exist; exploring some of these alternatives could further reduce user effort.

Finally, there are many problems in computer vision and graphics in which fully-automated techniques produce imperfect results, but could benefit greatly from a small amount of user input. We believe that developing efficient ways to couple user-guidance with numerical optimization techniques is a fertile ground for future research.

Acknowledgements. We are deeply indebted to Alex Hsu, who incorporated our code into his animation system and drew all of the animations. We also thank Siobhan Quinn and Lorie Loeb for assisting with early versions of this system, actors Benjamin Stewart and Amira Fahoum, and David Simons and Matt Silverman. This work was supported in part by National Science Foundation grant IIS-0113007, the UW Animation Research Labs, Microsoft Corporation, an ONR YIP grant, an NSERC Discovery Grant, and the Connaught Fund.

References

AGARWALA, A. 2002. SnakeToonz: A Semi-Automatic Approach to Creating Cel

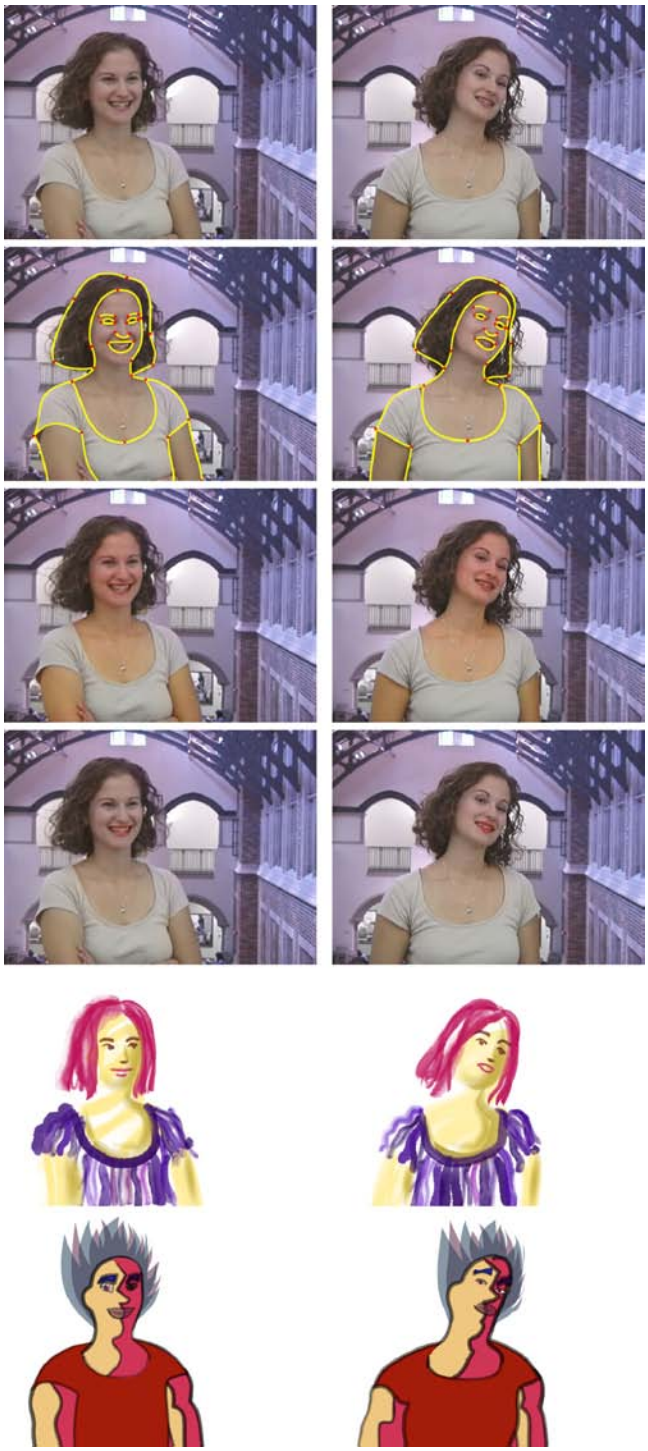

Figure 7 ***First row:*** two frames of a five-second sequence. ***Second row:*** the roto-curves (the first column is a keyframe, the second is interpolated). ***Third row:*** an effect simulating a tan. ***Fourth row:*** an effect simulating lipstick. ***Fifth & sixth row:*** we are also able to animate from the same sequence of roto-curves in multiple styles; two are shown here.

Animation from Video. In *NPAR 2002: Second International Symposium on Non Photorealistic Rendering*, 139–146.

BEIER, T., AND NEELY, S. 1992. Feature-based image metamorphosis. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, vol. 26, 35–42.

BERGEN, J. R., ANANDAN, P., HANNA, K. J., AND HINGORANI, R. 1992. Hierarchical model–based motion estimation. In *European Conference on Computer Vision*, 237–252.

BLACK, M. J., AND ANANDAN, P. 1996. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding 63*, 1, 75–104.

BLAKE, A., AND ISARD, M. 1998. *Active Contours*. Springer-Verlag.

BURTNYK, N., AND WEIN, M. 1976. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *CACM 19* (Oct.), 564–569.

CHUANG, Y.-Y., AGARWALA, A., CURLESS, B., SALESIN, D. H., AND SZELISKI, R. 2002. Video matting of complex scenes. *ACM Transactions on Graphics 21*, 3, 243–248.

COHEN, M. F. 1992. Interactive spacetime control for animation. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, vol. 26, 293–302.

FLEISCHER, M., 1917. Method of Producing Moving Picture Cartoons. US Patent no. 1,242,674.

GEIGER, D., GUPTA, A., COSTA, L., AND VLONTZOS, J. 1995. Dynamic programming for detecting, tracking and matching deformable contours. *IEEE Transactions On Pattern Analysis and Machine Intelligence 17*, 3, 294–302.

GOLDMAN, D., 2003. Computer graphics supervisor, Industrial Light & Magic, personal communication.

HALL, J., GREENHILL, D., AND JONES, G. 1997. Segmenting film sequences using active surfaces. In *International Conference on Image Processing (ICIP)*, 751–754.

HOCH, M., AND LITWINOWICZ, P. C. 1996. A semi-automatic system for edge tracking with snakes. *The Visual Computer 12*, 2, 75–83.

HSU, S. C., AND LEE, I. H. H. 1994. Drawing and animation using skeletal strokes. In *Proceedings of SIGGRAPH 94*, 109–118.

IRANI, M. 2002. Multi-Frame Correspondence Estimation Using Subspace Constraints. *International Journal of Computer Vision 48*, 3, 173–194.

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

KASS, M., WITKIN, A., AND TERZOPOULOS, D. 1987. Snakes: Active contour models. *International Journal of Computer Vision 1*, 4, 321–331.

KORT, A. 2002. Computer aided inbetweening. In *NPAR 2002: Second International Symposium on Non Photorealistic Rendering*, 125–132.

LINKLATER, R., 2001. *Waking Life*. 20th Century Fox.

LITWINOWICZ, P., AND WILLIAMS, L. 1994. Animating images with drawings. In *Proceedings of SIGGRAPH 94*, 409–412.

LITWINOWICZ, P. 1997. Processing images and video for an impressionist effect. In *Proceedings of SIGGRAPH 97*, 407–414.

LUCAS, B. D., AND KANADE, T. 1981. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, 674–679.

LUO, H., AND ELEFTHERIADIS, A. 1999. Spatial temporal active contour interpolation for semi-automatic video object generation. In *International Conference on Image Processing (ICIP)*, 944–948.

MEIER, B. J. 1996. Painterly rendering for animation. In *Proceedings of SIGGRAPH 96*, 477–484.

MITSUNAGA, T., YOKOYAMA, T., AND TOTSUKA, T. 1995. Autokey: Human assisted key extraction. In *Proceedings of SIGGRAPH 95*, 265–272.

MORTENSEN, E. N., AND BARRETT, W. A. 1995. Intelligent scissors for image composition. In *Proceedings of SIGGRAPH 95*, 191–198.

MORTENSEN, E. N. 1999. Vision-assisted image editing. *Computer Graphics 33*, 4 (Nov.), 55–57.

NOCEDAL, J., AND WRIGHT, S. J. 1999. *Numerical Optimization*. Springer.

SEDERBERG, T. W., AND GREENWOOD, E. 1992. A physically based approach to 2d shape blending. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, vol. 26, 25–34.

SEDERBERG, T. W., GAO, P., WANG, G., AND MU, H. 1993. 2d shape blending: An intrinsic solution to the vertex path problem. In *Proceedings of SIGGRAPH 93*, 15–18.

SMITH, A. R., AND BLINN, J. F. 1996. Blue screen matting. In *Proceedings of SIGGRAPH 96*, 259–268.

STEIHAUG, T. 1983. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis 20*, 3, 626–637.

STEWART, S., 2003. Confessions of a roto artist: Three rules for better mattes. http://www.pinnaclesys.com/SupportFiles/Rotoscoping.pdf.

SZELISKI, R. 1990. Fast surface interpolation using hierarchical basis functions. *IEEE Transactions On Pattern Analysis and Machine Intelligence 12*, 6, 513–528.

TORRESANI, L., AND BREGLER, C. 2002. Space-time tracking. In *European Conference on Computer Vision*, 801–802.

WITKIN, A., AND KASS, M. 1988. Spacetime constraints. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, vol. 22, 159–168.