AniPaint: Interactive Painterly Animation from Video

Peter O'Donovan, Aaron Hertzmann, *Member, IEEE*

Abstract—This paper presents an interactive system for creating painterly animation from video sequences. Previous approaches to painterly animation typically emphasize either purely automatic stroke synthesis or purely manual stroke keyframing. Our system supports a spectrum of interaction between these two approaches which allows the user more direct control over stroke synthesis. We introduce an approach for controlling the results of painterly animation: keyframed *Control Strokes* can affect automatic stroke's placement, orientation, movement, and color. Furthermore, we introduce a new automatic synthesis algorithm that traces strokes though a video sequence in a greedy manner, but, instead of a vector field, uses an objective function to guide placement. This allows the method to capture fine details, respect region boundaries, and achieve greater temporal coherence than previous methods. All editing is performed with a WYSIWYG interface where the user can directly refine the animation. We demonstrate a variety of examples using both automatic and user-guided results, with a variety of styles and source videos.

Index Terms—Non-photorealistic rendering, painterly animation, interactive video processing

1 Introduction

PAINTERLY rendering has the potential to enable new forms of animation, by combining the aesthetics of painting with the flexibility of computer tools. However, progress has been slow over the past decade. An animation system must balance the conflicting goals of capturing appearance and fine detail, following object motion and optical flow, avoiding unwanted flickering, reproducing a desired artistic style, all while allowing an animator total control over the result in an efficient manner. Many of these goals are challenging on their own; together, they make the problem formidable. Furthermore, the few extant examples of painterly animations using traditional media provide little guidance, because of the extreme difficulty of traditional paint-on-glass animation; arguably, these examples exhibit much less temporal coherence than we expect from computer-generated animation. New animation tools for painterly animation offer not only greater speed or fewer artifacts than traditional methods; these tools offer the promise of exciting, previously unrealizable styles of animation.

Previous work in painterly animation has typically followed one of two distinct approaches. Purely automatic approaches generate brush strokes as a batch process for an entire video. These methods can produce inspiring results while requiring little effort or skill from a user. Unfortunately, they provide very little artistic control, limited ranges of styles, and often undesirable artifacts such as stroke flickering, poor edge definition, or undesirable stroke movement. In contrast, rotoscoping allows users to animate every stroke. This provides total artistic control, but is generally limited to styles with sparsely-placed strokes, since keyframing requires significant effort. Designing interfaces for directing the style of an automatic painterly rendering algorithm is an important open problem. The user must be allowed high level control to guide the algorithm, while also being able to influence the results at a fine scale. Furthermore, interaction must also be fast enough for iteration, e.g., overnight computation times are unacceptable.

We introduce a system that supports a spectrum of techniques for creating stroke-based painterly animation, including rotoscoping, purely automatic and guided stroke synthesis, and direct stroke manipulation.[1] The user may first perform semi-automatic rotoscoping of an input video sequence, in order to determine video segmentation. A stroke synthesis algorithm is then applied to each image region with user-specified settings. The user may place *Control Strokes* for more fine-tuned control over the placement, orientation, movement, and color of automatic strokes. It is also possible to make detailed refinements by directly painting individual strokes over the video with keyframes. All editing is performed with a WYSIWYG interface, where all strokes are shown, and the user can iteratively redraw new layers or add strokes to refine and correct the result. Novice users can use the system in a purely-automatic batch mode, using preset parameter settings.

An important contribution of this paper is a new automatic synthesis algorithm. There are numerous challenges in

• *P. O'Donovan and A. Hertzmann are with the Department of Computer Science, University of Toronto, Toronto, ON, M5S 2E4.*
E-mail: {odonovan,hertzman}@dgp.toronto.edu

1. Accompanying videos for this paper are located at http://www.dgp.toronto.edu/∼donovan/anipaint/. The Dolphin and Lily videos were purchased from Artbeats.com. We thank Chris Landreth, Copperheart and the NFB for the clip from Ryan, and Liang Lin for the Lady video. We also thank the following individuals for their Creative Commons licensed source videos: Wen Zhang for the Pool Player video, Rick Cooper for the Horses video, Eugenia Loli-Queru for the Jellyfish video, and Mike McCabe for the Sunset video. Links to the original videos will be posted on the project page.

automatically generating appealing and effective painterly animation. First, existing methods rarely respect object boundaries and important fine-scale details that are crucial for depicting a scene clearly. While it is possible to guarantee that details are captured by using many tiny strokes, this results in an overly-faithful duplication of the source. Second, most methods exhibit undesirable flickering artifacts from rapid stroke changes uncorrelated to the underlying video. We present a new approach to automatic synthesis that can capture fine details, abstract regions, and removes most flickering artifacts. While our method still exhibits some flickering artifacts at object boundaries, these issues are greatly reduced compared to previous methods. Rather than tracing strokes through a vector field, the key idea of our algorithm is to guide placement with a carefully-designed objective function. This objective function allows high-fidelity rendering, precise user control, and efficient computation.

We demonstrate numerous examples of painterly animated videos. Often, good results are achieved through purely automatic processing alone, though the user may always refine the result. We also show examples in which a user has modified and refined the results in various ways, producing many different painterly styles. We include several challenging examples of videos with complex motion, occlusions, and transparency, in which capturing fine details are critical to producing good results.

2 Previous Work

Our system builds on many previous stroke-based synthesis algorithms. Painterly processing of images was introduced by Haeberli [1], who proposed both interactive and optimization-based approaches. Meier [2] used 3D geometry to control stroke placement and coherence. Litwinowicz [3] introduced the use of optical flow estimation to automatically process video sequences, using many small brush strokes. Edge preservation was encouraged by truncating strokes at image edges. Hertzmann [4] generalized this approach to long, curved strokes with varying levels of detail. Hertzmann and Perlin [5] applied this method to video processing, but the resulting video exhibits many visual artifacts. Hays and Essa [6] introduced a number of refinements to reduce flickering artifacts and increase visual appeal. Their system creates fuzzy object boundaries and uses only small straight brush strokes. All of these approaches use efficient, algorithmic placement strategies and can produce promising results. However, because they do not use any explicit objective functions, such methods have difficulty balancing multiple goals, such as matching input colors with large strokes while respecting object edges. This approach results in visual artifacts and also provide no fine-scale control to the user. Our method traces strokes in a similar greedy way but, inspired by optimization methods, uses an objective function, rather than a vector field, to guide stroke placement as well as stroke shape.

A few previous approaches use optimization for painterly rendering. Hertzmann [7] introduced an optimization-based approach that captured better fine-scale detail, while allowing the user to paint control weights onto the image. Collomosse and Hall [8] iteratively relax strokes using a genetic algorithm for processing single images. These methods express painting as a global optimization, considering all strokes simultaneously and minimizing the difference between the painting and current frame. However, these methods are too slow for interaction, especially on video sequences, e.g., Hertzmann's algorithm takes four days to process thirty frames of video at 640x360 resolution. This method also lacks temporal energy terms, resulting in significant flickering. At present, optimizing paintings at interactive rates seems far beyond reach. Unlike these previous methods, we define an energy function for each single new stroke given the current painting so far, rather than globally over the entire painting. While perhaps less elegant, this allows us to define an algorithm that is much faster, simpler to implement, and more flexible than previous methods. Our objective function introduces many new terms, including user-specified orientation, deformation, and placement constraints, which greatly enhance the quality and variety of results.

Many existing animation packages allow users to interactively create painterly animation, but with relatively little automation. Most notably, Rotoshop [9] allows users to keyframe stroke movements with standard spline interpolation across time. Our approach is in the spirit of the rotoscoping systems of Agarwala et al. [10] and Liu et al. [11], which combine automatic video processing with interactive editing. While these systems are primarily aimed at tracking and optical flow, Agarwala et al. also showed how painted strokes could be attached to tracking curves. We also draw inspiration from the work of Kalnins et al. [12] and Disney's DeepCanvas [13] which allow users to draw strokes on 3D models. Many previous methods have shown how image or video segmentation can be useful for NPR [14], [15], [16], [17], [18], [19], [20], [21].

Recent work by Lin et al. [22] and Kagaya et al. [23] describe painterly animation systems that share several ideas with our approach. Both use semi-automatic video segmentation, and assign different stroke synthesis parameters to each region. Orientation fields may also be keyframed and combined with image gradients. Both systems perform synthesis as batch computation, without providing an interface for an artist to refine the result, other than by adjusting parameters and re-running. Once created, intermediate strokes cannot be modified. In contrast, we describe a unified system for synthesis and editing strokes. There are a number of stylistic differences between our methods as well. Kagaya et al. produce a very smooth result, due to their use of blending, but still exhibit flickering, especially at object boundaries. Lin et al. produce very clean results with little or no flickering, but their method produces only a single style, and generalizing this method to other styles may be difficult since they use a large database of paintings to learn stroke variation for different regions.

3 Workflow

There are three main steps to our approach: tracking, automatic painting, and interactive editing. These steps are designed to allow a user to iterate them in any order: an artist will generally begin with tracking and automatic painterly animation steps, but may then work back and forth between adjusting automatic processing results and editing individual stroke across time. Each of these steps is illustrated in Fig. 1 and the accompanying video.

3.1 Region tracking and refinement

The first step in our approach is to determine temporal segmentation and dense per-pixel optical flow. This step is performed using the system of Liu et al. [11]. A user marks region boundaries in a few video frames. These regions are tracked through time automatically and then corrected by the user. Using these region boundaries, optical flow is then computed (see Fig. 1). User-specified layer ordering for regions are also used as input to our algorithm. This ordering defines occlusions which are useful for painting layers which cover the background and more detailed foreground layers. Time-varying depths are interpolated as in Liu et al. [11]. These regions will form the basis for automatic processing; the user may come back and refine the region segmentation later if necessary.

One possible alternative to our rotoscoping approach is more advanced video segmentation techniques. By comparison, Lin et al. use Video SnapCut [24], while Kagaya et al. use the approach of Brendel and Todorovic [25]. Both of these approaches use user input to define a foreground and background region which is automatically segmented and iteratively corrected by the user. However, segmentation is difficult when the foreground and background have little contrast, such as the black hair and background of Fig. 1. Also, these methods cannot correctly matte multiple occluding layers. Defining regions boundaries with control points allows an orientation and flow-field to be created independently of the video, which is important for occluded regions. Though these advanced techiques are useful, rotoscoping is the most general approach for interactive segmentation, and is widely used in film production. Our system shows that even relatively coarse segmentation from rotoscoping permits good quality results.

3.2 Automatic painterly animation

In this step, the user selects a region to paint, and the system produces a *layer* of paint strokes for the region across time. Further layers are generated separately and finally composited together. The user controls the results by setting stylistic parameters for automatic processing and by manually drawing Control Strokes, described below. This process, described in Section 4, may be undone, repeated, and refined as desired.

3.3 Control Strokes

Control Strokes allow a user to guide the automatic synthesis process. Three different types of *Control Strokes* are defined that influence automatically generated strokes. *Guide Strokes* control the orientation and movement of automatic strokes. *Color Strokes* control the color of automatic strokes. *Seed Strokes* specify where automatic strokes will be initialized, in order to regenerate the painting in a very localized region. Once drawn, Control Strokes are used whenever the user runs the automatic synthesis procedure. See Fig. 2 for an example.

Automatic stroke placement is often sufficient for large regions, but not for crucial details like faces and eyes. For these cases, our system also allows user-drawn strokes to be keyframed. Furthermore, a stroke may play multiple roles. For example, the user might keyframe a blue stroke that will appear in the final animation. This stroke could then also be marked as a Color and/or Guide Stroke, so that the nearby strokes will be drawn with a similar color and/or orientation to the new stroke. Control Strokes need not be rendered in the final animation, however.

Interactive stroke keyframing. Our system provides tools for keyframing strokes, similar to the systems of Agarwala [10] and Sabiston [9]. The artist draws a stroke in a particular frame, and the stroke may then be automatically propagated forward and backward in time using optical flow, or another flow field (Section 4.6) for a specified number of frames. The stroke may then be edited in another frame using oversketching [26], or control point pushing or pulling. The user may also select groups of strokes and move them together. Modifying a stroke fixes it as a keyframe and the edits are propagated by linear blending to the surrounding keyframes. If a control point $\mathbf{p}$ at time t_1 is displaced by a vector $\mathbf{d}$, and the previous keyframe is at frame t_0, then the control points at intermediate frame t are displaced by $\mathbf{d}(t - t_0)/(t_1 - t_0)$. If no other keyframe is defined, the edit is applied directly to the other strokes. Strokes could also be refined by optimization, but this would be much slower. User strokes can be selected and their properties (texture, color, width, etc.) changed at any time. Strokes are associated to the layer in which they were drawn, and can be re-assigned to other layers.

Rendering. Strokes are rendered at a user-specified resolution. Strokes for each region are rendered separately and composited according to user-specified region depths. In each region, user-drawn strokes are always drawn over automatic strokes. Strokes can be rendered with height-map texturing [27].

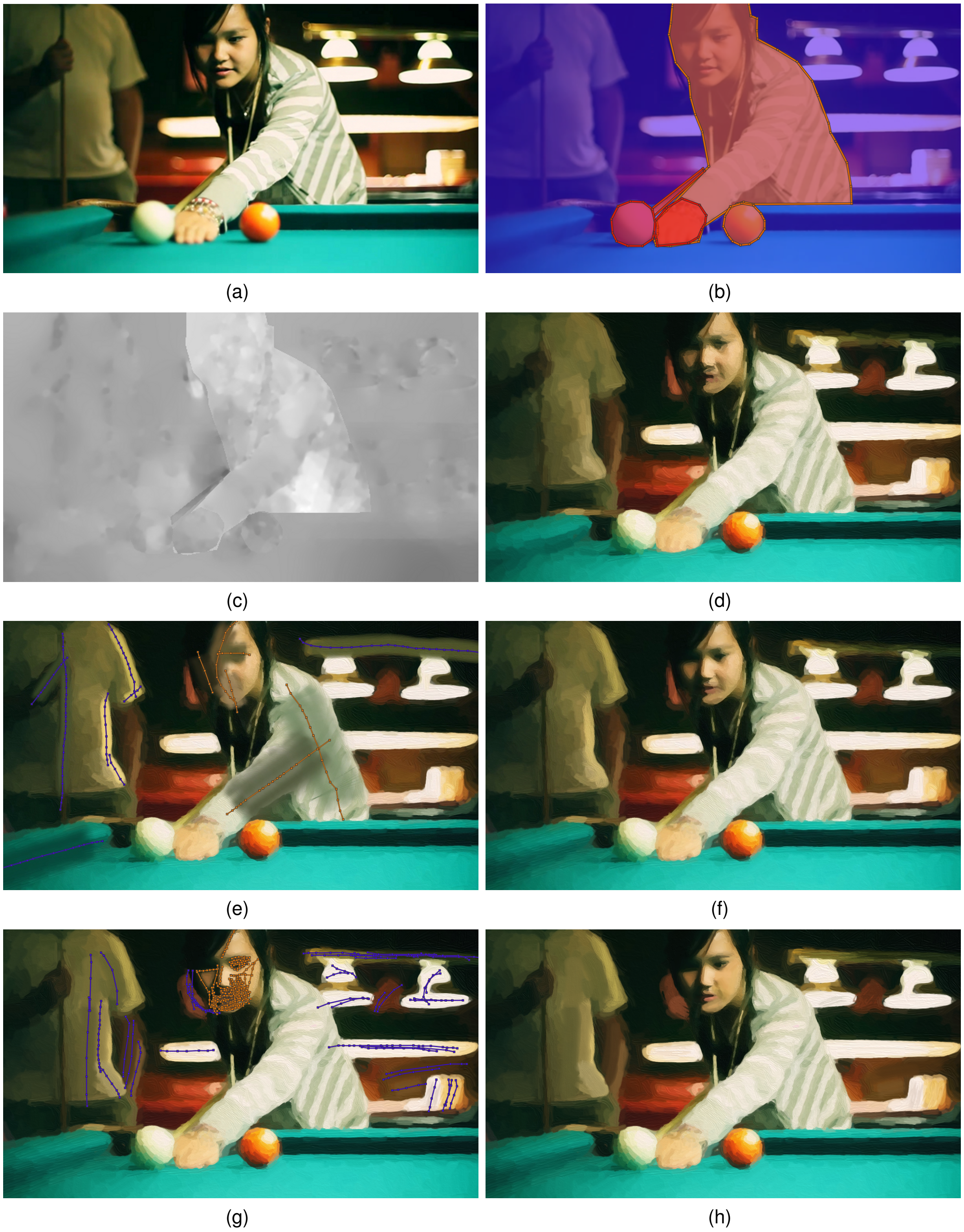


Fig. 1. Sample animation workflow: a) Source video, b) Rotoscoped regions, c) Layer-aware optical flow, d) Automatically generated painting, e) User-drawn Control Strokes, f) Result after automatic synthesis using Control Strokes, g) User-drawn detail strokes, h) Final result.

4 Automatic Painterly Animation

We now describe a new automatic painting algorithm for processing video sequences. This process is designed to produce strokes that move with a flow field, depict clear region boundaries and fine-scale detail, and allow different styles to be applied to different image regions. In later

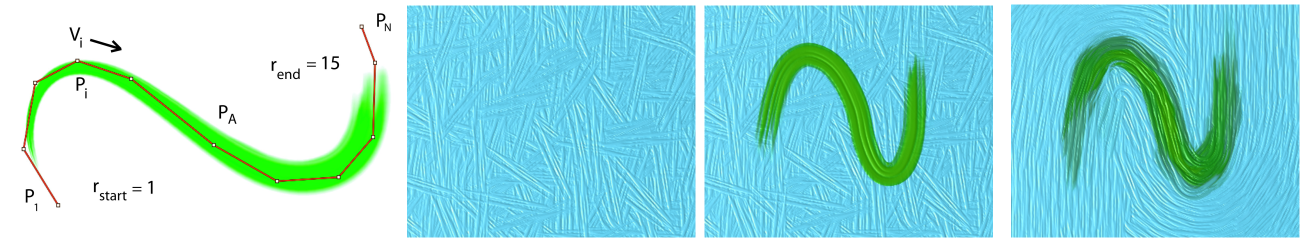


Fig. 2. Automatic and User-Drawn Strokes. (a) A stroke $\mathbf{s}$ is represented by control points $\mathbf{p}_{1:N}$, a starting and ending width $\mathbf{r} = (r_{start}, r_{end})$, a texture, and a color $\mathbf{c}$. The vector $\mathbf{v}_i$ is the normalized direction between $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$. One of the control points is designated the anchor point $\mathbf{p}_A$. (b) Automatically-generated strokes in a textureless region. (c) Stroke drawn as a Guide and Color Stroke. (d) After running synthesis, new strokes are affected by the color and orientation of the Control Stroke. Rendering the Control Stroke is optional.

stages, an artist may edit and refine the results, as described in Section 3.

The input to the algorithm is a video sequence, a segmentation of the video, depths for the segments, dense flow fields, stroke orientation fields, and a set of stylistic parameter settings. Creation of the orientation fields, flow fields and parameter settings are described in the following sections. To compute temporal segmentation and dense optical flow, we use Liu et al.'s [11] interactive system.

Our algorithm repeatedly places strokes on the canvas, tracing in an initial frame then propagating forward and backward in time. However, unlike previous algorithms which follow an orientation field, each stroke has its own objective function which is minimized when choosing control points. This is also distinct from previous optimization methods which consider a global optimization over all strokes. Our approach also allows many new terms in the objective function which enhance the variety and quality of possible styles, such as drawing fine black strokes near edges as in Fig. 8. We describe the energy function next, and the placement algorithm in Section 4.4. In the accompanying video, we show a comparison of stroke placement without the energy formulation, using only optical flow and redrawing the stroke at each frame.

4.1 Stroke Representation

A stroke $\mathbf{s}$ at a single video frame t is represented by a list of N 2D control points $\mathbf{p}_{1:N}$ (Fig. 2). We designate one of these points as the anchor point $\mathbf{p}_A$. Each stroke has a color $\mathbf{c}$, and a starting and ending width $\mathbf{r} = (r_{start}, r_{end})$. A stroke is rendered as a cubic B-spline, with width linearly interpolated from the starting to ending width. An animated stroke is represented by one stroke $\mathbf{s}_t$ for each frame of a sequence. The same representation is used for both automatic and user strokes.

4.2 Stroke Energy Function

We define an energy function which encapsulates the goals of painterly rendering from a video. To avoid the difficulty of optimizing a global energy function, we define an energy function for a single stroke $\mathbf{s}$ in a single frame t, . The energy function is defined based on the source image at time t, as well as a corresponding stroke $\mathbf{s}'$ in the previous or next frame.

$$E(\mathbf{s}) = \sum_i w_i E_i(\mathbf{s}, \mathbf{s}') \tag{1}$$

where each energy term has a weight w_i, and may depend on both $\mathbf{s}$ and $\mathbf{s}'$, or just $\mathbf{s}$.

Image terms. The color energy penalizes differences between the colors of a stroke and their corresponding pixels:

$$E_{color}(\mathbf{s}) = \sum_{j \in \mathbf{s}} q(||\mathbf{c} - \mathbf{g}_\sigma(\mathbf{p}_j)||^2, \tau_{col}) \tag{2}$$

where $\mathbf{c}$ is the stroke's color, $\mathbf{g}_\sigma(\mathbf{p}_j)$ is the color of a Gaussian-blurred version of the image at the location of control point j, and σ is the variance of the blur kernel, which depends on the linear interpolated stroke width at the current control point. The Gaussian kernel may be optionally masked by the region segmentation and re-normalized to reduce color bleeding between regions. Colors are represented as 3D vectors of RGB values. The thresholding operator q is used to prevent very large deviations from the target color:

$$q(d, \tau) = \begin{cases} d & d < \tau \\ \infty & \text{otherwise} \end{cases} \tag{3}$$

where τ is a threshold.

The user may specify a target orientation field which defines a direction vector $\mathbf{d}(\mathbf{x})$ and magnitude $m(\mathbf{x})$ at each image point $\mathbf{x}$. We define a term which encourages strokes to follow this field:

$$E_{orientation}(\mathbf{s}) = \sum_{j \in \mathbf{s}} \sum_{\mathbf{x} \in \text{line}(\mathbf{p}_j, \mathbf{p}_{j+1}))} m(\mathbf{x})(1 - |\mathbf{d}(\mathbf{x}) \cdot \mathbf{v}_j|) \tag{4}$$

where $\mathbf{v}_j$ is the unit direction vector between consecutive control points:

$$\mathbf{v}_j = (\mathbf{p}_{j+1} - \mathbf{p}_j)/||\mathbf{p}_{j+1} - \mathbf{p}_j|| \tag{5}$$

For each control point $\mathbf{p}_j$, this term sums over all pixels $\mathbf{x}$ on the line between $\mathbf{p}_j$ and $\mathbf{p}_{j+1}$. The possible choices of direction fields are given in Section 4.5.

Preserving image edges is important to depicting region boundaries and fine-scale details in an image. We add an image edge term:

$$E_{edge}(\mathbf{s}) = \sum_{j \in \mathbf{s}} e_\sigma(\mathbf{p}_j) \tag{6}$$

where $e_\sigma(\mathbf{p})$ is a real-valued edge image computed by convolving the image with Gaussian and edge-detection kernels and applying non-maxima supression [28]. This result is then blurred with a Gaussian kernel of variance σ, which depends on the current stroke width. This will penalize strokes which overlap edges and shrink the stroke width.

Stroke shape terms. These terms are used to constrain stroke shapes within a single frame to be smooth and to have similar widths:

$$E_{smoothness}(\mathbf{s}) = \exp\left(\cos^{-1}(\mathbf{v}_j \cdot \mathbf{v}_{j+1})/\sigma_{smooth}\right) \tag{7}$$

$$E_{userwidth}(\mathbf{s}) = q\left(\frac{||\mathbf{r} - \mathbf{r}_{user}||}{||\mathbf{r}_{user}||}, \sigma_{width}\right) \tag{8}$$

$$E_{grow}(\mathbf{s}) = ||\mathbf{r}||^{-1} \tag{9}$$

where $\mathbf{r}_{user}$ is a user-specified vector indicating the target stroke width at the endpoints. The smoothness term defines how flexible or rigid the strokes are allowed to be when traced. The user width term constrains the strokes to remain close to a specified stroke size, otherwise strokes could become extremely large or small. The growth term can be used to encourage larger strokes.

Temporal coherence terms. The following terms encourage similarity between a stroke $\mathbf{s}$ in one frame, and the stroke $\mathbf{s}'$ in an adjacent frame, and do not apply when initially tracing a stroke in its first frame. These terms encourage strokes adjacent in time to have similar shapes:

$$E_{shapechange}(\mathbf{s}, \mathbf{s}') = \sum_{j \in \mathbf{s}} \exp\left(\cos^{-1}(\mathbf{v}_j \cdot \mathbf{v}'_j)/\sigma_{coh}\right) \tag{10}$$

$$E_{widthchange}(\mathbf{s}, \mathbf{s}') = \frac{||\mathbf{r} - \mathbf{r}'||^2}{||\mathbf{r}'||^2} \tag{11}$$

4.3 Location Constraints

Stroke control points locations may be constrained in order to use different styles for different parts of an image. For example, one may wish to use a style with large strokes to fill the interiors of regions, and then refine the region boundaries and image edges with a style employing small strokes. All stroke control points $\mathbf{p}_i$ are constrained to lie within a specified tracked image region. Control points may further be restricted to lie:

1) within a particular distance τ_{dist} of region boundaries, images edges, or Guide Strokes,
2) on pixels where the source video color lies within a luminance range $(\tau_{minL}, \tau_{maxL})$,
3) on pixels where the source video color is within a threshold τ_{col} of the stroke's color $\mathbf{c}$, and/or
4) at points $\mathbf{x}$ where the density $\delta(\mathbf{x})$ of previous strokes is below a particular threshold $\tau_{density}$. The density function $\delta(\mathbf{x})$ is computed by rendering all other strokes in white with alpha-compositing.

Stroke size and time duration may also be constrained. All strokes are constrained to have between $\tau_{minLength}$ to $\tau_{maxLength}$ control points. To reduce short lived strokes, we define a hard constraint on the stroke age which forces each stroke to last for at least τ_{minAge} frames. Variation can be encouraged by limiting strokes to last at most τ_{maxAge} frames. Default parameters for the system are listed in Appendix (see supplementary material or the project page).

4.4 Algorithm

Our greedy stroke placement algorithm for a particular region works as follows. A set of seed points are generated within a particular region throughout the video. Seed points are either placed randomly, or from user-drawn Seed Strokes. Seeds are placed only in points satisfying the constraints in Section 4.3. Strokes are then traced from the pixels by the following procedure:

1. Select a seed point. A seed point is removed from the list of seeds. Optionally, these seed points may be ordered by edge magnitude (picking seeds that lie on edges first), or current color difference error between the painting and the video as the algorithm progresses (picking seeds that lie on regions of greater error first).

A seed point specifies a particular location $\mathbf{x}_{seed}$ and time t in a video. The seed point can be adjusted with the stroke width to line up with ridges in the orientation field. This is done by determining the nearest maxima in the orientation magnitude image $m(\mathbf{x})$ using gradient ascent, and if this distance is greater than the stroke width, moving the seed point by gradient descent to line up with the image edge. This point is then checked to see if it satisfies the Location Constraints (Section 4.3). If so, the first control point $\mathbf{p}_A$ is placed at the seed point and designated the stroke's Anchor Point. If not, the stroke is not traced.

2. Trace a stroke in a single frame. The stroke's color $\mathbf{c}$ is set to the pixel color (after blurring the image by variance σ) at anchor point $\mathbf{p}_A$. We then repeatedly add new control points. Each new control point is constrained to lie $(r_{start} + r_{end})/2$ pixels away from the previous control point. The point that minimizes the complete stroke energy (Equation1), subject to the constraints in Section 4.3 is found by a local brute-force search over angles (every $5\,^\circ$) around $\mathbf{p}_A$.

Iteration continues until placing the next control point would cause violation of a constraint. The process is per-

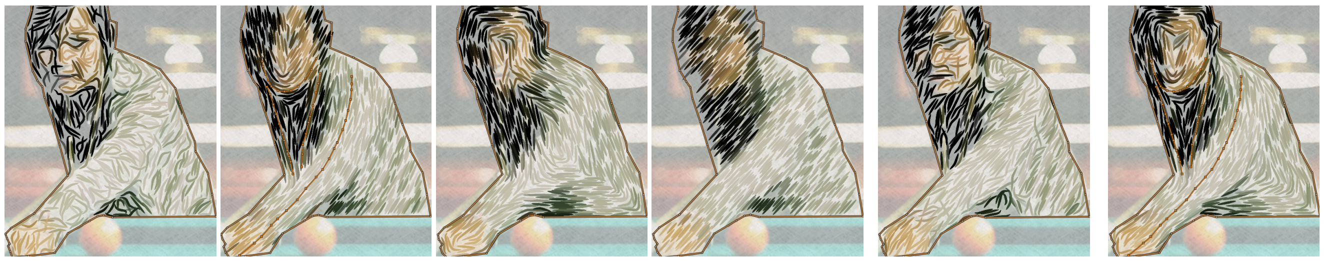


Fig. 3. Examples of orientation fields provided by our system. Mixture fields provide weighted combinations of fields.

formed twice, tracing strokes both forward and backward from the seed point.

3. Adjust stroke width and set color. Optionally, the stroke width (r_{start}, r_{end}) is optimized by a brute-force search over widths to minimize the stroke energy. The stroke color $\mathbf{c}$ is set by averaging the colors at the control points (blurred by variance σ depending on the width). Optionally, an HSV offset and Gaussian noise is added. [2]

4. Propagate the stroke through time. The Anchor Point $\mathbf{p}_A$ is moved to the next video frame according to the flow field [6], [3]. The stroke is then traced in this frame, by repeating the above steps 2-3. The energy $E(\mathbf{s}, \mathbf{s}')$ for this stroke $\mathbf{s}$ uses the stroke from the previous frame as $\mathbf{s}'$. Stroke colors may be smoothed over time using a moving average: $\bar{\mathbf{c}}_i = \alpha_c \mathbf{c}_i + (1-\alpha_c)\bar{\mathbf{c}}_{i-1}$. This process is repeated to propagate the stroke forward and backward in time. By default, $\alpha_c = 0.3$, though for quickly changing regions, $\alpha_c = 0.5$ is useful. Once the strokes are computed, alpha-blending is used to fade the stroke in at the beginning of its lifetime and fade out at the end. Note that all strokes have the same number of control points. As the stroke is being traced, it will not exceed the length of the previous frame. All strokes are finally truncated to the shortest length over all frames.

4.5 Orientation Fields

There are several options that may be used for the user-specified orientation field $\mathbf{d}(\mathbf{x})$ in Equation 4. Fig. 3 shows an example of these orientation fields. Each orientation field also has an associated magnitude field $m(\mathbf{x})$. The options we provide are as follows.

Image orientation. To follow image orientations, the *Edge Tangent Field* (ETF) may be used. The ETF is computed by a non-linear filtering of image gradient [29]. The magnitude $m(\mathbf{x})$ is computed by performing the analogous non-linear filtering of image gradient magnitudes.

Guide Strokes. As described in Section 3.2, the user may paint their own strokes into the image in order to define an orientation field. These strokes are called *Guide Strokes.* The stroke is converted to a cubic B-spline, and then sampled to produce a set of positions and tangents $(\mathbf{p}_i, \mathbf{v}_i)$ along the stroke. This stroke then defines an orientation field by Radial Basis Function averaging:

$$d(\mathbf{x}) = \frac{\sum_i \mathbf{v}_i w(\mathbf{x}, \mathbf{p}_i)}{\sum_i w(\mathbf{x}, \mathbf{p}_i)} \tag{12}$$

where the sum is over the five points nearest to $\mathbf{x}$ with basis functions $w(\mathbf{x}, \mathbf{y}) = \exp(-\sqrt{||\mathbf{x} - \mathbf{y}||}/5)$. This procedure is similar in spirit of [30], but allows keyframed orientations through time. The magnitude is a function of the distance to the nearest control point: $m(\mathbf{x}) = \min_i 20 \cdot \exp(-||\mathbf{x} - \mathbf{p}_i||/10) + 0.5$

Region Boundary Field. Near region boundaries, we may wish to draw strokes that closely follow the region boundaries. The *Region Boundary* field is computed by creating a distance transform from the boundaries, and computing the gradient of the distance field [31], [19]. The magnitude is a function of the distance to the boundary as above.

Constant orientation. The user may also define a constant field $d(\mathbf{x}) = \mathbf{v}_{const}$, with a constant magnitude $m(\mathbf{x}) = 1$.

Mixture fields. Linear combinations of these fields may also be used. For example, the user may wish to specify the orientation in one part of a region manually using a Guide Stroke while still following the image in the rest of the region. The user may also wish the strokes to follow image gradients near edges, but follow a fixed orientation or the region boundary in homogenous regions, e.g. the sky in the Sunset of Fig. 8. This is often useful as gradients are poor in untextured regions (see Fig. 2).

In order to respect 180°-symmetries, orientation fields are combined as follows [32]: Let the angular representation of field f be $\theta_f(\mathbf{x}) = \tan^{-1}(\mathbf{d}_x(\mathbf{x}), \mathbf{d}_y(\mathbf{x}))$. Each orientation field f is rotated as $\bar{\mathbf{d}}(\mathbf{x}) = (\cos(2\theta(\mathbf{x})), \sin(2\theta(\mathbf{x})))$. The mixture field is then

$$\theta_{mix}(\mathbf{x}) = \frac{1}{2}\tan^{-1}\left(\sum_f \bar{\mathbf{d}}(\mathbf{x}) m_f(\mathbf{x}) w_f\right) \tag{13}$$

where $m_f(\mathbf{x})$ is the magnitude of field f and w_f is its weight. The orientation is then converted back to vector form as $\mathbf{d}_{mix}(\mathbf{x}) = (\cos(\theta_{mix}(\mathbf{x})), \sin(\theta_{mix}(\mathbf{x})))$. The magnitude of the mixture field is the weighted sum of base magnitudes: $\sum_f w_f m_f(\mathbf{x})$. Note that the above algorithm

2. This offset is specified in HSV for user convenience and converted to RGB. The RGB color space is used in all other calculations.

is further controlled by user weights on each of the fields. This allows the user to disable certain fields, or increase the relative effects of particular fields.

4.6 Flow Fields

The flow field $\mathbf{f}(\mathbf{x})$ defines the displacement between frames for each pixel. As with orientation fields, several options are available. The default option is to use the **optical flow** field computing during the rotoscoping step. Flow from user-keyframed **Guide Strokes** may be used by blending temporal displacements of user-drawn strokes. Specifically, we blend the displacements $\mathbf{f}_i$ of the control points $\mathbf{p}_i$ between video frames. Similarly, **Region Boundary Flow** is computed by averaging displacements of region boundaries. A **constant flow field** may be used [5]. Finally, flow fields may be combined by weighted averaging into **Mixture Flow fields:**

$$\mathbf{f}_{mix}(\mathbf{x}) = \frac{\sum_f t_f(\mathbf{x}) m_f(\mathbf{x}) w_f}{\sum_f m_f(\mathbf{x}) w_f} \tag{14}$$

where $t_f(\mathbf{x})$ is the temporal flow for pixel $\mathbf{x}$ of temporal flow field f, $m_f(\mathbf{x})$ is the corresponding spatial orientation magnitude at $\mathbf{x}$ for the flow field f, and w_f is a user-specified weight. Commonly, the weights for the orientation and the flow fields are identical. However, it is possible to use different weights for stylistic effects, and also to compensate for poor optical flow calculation.

5 Results and Techniques

In this section, we describe the results of our system, and the lessons learned in its development. Our system enables many different approaches to painterly animation spanning fully automatic and manual approaches. Videos may be processed entirely automatically, using default optical flow results and preset parameters. A user may rotoscope the video if desired; roto-curves are most useful for correcting occlusion boundaries, and to define regions that will have different styles. The user may also keyframe Control Strokes and/or keyframe painting strokes as described previously. Examples with various combinations of these options are shown in the accompanying video.

While rotoscoping can be important for videos like the Pool Player (Fig. 1) with significant occlusions, our approach does not depend on it. Many videos require no rotoscoping, including the Horses, Sunset, Ryan, and Jellyfish sequences (Figs 4 and 8). In many cases, it is not desirable to use strict rotoscoping boundaries to control stroke placement for refinement or spatially varying styles. We find Seed Strokes are an extremely useful way to refine individual regions by generating new strokes in a localized area. New paint strokes are synthesized around the Seed Stroke, and the user may immediately save, refine, or undo the results. This allows a fluid approach to applying different styles, rather than first segmenting discrete regions to apply different styles; one may always perform rotoscoping later if necessary without affecting previous automatic or user strokes. Creating new regions, and possibly changing the optical flow of a region after rotoscoping, does not affectly previously drawn strokes. However, when a region is deleted the associated automatic and user-drawn strokes are removed.

In most cases, we follow the animation practice of working “on twos,” i.e., at 12 fps rather than 24 fps. As noted by Hertzmann and Perlin [5], working “on ones” (at 24 fps) can give too strong an impression of video. However, working on ones for fast-moving objects can reduce temporal aliasing. Alternatively, we find the “double exposure” technique of working on twos and then using blending to create intermediate frames creates smoother motion.

5.1 Time and effort

Our system supports two classes of users, novice users and professional animators, whose time and effort requirements may vary significantly. For novice users or those with time constraints, purely automatic painterly rendering of multiple stroke layers with pre-specified parameter settings can be run without rotoscoping or user input. Creating a layer of 500 strokes over 50 frames for a large region (640x480), can take between 0.5 and 2 minutes, depending on the parameter settings. Although there are many parameters to the stroke placement algorithms, we provide a collection of preset parameter settings that enable different styles. Optimization parameters are generally stable and require little tweaking. Furthermore, most stroke parameters are easy to understand: e.g., short vs. long strokes, etc. In this mode, an animation can be created in 30-60 minutes. Script files can also be used to animate several sequences, or to generate several styles automatically.

For professional animators, we also provide tools which allow finer control and refinement of the automatic synthesis. It is worth noting that all high-quality animation, whether hand-drawn and computer-generated, is extraordinarily labor-intensive. Rotoscoping can take several hours depending on the number of objects and the complexity of movement. However, for professional animators, the goal is not to make animation faster, but to enable new kinds of animations. To this end, our system is designed to support iteration and experimentation. This mode’s time requirements are open-ended, but an animation can usually be created in 3-4 hours. We also include several tools reducing the effort of stroke keyframing (see Section 3).

For the Impressionist example in Fig. 1, 35-45 stroke layers (a set of strokes generated by our automatic algorithm) were drawn for the player and background, with 10 for the hand, cue, and balls. For the abstracted style, 8 layers were used on the background, 12 for the player, and 3-4 for the other regions. For the “sketchy” style, 5-8 layers were used for the background and woman, and 5 for the balls. For the sparse black style, only a single layer was drawn for each

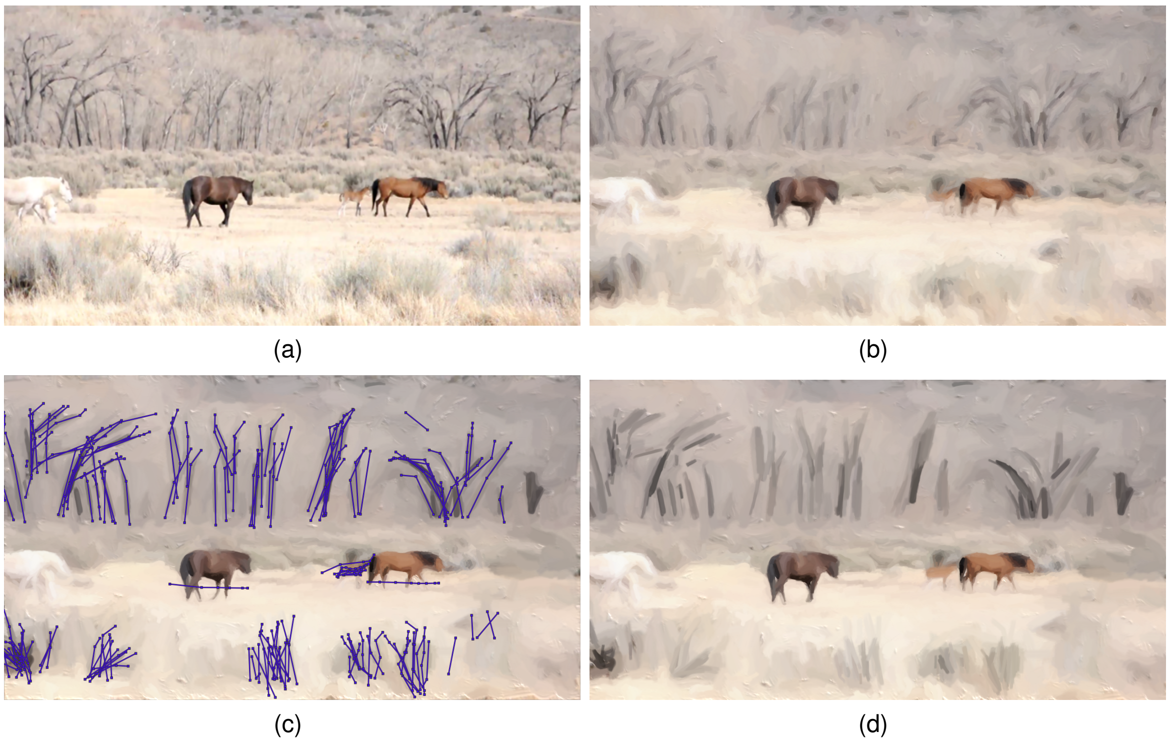


Fig. 4. Painting refinement. a) Source video, b) Result after automatic algorithm, c) Control Strokes drawn to refine trees and horses, d) Final result after new synthesis.

region. Ranges are used as layers may not extend to all frames, often when Seed Strokes are used.

5.2 Comparisons to other systems

In the accompanying video, we demonstrate our system on videos used by previous painterly animation algorithms [6], [5], [7], [23], [22]. Previous methods tend to have significant stroke flickering and poor handling of details/edges. Our method yields more stable strokes and finer detail preservation. Furthermore, previous methods each have a few built-in styles, with little user control over the style provided to the user beyond parameter tuning, whereas our method supports creating very distinct styles. Fig. 5 shows a screenshot of the comparison with the results of Lin et al. [22] and Kagaya et al. [23]. Since the work of Lin et al. automatically applies different styles to different regions, we use a few Seed Strokes to manually influence the style. Both Lin et al. and our system have good fine-detail, edge preservation, and temporal coherence. The fundamental distinction between the systems is that Lin et al.'s approach is automatic, whereas our approach emphasizes user control over the painterly process. When comparing with Kagaya et al., we demonstrate one example with a similar level of input, and another using the finer edits possible with our system.

Most previous painterly animation algorithms are far too slow for interactive applications, taking several hours to animate a sequence. Hertzmann [5] took 3-4 hours to render a single frame with a 3.4 GHz CPU, taking weeks to complete a longer sequence. Hays and Essa [6] took 300 seconds a frame to paint in an impressionist style. In Lin et al. [22], rendering takes between 150-250 seconds a frame. In contrast, our system is significantly faster. Depending on complexity, our system requires 0.5-2.5 seconds per layer per frame, and thus 5-25 seconds for a 10-layer sequence. Kagaya et al. [23] report an averaging 2.5 seconds per frame.

All timing results except Hays' (created on a 2004 era desktop) are on modern desktops, with CPUs between 3.4-3.8 GHz. However, CPU architecture, memory, code optimization, among other factors can all affect performance and were not reported. Hence, the timing results are not perfectly comparable.

5.3 Stylistic Variations.

Here we briefly describe a few "case studies," illustrating different examples of producing different styles interactively. Figs. 1 and 6 show several styles applied to the Pool Player sequence: an impressionist style, some "sketchy" styles, and coarser abstracted styles. For the Impressionist style, Seed Strokes were used to correct areas after automatic processing and rendered user strokes were drawn in the face. The abstract style in Fig. 6 a) was created with Guide Strokes in the face and jacket, a Color Stroke is used to emphasize the jacket contour. Strokes were also constrained to not be drawn in dark or occluded pixels.

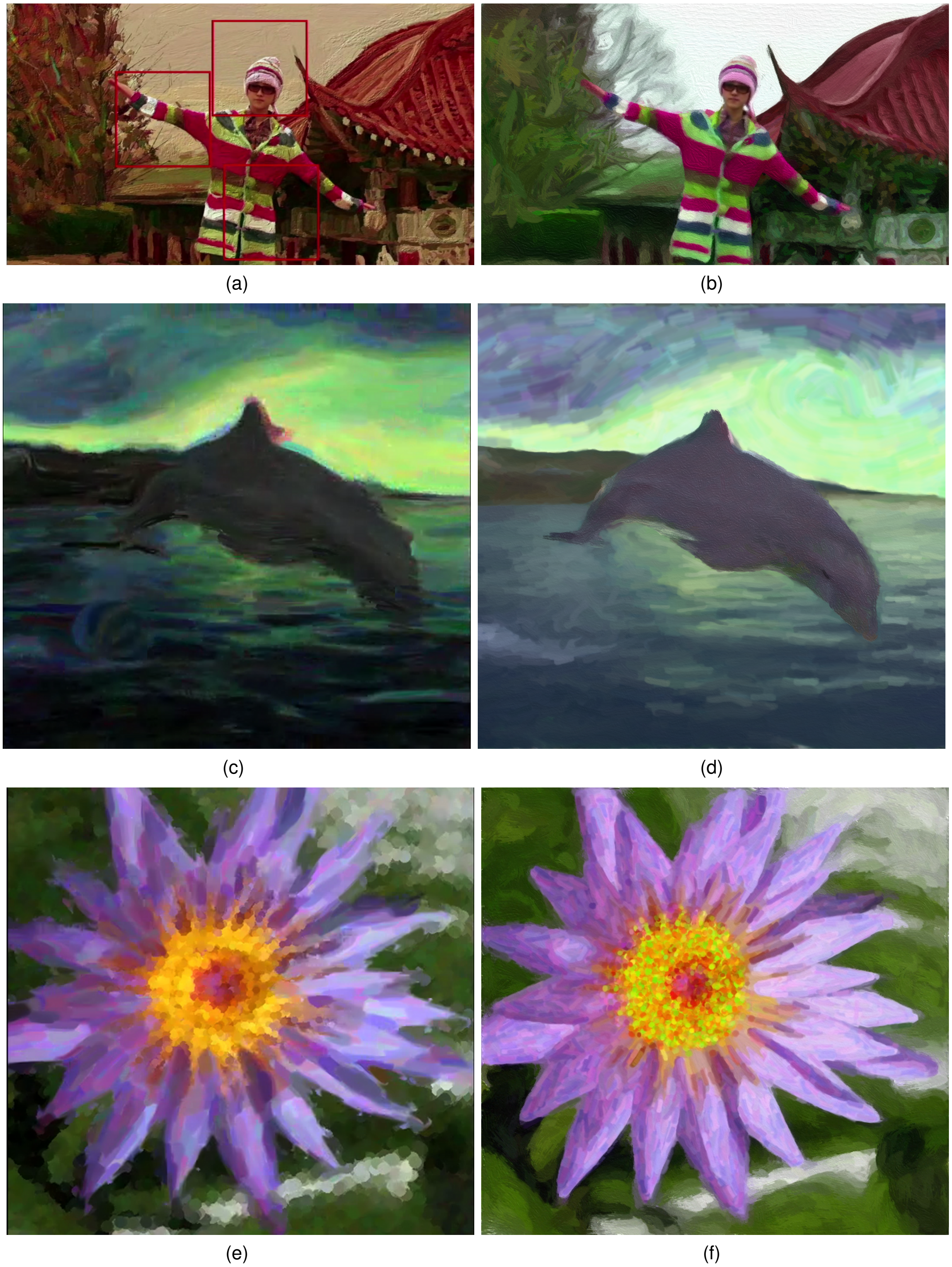


Fig. 5. Comparison with other systems. a-b) Lady sequence rendered by [22] and ours, c-d) Dolphin sequence rendered by [23] and ours, e-f) Lily sequence rendered by [23] and ours

Control Strokes were used to further refine the face and jacket with fine-details in Fig. 6 c). The "sketchy" style in Fig. 6 b). was created by filling regions with constant color using region boundaries and Seed Strokes. Next, Guide/Seed Strokes were drawn as black outlines. Finally, fine strokes were automatically drawn near those strokes. The sparse black style of Fig. 6 d) was created with a low density constraint, Seed Strokes to control stroke width in the face and jacket, and a few user-drawn detail strokes in the face.

Figs. 5 and 7, illustrate two styles for the Dolphin sequence. In Fig. 5, a more impressionist style is created with fine details on the dolphin, including manually drawn eyes, and a 'Van Gogh' style sky drawn with guide strokes. Fig. 7, shows a different style, with partially transparent and short-lived strokes drawn for the waves, an abstracted dolphin, and several rain layers drawn with fixed orientation and flow fields. This figure also illustrates how an animation is created, with the user modifying and experimenting on particular regions.

Fig. 4 shows a second workflow example of our system with source, automatic processing, and interactive editing. After automatic processing, the trees were drawn as Guide/Seed/Color Strokes with further automatic strokes generated. Seed Strokes were used to draw finer strokes near the horses. Rendered user strokes were drawn to refine the foal. Fig. 8 shows more results of our system. In the Jellyfish sequence, only automatic processing was used in an impressionist style. In the Castle sequence, the video was segmented and a coarse style applied to the castle and sky, with an automatic sketchy line-drawing style applied. A finer set of strokes was applied to the foliage. In the Sunset video, Guide and Seed Strokes were used to exaggerate the star shape of the sunset. For the Ryan sequence, after initial processing, Seed Strokes were used for several features of the face to improve fine details. Lastly, large strokes were automatically drawn in homogenous regions while preserving edges.

6 Conclusion

In this work, we define a new approach for controlling automatic animation by combining automatic painterly rendering with detailed interactive controls. Our system allows the user to work in a range of modalities, from high-level stylistic specifications, to interactive over individual regions, to keyframing individual strokes. Our automatic stroke synthesis algorithm has been designed to capture fine detail and object boundaries—which cause significant problems with previous methods—while also providing abstraction and temporal coherence. As demonstrated in the videos, our system can be used for a range of video inputs and styles, including challenging cases with significant amounts of detail. Similar to WYSIWYG NPR [12], our system suggests new ways in which automatic and interactive techniques can be combined in NPR.

Our examples exhibit numerous small artifacts. Many of these arise from the greedy nature of our optimization, such as the limited consideration of previously-drawn strokes during tracking. In many of our examples, strokes in static regions slowly appear and disappear. This could be improved by better optimization, or by explicit user controls to force regions to be static over certain durations. Inaccurate optical flow can cause surfaces to deform in unexpected ways, though further correction with flow mixtures is possible.

More generally, it may be possible to create more powerful ways to interact with automatic NPR synthesis. For example, one could perform direct manipulation on groups of strokes. Another exciting possibility is to learn a style of drawing from the user's example strokes. With more powerful controls, it should be possible to enable an even wider range of new styles of animation.

Acknowledgments

The authors would like to thank Simon Breslav, Igor Mordatch, Ryan Schmidt for their user testing and helpful feedback. This research is supported in part by NSERC, CIFAR, CFI, and Ontario MRI.

References

[1] P. Haeberli, "Paint By Numbers: Abstract Image Representations," in *Proc. SIGGRAPH*, 1990, pp. 207–214.

[2] B. J. Meier, "Painterly Rendering for Animation," in *Proc. SIGGRAPH*, 1996, pp. 477–484.

[3] P. Litwinowicz, "Processing Images and Video for an Impressionist Effect," in *Proc. SIGGRAPH*, 1997, pp. 407–414.

[4] A. Hertzmann, "Painterly Rendering with Curved Brush Strokes of Multiple Sizes," in *Proc. SIGGRAPH*, 1998, pp. 453–460.

[5] A. Hertzmann and K. Perlin, "Painterly Rendering for Video and Interaction," in *Proc. NPAR*, 2000, pp. 7–12.

[6] J. Hays and I. Essa, "Image and Video Based Painterly Animation," in *Proc. NPAR*, 2004, pp. 113–120.

[7] A. Hertzmann, "Paint By Relaxation," in *Proc. CGI*, 2001, pp. 47–54.

[8] J. Collomosse and P. Hall, "Genetic Paint: A Search for Salient Paintings," in *Proceedings of EvoMUSART (LNCS)*, vol. 3449, 2005, pp. 437–447.

[9] B. Sabiston, "Waking Life: Making Of," 2001, dVD featurette.

[10] A. Agarwala, A. Hertzmann, D. H. Salesin, and S. M. Seitz, "Keyframe-Based Tracking for Rotoscoping and Animation," in *Proc. SIGGRAPH*, 2004, pp. 584–591.

[11] C. Liu, W. Freeman, E. Adelson, and Y. Weiss, "Human-Assisted Motion Annotation," in *Proc. CVPR*, 2008.

[12] R. D. Kalnins, L. Markosian, B. J. Meier, M. A. Kowalski, J. C. Lee, P. L. Davidson, M. Webb, J. F. Hughes, and A. Finkelstein, "WYSIWYG NPR: Drawing Strokes Directly on 3D Models," in *Proc. SIGGRAPH*, 2002, pp. 755–762.

[13] K. Odermatt and C. Springfield, "Creating 3d Painterly Environments for Disney's "Treasure Planet"," in *SIGGRAPH '02: ACM SIGGRAPH 2002 conference abstracts and applications*. New York, NY, USA: ACM, 2002, pp. 160–160.

[14] A. Bousseau, M. Kaplan, J. Thollot, and F. X. Sillion, "Interactive watercolor rendering with temporal coherence and abstraction," in *Proc. NPAR*, 2006, pp. 141–149.

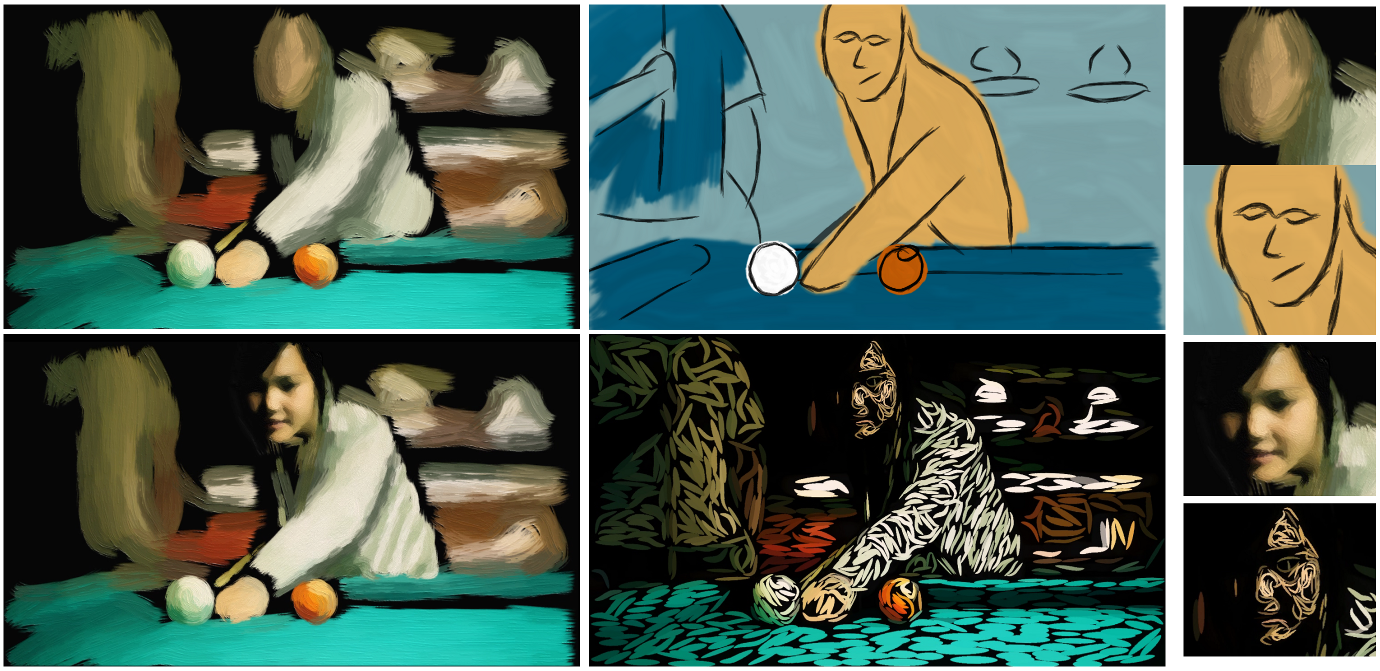


Fig. 6. Stylistic variations. Top left: A coarse, abstract style, Top middle: A sketchy style, Bottom left: Combining coarse and detailed styles, Bottom middle: A sparse black style, Right: details of each image.

[15] A. Bousseau, F. Neyret, J. Thollot, and D. Salesin, "Video Watercolorization using Bidirectional Texture Advection," *ACM Trans. Graphics*, vol. 26, no. 3, 2007.

[16] J. P. Collomosse, D. Rowntree, and P. M. Hall, "Stroke Surfaces: Temporally Coherent Artistic Animations from Video," *IEEE TVCG*, vol. 11, no. 5, pp. 540–549, 2005.

[17] D. DeCarlo and A. Santella, "Stylization and Abstraction of Photographs," in *Proc. SIGGRAPH*, 2002, pp. 769–776.

[18] B. Gooch, G. Coombe, and P. Shirley, "Artistic vision: painterly rendering using computer vision techniques," in *Proc. NPAR*, 2002, pp. 83–ff.

[19] A. Kolliopoulos, J. M. Wang, and A. Hertzmann, "Segmentation-Based 3D Artistic Rendering," in *Proc. EGSR*, 2006, pp. 361–370. [Online]. Available: http://www.dgp.toronto.edu/ alexk/segegsr.html

[20] J. Wang, Y. Xu, H.-Y. Shum, and M. F. Cohen, "Video Tooning," in *Proc. SIGGRAPH*, 2004, pp. 574–583.

[21] K. Zeng, M. Zhao, C. Xiong, and S. Zhu, "From Image Parsing to Painterly Rendering," *ACM Trans. Graph.*, vol. 29, no. 1, 2009.

[22] L. Lin, K. Zeng, H. Lv, Y. Wang, Y. Xu, and S.-C. Zhu, "Painterly Animation with Video Content Extraction," in *Proc. NPAR*, 2010.

[23] M. Kagaya, W. Brendel, Q. Deng, T. Kesterson, S. Todorovic, P. Neill, and E. Zhang, "Video Painting with Space-Time-Varying Style Parameters," *TVCG*, In Press.

[24] X. Bai, J. Wang, D. Simons, and G. Sapiro, "Video SnapCut: Robust Video Object Cutout Using Localized Classifiers," *Proc. SIGGRAPH*, vol. 28, no. 3, 2009.

[25] W. Brendel and S. Todorovic, "Video Object Segmentation by Tracking Regions," in *ICCV*, 2009.

[26] T. Fleisch, F. Rechel, and A. Santos, P.and Stork, "Constraint Stroke-Based Oversketching for 3D Curves," in *Proc. SBIM*, 2004.

[27] A. Hertzmann, "Fast Paint Texture," in *Proc. NPAR*, 2002, pp. 91–ff.

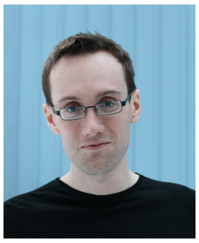
[28] J. Canny, "A Computational Approach to Edge Detection," *IEEE PAMI*, vol. 8, no. 6, pp. 679–698, 1986.

[29] H. Kang, S. Lee, and C. K. Chui, "Coherent Line Drawing," in *Proc. NPAR*, 2007, pp. 43–50.

[30] M. P. Salisbury, M. T. Wong, J. F. Hughes, and D. H. Salesin, "Orientable Textures for Image-Based Pen-and-Ink Illustration," in *Proc. SIGGRAPH*, 1997, pp. 401–406.

[31] A. Hausner, "Simulating Decorative Mosaics," in *Proc. SIGGRAPH*, 2001, pp. 573–580.

[32] M. Kass and A. Witkin, "Analyzing Oriented Patterns," *CVGIP*, vol. 37, no. 3, pp. 362–385, 1987.


Peter O'Donovan received his BSc in Computer Science from the University of Saskatchewan in 2005, and MS from the University of Toronto in 2009. He has worked in the past for Adobe Systems and is currently pursuing his PhD with Aaron Hertzmann in the areas of non-photorealistic rendering and modeling of aesthetic preferences.

Aaron Hertzmann is an Associate Professor of Computer Science at University of Toronto. He received a BA in Computer Science and Art// Art History from Rice University in 1996, and an MS and PhD in Computer Science from New York University in 1998 and 2001, respectively. In the past, he has worked at Pixar Animation Studios, University of Washington, Microsoft Research, Mitsubishi Electric Research Lab, Interval Research Corporation and NEC Research Institute. He is a fellow of the Canadian Institute for Advanced Research, and an associate editor for ACM Transactions on Graphics. His awards include the MIT TR100 (2004), an Ontario Early Researcher Award (2005), a Sloan Foundation Fellowship (2006), a Microsoft New Faculty Fellowship (2006), the CACS/AIC Outstanding Young CS Researcher Award (2010), and the Steacie Prize for Natural Sciences (2010).

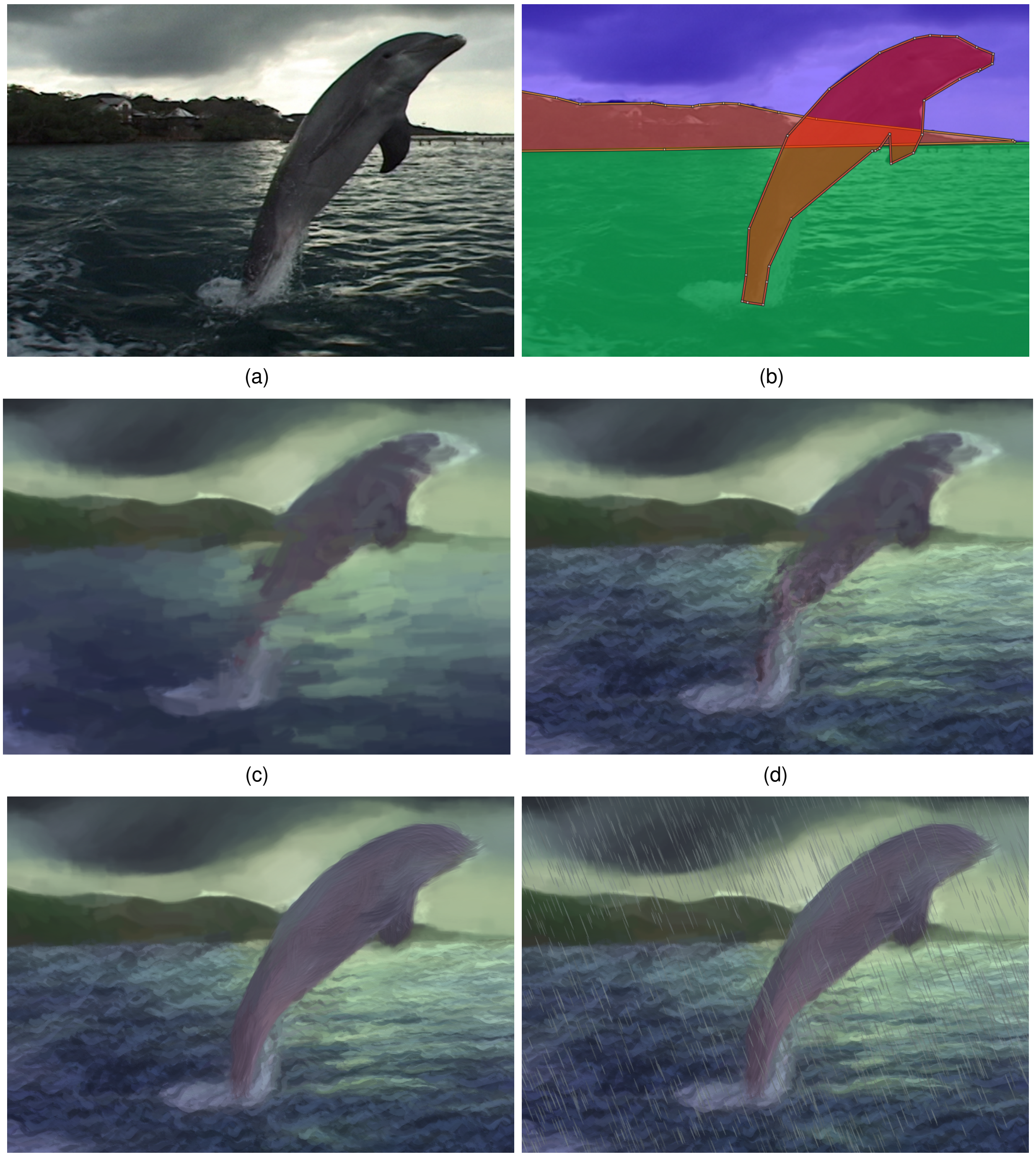


Fig. 7. Stroke layering a) Source, b) Rotoscoping, c) Initial coarse painting over background, d) Water refined with wavy strokes, e) Body painted with large strokes, f) Rain layers

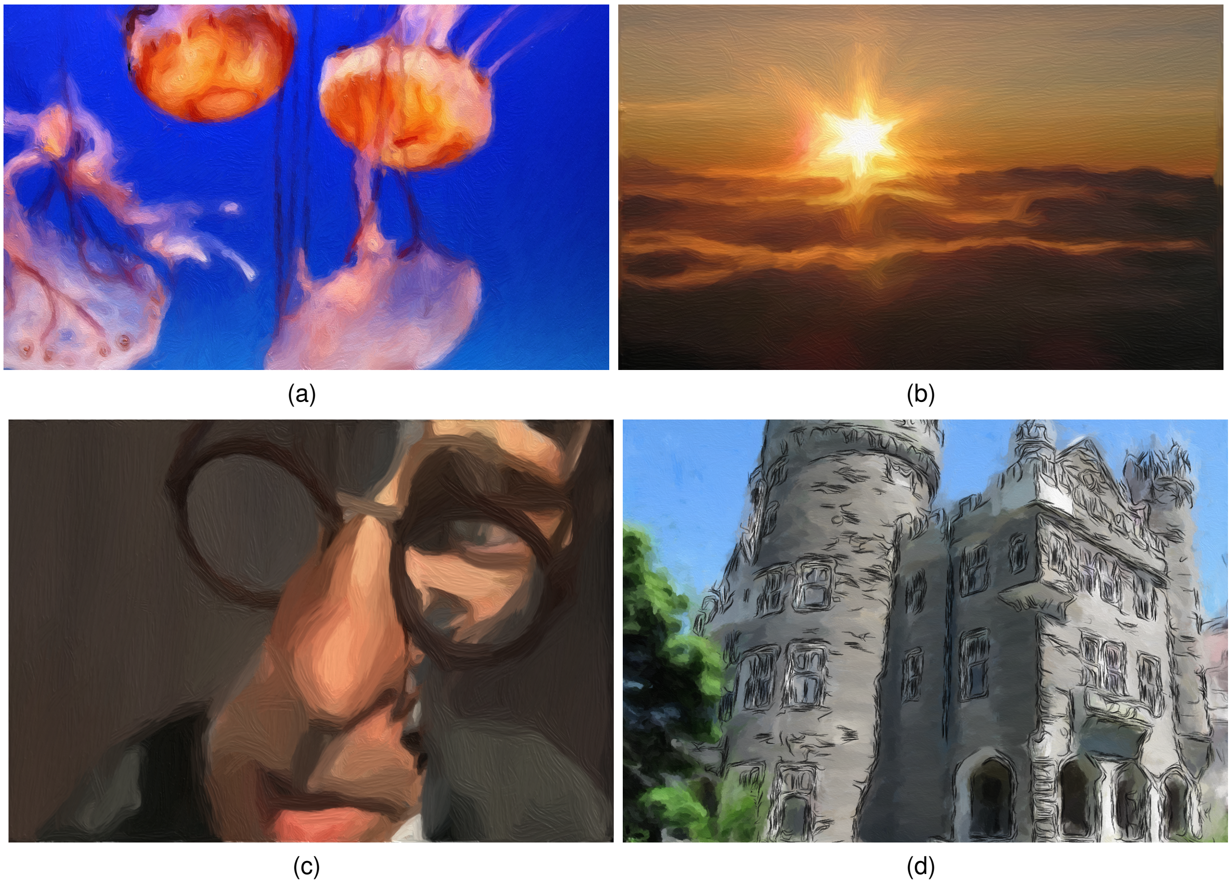


Fig. 8. More results. a) Jellyfish sequence, b) Sunset sequence, c) Ryan sequence (Ryan ©2004 Copperheart Entertainment and the National Film Board of Canada, dir. Chris Landreth) , d) Castle sequence