Stylizing Animation By Example

Pierre Bénard[1,2] Forrester Cole[1] Michael Kass[1] Igor Mordatch[1,3] James Hegarty[1,4]
Martin Sebastian Senn[1] Kurt Fleischer[1] Davide Pesare[1] Katherine Breeden[1,4]

[1]Pixar Animation Studios, [2]University of Toronto, [3]University of Washington, [4]Stanford University

Figure 1: *Frame 20 (right) to 34 (left) – top row: stylized animation; bottom row: input shaded images. The two extreme frames are keyframes painted by an artist. Our algorithm synthesized the in-between frames. Note the important variations in terms of shape and appearance (texture and color) during this sequence. The accompanying video illustrates the temporal behavior.*

Abstract

Skilled artists, using traditional media or modern computer painting tools, can create a variety of expressive styles that are very appealing in still images, but have been unsuitable for animation. The key difficulty is that existing techniques lack adequate temporal coherence to animate these styles effectively. Here we augment the range of practical animation styles by extending the guided texture synthesis method of *Image Analogies* [Hertzmann et al. 2001] to create temporally coherent animation sequences. To make the method art directable, we allow artists to paint portions of keyframes that are used as constraints. The in-betweens calculated by our method maintain stylistic continuity and yet change no more than necessary over time.

CR Categories: I.3.6 [Computing Methodologies]: Computer Graphics—Methodologies and Techniques; I.4.9 [Computing Methodologies]: Image Processing and Computer Vision—Applications

Keywords: keyframe, texture synthesis, temporal coherence, non-photorealistic rendering

Links: DL PDF WEB VIDEO

1 Introduction

For many artistic purposes, hand-painted imagery can produce a warmth and range of styles that is difficult to achieve with 3D computer rendering. To date, however, most handpainted styles remain unsuitable for story-telling and character animation because of the difficulty of maintaining temporal coherence. Fine-scale texture detail is a particularly important feature of many handpainted styles, yet it is extremely time consuming – when even possible – for an artist to ensure by hand that fine-scale texture details change smoothly enough from frame to frame to prevent flickering, popping or other unpleasant visual artifacts.

The existing literature includes some algorithmic techniques that are able to create coherent, detailed, stylized animations for very particular looks (see Bénard et al. [2011] for a survey). However, the existing algorithms are not only limited to a narrow range of styles, but also provide little direct control over the end result. These limitations pose severe difficulties for animation production when an art director is trying to achieve a very specific look. It may be hard to know if a desired look is close to the range of any existing algorithm, and difficult to modify the algorithms to move towards a desired look. High-quality character-based story telling requires a level of flexibility and control that these methods do not provide.

A more direct approach is to specify a desired look with a set of visual examples, allowing artists to communicate their desires by using tools with which they already have considerable skill: paintings or drawings done with traditional or digital media. Hertzmann et al. [2001] have shown with their *Image Analogies* work that an example-based approach can do a good job of generalization for the creation of stylized still images. Their method uses a pair of images to define a style transformation, and then applies this transformation to a new input image using guided texture synthesis (Figure 2). Here we seek to extend this approach to animation. Our

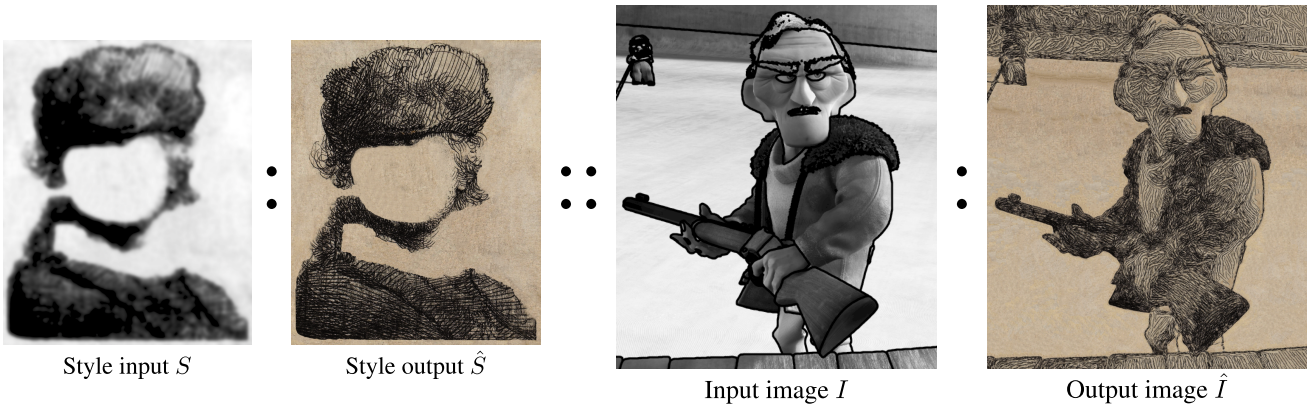


Figure 2: *Image Analogy setup. Given a style input S, a style output $\hat{S}$, and a new input image I, the image analogies approach corresponds colors in S and I to create a stylized output $\hat{I}$. Here, the input I has been rendered to emphasize its silhouettes, encouraging dark outlines in the output $\hat{I}$.*

goal is to create an example-based stylization method that supports a broad range of styles, provides artists with direct control over the result, and achieves the temporal coherence necessary for pleasing painterly animation.

To take advantage of highly-developed tools for creating and editing character motion and lighting with computers, we begin with lit 3D CG animation as input. Our Temporally Coherent Image Analogies algorithm (TCIA) then transforms the input animation into a sequence of stylized images with both a characteristic 2D look and strong temporal continuity. In order to deduce how the image texture should move, we render image-space velocity fields from our input animation sequence. We add goal functions to the Image Analogies algorithm that penalize unwanted changes over time, while taking into consideration both occlusion and dis-occlusion events. We solve the underlying optimization problem using a coarse-to-fine version of *PatchMatch* [Barnes et al. 2009] on a GPU. Our implementation can compute film-resolution (1080p) animations at the rate of 10 to 12 minutes per final frame. Full details of the temporally coherent Image Analogies algorithm (TCIA) are presented in Section 3. Besides the temporal extensions, we also extend the image analogies method to handle rotations of the textures, raster lines stylization, and control of the overall texture statistics. These additions allow synthesis of patterns along a vector field to better depict shape (e.g., hatching), generation of textured strokes along object contours without extracting vector lines, and avoidance of obvious texture repetition.

In addition to a generic style transformation, we allow the artist to paint exactly what is desired at important frames (Figure 1). When full control is applied, this amounts to a keyframe painting system. The modifications to TCIA necessary to provide interpolation of the keyframe constraints are presented in section 4. We have found that good artistic control can be achieved by painting an average of one out of every 10 to 20 frames of animation. By some standards, this is a lot of painting, but it is a small fraction of the painting required for traditional 2D hand-painted methods.

Our main contribution is two-fold. First, we extend image analogies to animation, achieving temporal continuity while taking account of occlusion and disocclusion. Second, we allow art direction by modifying the approach to interpolate hand-painted elements at keyframes. The result is a practical production method that combines the strengths of existing tools for creating and editing motion and lighting with the warmth, feeling and artistic control of hand-painted visual styles.

2 Related Work

Our approach is at the intersection of two research fields: non-photorealistic rendering and texture synthesis. Since they are both wide fields, we will only review the work most related to our approach.

2.1 Temporally coherent NPR

Many specialized techniques have been proposed to address the temporal coherence problems that arises when stylizing animations [Bénard et al. 2011]. A common approach is to focus on a specific style of animation and design a method to exploit the nature of that style: for example, shape abstraction in watercolor (e.g., [Bousseau et al. 2007]), or the hierarchy of strokes in hatching (e.g., [Praun et al. 2001; Kalnins et al. 2002]). This approach can be very effective for the specific target style, but the assumptions usually do not translate far beyond the target.

A second approach is to adopt a general, temporally-coherent noise function, and transform it to yield a visually pleasing stylization result [Bénard et al. 2009; Bénard et al. 2010; Kass and Pesare 2011]. While noise functions provide a broader range of styles than tailored algorithms, this range is still limited and those methods are difficult to art direct.

Stroke-based painting and animation systems [Daniels 1999; Agarwala et al. 2004; Schmid et al. 2011; Whited et al. 2012] combine good temporal coherence with good directability, since the artist can simply paint the strokes they desire. This flexibility, however, has a major drawback: the artist *must* paint each stroke, even in uninteresting areas. In the context of video stylization, semi-automatic approaches have been proposed to overcome this limitation [Lin et al. 2010; Kagaya et al. 2011; O'Donovan and Hertzmann 2011]. Starting from sparse user inputs (assisted segmentation and labeling, guiding strokes, spatially-varying style parameters), they automatically propagate these constraints in space and time while optimizing for temporal coherence. However, the range of styles of these approaches is limited to those that can be depicted with discrete strokes. For instance, effects created by raster-based filters (e.g., [Winnemöller et al. 2006]) cannot be easily represented as strokes. Additionally, we are not aware of any example-based video stylization methods that support keyframes (see Kyprianidis et al. [2013] for a survey).

Our approach supports a broader range of styles than previous methods by using example images to define the style. By styliz-

ing animations in 2D image-space while relying on accurate 3D information, our method gains the best of both worlds: it natively handles foreshortening and zooming, while ensuring good motion coherence and temporal continuity. Unlike most of the above techniques, our method produces temporal coherence through optimization rather than by construction.

Like Bousseau et al. [2007], we make use of both forward-time and backwards-time advection to create temporal coherence. However, instead of using a time-varying blend of the two advections to compute the final result, we embed our advection in the optimization process. Since optimization only selects existing colors from the style, our algorithm produces a warp rather than a blend. In the case of a pen-and-ink style containing only blacks and whites, for example, our output contains only blacks and whites, while any blending approach will create intermediate gray values, perceptible as ghosting artifacts.

2.2 Texture Synthesis

The Image Analogies work lives within a wider and well-developed field of texture synthesis. Wei et al. [2009] provide an excellent survey of the available methods. In principle, any of the methods in the literature could provide the basis for a by-example stylization system. We use a scheme based on PatchMatch [Barnes et al. 2009] because of its simplicity, good performance, parallelizability, and suitability for adaptation to our needs. It is possible that texture synthesis algorithms based on PCA vectors (e.g., [Lefebvre and Hoppe 2006]), global optimization (e.g., [Kwatra et al. 2005]) or explicit patch transfer (e.g., [Efros and Freeman 2001; Lasram and Lefebvre 2012]) could further improve our results.

Synthesis for animation. A number of researchers have investigated using texture synthesis to create temporal image sequences, but none have specifically addressed the problems of using texture synthesis to author high-quality non-photorealistic renderings of character animation.

Kulla et al. [2003] extended the texture transfer algorithm of Efros and Freeman [2001] to animations by using *Image Quilting* to shade animated 3D scenes with hand-painted color and texture gradients. They propose three simple techniques to increase temporal coherence: limited 2D re-synthesis, view-aligned 3D texturing, and view-dependent interpolation. However, none of these is fully satisfactory, causing either blending, sliding or 3D distortion artifacts.

Cao et al. [2011] use a comparable approach to stylize videos by relying on optical flow. Hashimoto et al. [2003] and Haro [2003] applied the Image Analogies method to low-resolution video sequences, using motion estimation to increase temporal coherence. The results of these methods also exhibit sliding and popping artifacts, due to inaccurate motion estimation, lack of explicit handling of occlusion, and low overall resolution.

A family of work in flow-guided texture synthesis combines texture synthesis with advection, but addresses different problems from ours. Bargteil et al. [2006] and Kwatra et al. [2007] generate temporally coherent textures on the surface of liquids to create surface details such as waves or bubbles. Han et al. [2006] synthesize animated patterns on the surface of 3D meshes to help visualize flow fields. Wei and Levoy [2000] synthesize video textures. Unlike our work, none of these applications addresses occlusion during the texture synthesis process.

Bonneel et al. [2010] use guided texture synthesis to create a system that targets near-interactive pre-visualization of natural environments. Their system provides the quality needed for previsualization using only approximate geometry, while operating within the constraints of near real-time performance. The application of character animation, however, demands a much higher-quality result with better temporal coherence, but allows for offline computation.

Image and texture interpolation. Our keyframe in-betweening approach tackles the problem of finding smooth interpolation paths between two textured images. In a more specific context, methods have been proposed to metamorphose or morph one static texture into another. They can be classified into two main families: (1) approaches that compute a global warping field between the two textures and linearly blend them after alignment [Liu et al. 2002; Matusik et al. 2005; Lai and Wu 2007] or compute both simultaneously [Kabul et al. 2011]; (2) methods that locally warp a feature map and use it for guided pixel- or patch-based texture synthesis [Zhang et al. 2003; Ray et al. 2009; Ruiters et al. 2010].

The latter approaches have the major benefits of limiting blending artifacts and allowing topology changes. Gradient domain blending can be used to further smooth patch boundaries and avoid explicit feature map computation [Darabi et al. 2012]. Extending this idea, Shechtman et al.[2010] generate morphing sequences between pairs of static images by optimization. Our method takes inspiration from these techniques, while dealing with additional challenges because painted keyframes encode both texture and structure information and relate to each other through complex 3D motion paths with occlusions.

3 Temporally Coherent Image Analogies

The original Image Analogies method transforms an input image I into a stylized version $\hat{I}$ using an example-based style transformation defined by an style input image S and a corresponding style output image $\hat{S}$ (Figure 2). The transformation is defined by a mapping, or *offset field*, $\mathbf{M}$ from each point $\mathbf{p}$ in $\hat{I}$ to a corresponding point $\mathbf{M}(\mathbf{p})$ in $\hat{S}$, such that $\hat{I}(\mathbf{p}) = \hat{S}(\mathbf{M}(\mathbf{p}))$. A good mapping $\mathbf{M}$ minimizes a goal function $G(\mathbf{p})$, where G measures how closely the analogy between S and $\hat{S}$ is transferred to I and $\hat{I}$. In the language of Wei et al. [2009], this is a "pixel-based" texture synthesis method where the fundamental operation is to change the mapping at an individual output pixel.

We extend this approach to an input animation I_t and an output animation $\hat{I}_t$ by optimizing for a sequence of offset fields $\mathbf{M}_t$. The motion of the animation is defined using input velocity and orientation fields (Figure 3). We encourage temporal coherence between the frames of $\hat{I}_t$ by adding a temporal coherence term to the goal function. To produce the sequence of offset fields $\mathbf{M}_t$, we optimize each frame with a method based on PatchMatch [Barnes et al. 2009], then advect the results of the previous frame to the next using the image velocity fields.

3.1 Inputs

There are two kinds of inputs to our algorithm: (1) a CG animation rendered into buffers containing for each frame t input shading information I_t, orientations $\mathcal{O}(I_t)$, and velocities V_t^+ and V_t^-; (2) static style buffers with input S, output $\hat{S}$ and orientations $\mathcal{O}(S)$.

Orientations. In order to allow the style to be used at arbitrary orientations, we use two orientation fields: $\mathcal{O}(S) \in [0, \pi]$ defined in the coordinate system of the style, and $\mathcal{O}(I_t) \in [0, \pi]$ defined in the coordinate system of the input images. Similarly in spirit to Lee et al. [2011], our algorithm performs rotations to compensate for the different local orientations.

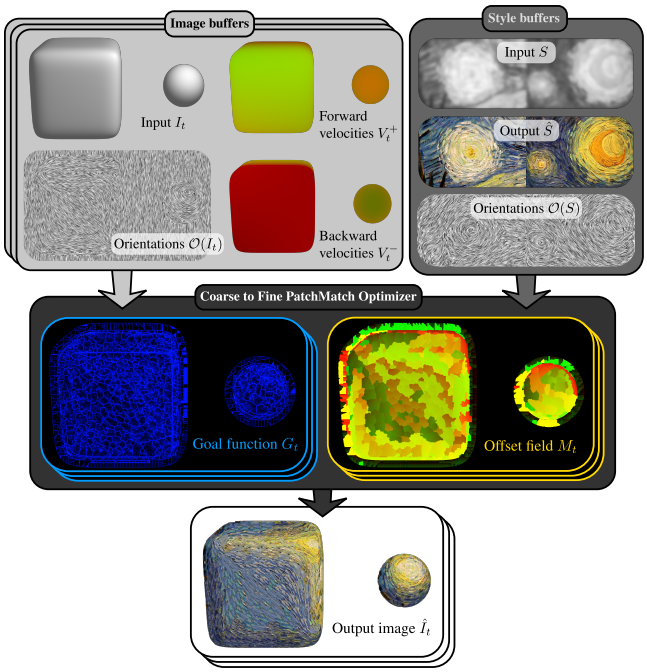


Figure 3: *Synthesis overview. The input to the system consists of an animation with input I_t, orientations $\mathcal{O}(I_t)$, velocity fields forward and backward in time (V_t^+ and V_t^-), and a static style with input S, output $\hat{S}$, and orientation $\mathcal{O}(S)$. The optimization scheme matches patches of the style with the input through an offset field M_t, while minimizing the goal function G_t. The output animation $\hat{I}_t$ is found by looking up pixels in the style output $\hat{S}$ using the offset field.*

There is an inherent ambiguity in the orientation of patterns that resemble lines or stripes because a rotation by an angle of π will leave the dominant orientation unchanged. As a result, the appropriate rotation is also ambiguous. To prevent a visible branch cut we always use the smallest angle of rotation $\phi(\mathbf{p}_1, \mathbf{p}_2)$ that brings the two directions $\mathcal{O}(I_t)(\mathbf{p}_1)$ and $\mathcal{O}(S)(\mathbf{p}_2)$ into alignment.

The orientation fields $\mathcal{O}(I_t)$ and $\mathcal{O}(S)$ can either be authored by a user for complete control (e.g., by rendering a vector field defined on the 3D surface), or computed automatically. Except where otherwise explicitly stated, we compute these orientation fields automatically as the by-pixel *structure tensor* of the grayscale image [Granlund and Knutsson 1995]. Structure tensors have the benefit over scalar angles of allowing meaningful interpolation and averaging.

Velocity and Occlusion. To stylize animations, we need to know how the input scene moves and when elements of the scene are occluded or disoccluded. To do so, we render two velocity fields for the input animation sequence: $V_t^+(\mathbf{p})$ and $V_t^-(\mathbf{p})$. The field $V_t^+(\mathbf{p})$ is a forward finite difference of image-space velocity. A point visible at $\mathbf{p}$ in frame t moves to $\mathbf{p} + V_t^+(\mathbf{p})$ in frame $t+1$. Similarly, $V_t^-(\mathbf{p})$ is a backward finite difference. A point visible at $\mathbf{p}$ in frame t came from $\mathbf{p} - V_t^-(\mathbf{p})$ in frame $t-1$.

$V_t^+(\mathbf{p})$ tells us where $\mathbf{p}$ will project in the next frame, and $V_t^-(\mathbf{p})$ tells us where it projected in the previous frame. Combined, they also tell us whether or not $\mathbf{p}$ was occluded in the previous frame $t-1$. In absence of occlusion, following the velocity field backward and then forward, we should come back to the starting position, i.e., $(\mathbf{p} - V_t^-(\mathbf{p})) + V_{t-1}^+(\mathbf{p} - V_t^-(\mathbf{p})) = \mathbf{p}$. From this condition, we

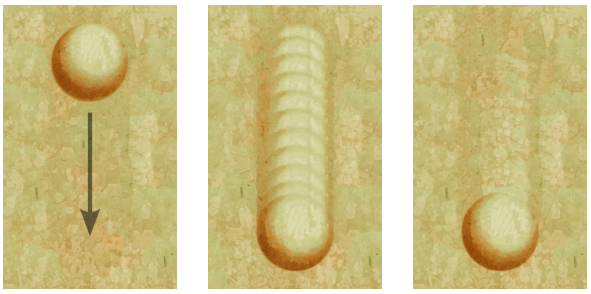


Figure 4: *Effect of occlusion term χ. When a foreground object moves across a static background (left), naively enforcing temporal coherence can leave trails as the background is disoccluded (middle). Including occlusion information in the advection and temporal coherence terms allows the background to resynthesize at each frame and removes the trails (right).*

define the following occlusion function which is 1 if $\mathbf{p}$ was not occluded at frame $t-1$:

$$\chi^-(\mathbf{p}, t) = \begin{cases} 1 & \text{if } ||V_{t-1}^+(\mathbf{p} - V_t^-(\mathbf{p})) - V_t^-(\mathbf{p})||^2 < \epsilon \\ 0 & \text{otherwise} \end{cases}$$

Similarly we define $\chi^+(\mathbf{p}, t)$ using the backward velocities at frame $t+1$:

$$\chi^+(\mathbf{p}, t) = \begin{cases} 1 & \text{if } ||V_{t+1}^-(\mathbf{p} + V_t^+(\mathbf{p})) - V_t^+(\mathbf{p})||^2 < \epsilon \\ 0 & \text{otherwise} \end{cases}$$

The velocities are rasterized on the pixel grid, potentially introducing round-off error; we compare them with a certain tolerance $\epsilon = 2$ in our implementation. The effect of the occlusion term is demonstrated in Figure 4.

3.2 Goal Function

Following Hertzmann et al. [2001], our algorithm is based on repeated evaluations of a goal function for the offset at individual pixels. When new offsets are found that improve the goal function, those offsets are updated. The goal function is a property of a local neighborhood and has the form:

$$G_t(\mathbf{p}) = \sum_{\Delta\mathbf{p} \in \Omega} w(\Delta\mathbf{p}) g(\mathbf{p}, \Delta\mathbf{p}) \tag{1}$$

where Ω represent the set of vectors from $\mathbf{p}$ to its neighbors. We typically use 9×9 pixel square neighborhoods. The weights $w(\Delta\mathbf{p})$ give more importance to samples near the center of the neighborhood. We use weights with Gaussian fall-off (variance $\sigma = 3$ in our experiments).

$$w(\Delta\mathbf{p}) = e^{\frac{-|\Delta\mathbf{p}|^2}{2\sigma^2}}$$

Our overall goal function is constructed as the weighted sum of four main contributing goal functions and two optional terms, each representing a distinct aim of the synthesis.

$$\begin{aligned} g(\mathbf{p}, \Delta\mathbf{p}) = {} & w_{out} g_{out}(\mathbf{p}, \Delta\mathbf{p}) + w_{in} g_{in}(\mathbf{p}, \Delta\mathbf{p}) \\ & + w_{sc} g_{sc}(\mathbf{p}, \Delta\mathbf{p}) + w_{tc} g_{tc}(\mathbf{p}) \\ & + w_h g_h(\mathbf{p}) + w_{dt} g_{dt}(\mathbf{p}) \end{aligned} \tag{2}$$

The weights w_{out}, w_{in}, w_{sc}, w_{tc}, w_h and w_{dt} allow the user to balance the relative importance of the different goals; the values used for generating our results are provided in the supplemental accompanying video. The three first terms match the goal proposed by Hertzmann et al. [2001], modified to allow per-pixel orientation.

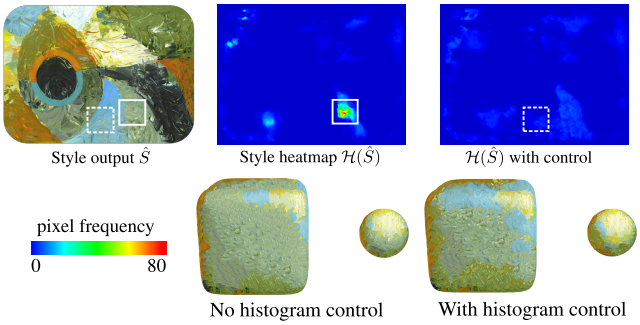


Figure 5: *Histogram term comparison. The optimization may use small patches of the style many times, producing an undesirable repeating pattern and a hotspot in the style histogram heatmap (solid box). With a penalty for histogram hotspots, the optimization is encouraged to use more of the style (e.g., dashed box), producing a more varied output.*

Output term. The first goal of our texture synthesis is to make each local neighborhood of the output image $\hat{I}_t$ near $\mathbf{p}$ "look like" a corresponding neighborhood of the style output $\hat{S}$ near $\mathbf{M}_t(\mathbf{p})$. In order to compare the neighborhoods of $\hat{I}_t$ and $\hat{S}$, we have to take into account the rotation $\phi(\mathbf{p}, \mathbf{M}_t(\mathbf{p}))$ between the local coordinates of the input and the style images. Let Rot_ϕ denote the rotation of a vector by angle ϕ. Then the point in the style corresponding to $\mathbf{p} + \Delta\mathbf{p}$ in the output is given by:

$$\mathbf{C}_t(\mathbf{p}, \Delta\mathbf{p}) = \mathbf{M}_t(\mathbf{p}) + \mathrm{Rot}_\phi(\Delta\mathbf{p})$$

Now we can write our first goal as:

$$g_{out} = |\hat{I}_t(\mathbf{p} + \Delta\mathbf{p}) - \hat{S}(\mathbf{C}_t(\mathbf{p}, \Delta\mathbf{p}))|^2 \quad (3)$$

Note that Eq. 3 is implicitly summed over the patch neighborhood $\Delta\mathbf{p}$ due to the summation in the global goal function (Eq. 1). We experimented with comparison in RGB and Lab^* color spaces. The Lab^* space can significantly improve the results in cases where adjacent regions are similar in RGB space (e.g., the skater's face and suit, Figure 1). However, we find that when these regions are separated into layers (Section 4), RGB calculations are sufficient.

Input term. The second goal of our synthesis is for the offsets to be chosen so that each neighborhood of the input image I will be matched to a neighborhood of the style input S that looks similar. We achieve this with the following goal function:

$$g_{in} = |I_t(\mathbf{p} + \Delta\mathbf{p}) - S(\mathbf{C}_t(\mathbf{p}, \Delta\mathbf{p}))|^2 \quad (4)$$

Spatial coherence term. The third goal is for the mapping $\mathbf{p} \mapsto M_t(\mathbf{p})$ to be spatially as continuous as possible. Ashikhmin [2001] showed spatial continuity to be an important component of pixel-based texture synthesis techniques. To promote continuity, our third component of the energy is:

$$g_{sc} = \min(|\mathbf{M}_t(\mathbf{p} + \Delta\mathbf{p}) - \mathbf{C}_t(\mathbf{p}, \Delta\mathbf{p})|^2, r_{max}) \quad (5)$$

This term is the squared distance between the actual offset at $\mathbf{p} + \Delta\mathbf{p}$ and the offset one would expect from $\mathbf{p}$ if the offset field was a simple rotation of the style output. We bound the error to measure how much of the mapping is continuous; the actual distance in the style images between discontinuous offsets is unimportant. Since our use of sizeable neighborhoods in g_{out} and g_{in} already produces spatial coherence in the offsets, a small value of w_{sc} has been sufficient for our purposes.

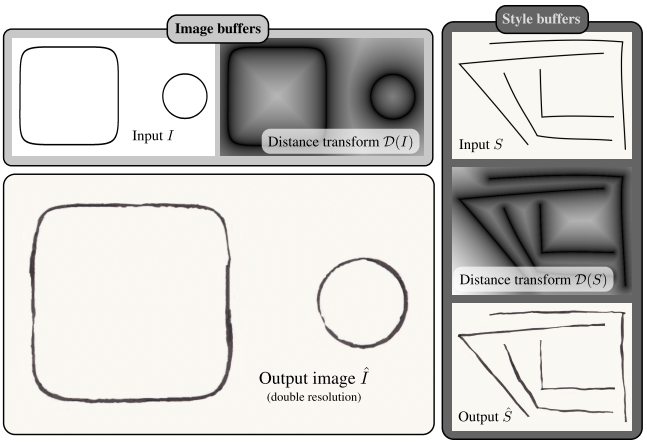


Figure 6: *Line stylization. Synthesis is guided by the distance transform of input lines $\mathcal{D}(S)$ and $\mathcal{D}(I)$ as well as the orientations (not shown).*

Temporal coherence term. Our fourth goal is to prevent unnecessary or sudden changes in time, i.e., to minimize color variations of the output images along valid motion paths (non-occluded positions). We use alternatively two versions of this term (Section 3.3), one looking backward in time g^-_{tc} and the other forward g^+_{tc}.

$$g^-_{tc} = \chi^-(\mathbf{p}, t)|\hat{I}_t(\mathbf{p}) - \hat{I}_{t-1}(\mathbf{p} - V^-_t(\mathbf{p}))|^2 \quad (6)$$

$$g^+_{tc} = \chi^+(\mathbf{p}, t)|\hat{I}_t(\mathbf{p}) - \hat{I}_{t+1}(\mathbf{p} + V^+_t(\mathbf{p}))|^2 \quad (7)$$

Histogram term. To penalize repeated patterns, we introduce a histogram term which encourages diversity in the output image by penalizing repetitions of the same offsets. In the spirit of Chen and Wang [2009], we build the histogram of the offsets field $\mathcal{H}(M_t)$ by counting the number of occurrences of each offset. This is equivalent to a map of the frequency of every pixel of $\hat{S}$ in $\hat{I}_t$. We can then write the following goal:

$$g_h = h_s \max(0, \mathcal{H}(\mathbf{M}_t)(\mathbf{p}) - h_\tau) \quad (8)$$

The parameters h_τ and h_s allow the user to separately control the minimum number of repetition after which this penalty starts to apply and its strength once the histogram value exceeds this threshold. To further encourage spatial variation, we blur the histogram with a small Gaussian kernel. Figure 5 illustrates the effect of this term on a painterly style. Unlike PatchMatch *completeness* term, which ensures that the synthesized image contains as much visual information from the exemplar as possible, this histogram term imposes a less restrictive constraint that prevents excessive repetition.

Distance transform term. Starting from raster lines extracted in image-space, painterly stroke styles can be produced by including the distance from input line pixels in the optimization. We compute the distance transform of black pixels in the input image $\mathcal{D}(I_t)$ and style input $\mathcal{D}(S)$ and define a goal that encourages them to match.

$$g_{dt} = |\min(\mathcal{D}(I_t)(\mathbf{p}) - \mathcal{D}(S)(\mathbf{M}_t(\mathbf{p})), d_{max})|^2 \quad (9)$$

We bound the distance transform by d_{max} (10 pixels at the highest resolution in our implementation) to roll back to regular synthesis when $\mathbf{p}$ is far enough from the lines. Figure 6 shows the result of stroke synthesis on a simple scene. A major benefit of this approach is that it does not require a temporally coherent vector representation of the lines. Constructing such a representation is a difficult problem, especially for lines extracted in image-space (e.g., [Bénard et al. 2012]).

3.3 Solution Method

For an animation sequence with T frames and N pixels per frame, the solution space of offset fields $\mathbf{M}_t$ has $O(TN)$ dimensions (roughly 10^9 for our examples). We can only afford to explore a small subset of this space, so we have adopted several heuristics to find good solutions. We optimize the animation frames in a forward-backward, coarse-to-fine order (Figure 7), sweeping through the animation several times for each resolution level. For each frame and resolution level, we apply several iterations of parallel PatchMatch [Barnes et al. 2009] to improve the offset field $\mathbf{M}_t$. Together, these methods guide the optimization to a pleasing result with a small number of iterations.

Forward-Backward Coarse-to-Fine. Coarse-to-fine spatial optimization is a standard technique in texture synthesis [Wei et al. 2009], used to improve both the rate of convergence and quality of the final result. We adopt coarse-to-fine optimization as the outermost loop of the algorithm: the entire animation is optimized for each resolution level, then upsampled to produce the starting guess for the next finer resolution.

At each resolution level, we sweep through the time dimension sequentially: we optimize frame t fully, then advect the solution to frame $t+1$ to initialize the next optimization. When the optimization reaches the last frame, the sweep reverses and reoptimizes frame t, incorporating the solution of frame $t-1$. Sweeping forwards and backwards through time allows the optimization to find temporally coherent results for sequences of arbitrary length. The results in this paper use four levels of resolution and one pair of forward-backward sweeps.

To take into account the solution of the forward sweep during the backward pass, we initialize each pixel of the offset field at frame t by randomly choosing between the previous solution at frame t and the advected result from frame $t+1$ (Figure 7 second row). This random merge is also applied during upsampling to initialize the first sweep of level l from the previous frame at level l and the result at level $l-1$ (Figure 7 third row). The random merge is well suited to PatchMatch optimization, because it effectively makes both the forward and backward (resp. coarse and fine) solutions available to the propagation stage. For every pixel in the merged field, it is likely to find a neighboring pixel from the forward (resp. coarse) solution and a neighboring pixel from the backward (resp. fine) solution. If a pixel is occluded in either of the sources for merging (last sweep or last level), the unoccluded source is used.

Parallel PatchMatch. The core of the optimization at each frame is parallel PatchMatch [Barnes et al. 2009]. PatchMatch is a fast algorithm for computing dense approximate nearest neighbor correspondences of image neighborhoods for arbitrary distance functions. It is a randomized algorithm that relies on iteratively improving a set of correspondences until convergence by alternating propagation and random search steps. It has been shown to perform very well in comparison with other algorithms [Barnes et al. 2010].

In the spirit of a Jacobi iteration, the parallel version of PatchMatch independently computes updates for each nearest-neighbor correspondence in parallel, without regard for the impact of one update on another. More specifically, for each pixel in parallel, Eq. 1 is evaluated for a set of candidate offsets, ignoring interactions between overlapping neighborhoods. The best candidates are then applied together and the process repeated. While a single pass of the parallel update is less effective than a serial update, the speed-up due to parallelism on GPUs more than compensates, and makes this an attractive scheme. Although this does not offer theoretical guarantees of convergence, we observed convergence to a visually pleasing solution after 3 or 4 full iterations in all our experiments.

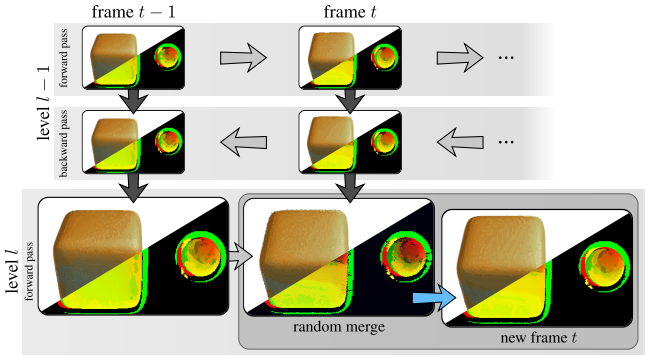
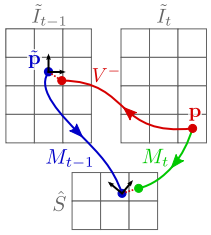


Figure 7: *Forward and backward sweeps. To create temporal coherence, multiple forward and backward sweeps are made through the entire animation at each level of coarse-to-fine synthesis. At each sweep, the last offset field at frame t is randomly merged with the advected result from frame $t-1$ (forward sweeps) or $t+1$ (backward). The randomly merged result forms the starting guess for the next round of optimization.*

Advecting and interpolating offsets. In order to initialize the solution at frame t, we advect the solution from frame $t-1$ as follows. For each pixel $\mathbf{p}$, we consider whether or not it was visible in the previous frame by consulting $\chi^-(\mathbf{p},t)$. If it was occluded, a random offset is used if we are on the coarsest level, or else the upsampled result is used. If it was visible, we advect the offset of the previous frame by looking up $\mathbf{M}_{t-1}(\mathbf{p}-V_t^-)$ to compute an estimate of its prior match. During the backwards pass, we compute the prior match estimate based on $\mathbf{M}_{t+1}(\mathbf{p}+V_t^+)$ and $\chi^+(\mathbf{p},t)$.

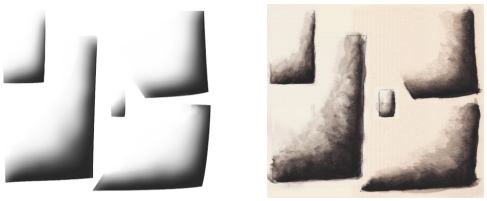


When looking up an offset during advection, we avoid interpolating neighboring offsets as interpolation across discontinuities in the offset field produces spurious results. Instead, we first find the nearest-neighbor sample $\tilde{\mathbf{p}}$ to $\mathbf{p}-V_t^-$, then extrapolate using the orientation at $\tilde{\mathbf{p}}$ and the style orientation at $\mathbf{M}_{t-1}(\tilde{\mathbf{p}})$ (inset figure). This lookup scheme provides sub-pixel accuracy while avoiding interpolation of offset values. Sub-pixel accuracy in this lookup is important when optimizing at coarse scales, where even moderate movements may be smaller than a pixel. The same extrapolation scheme is used when upsampling the coarse solutions to provide meaningful results around offset field discontinuities.

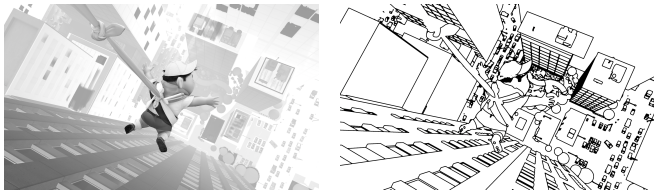
3.4 Results

Figure 8 shows an example result for both shading and lines. The style output $\hat{S}$ is respectively a hand-painted watercolor wash (Figure 8(a)) and irregular strokes (Figure 6) produced by an artist on paper and then scanned. The style input S was created by registering computer-generated gradients and lines in a drawing program. The input sequences are a rendering with conventional shading, and lines extracted from object boundaries. Watercolor texture was applied to the shading, and strokes applied to the lines using our method. Finally, the texture and lines were composited with solid colors from the original input (Figure 8(d)).

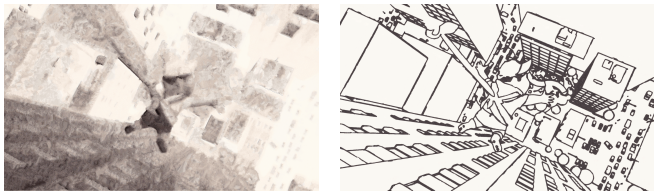
For still images, our algorithm is similar to the original Image Analogies work. The most salient differences are our use of orientation fields and our use of parallel PatchMatch for optimization. The use of orientations extends the artistic range of our method to include stylized outlines such as in Figure 6 and 8, while Hertzmann et al. [2001] provide only textured examples. By using parallel PatchMatch, we were able to implement our algorithm in CUDA



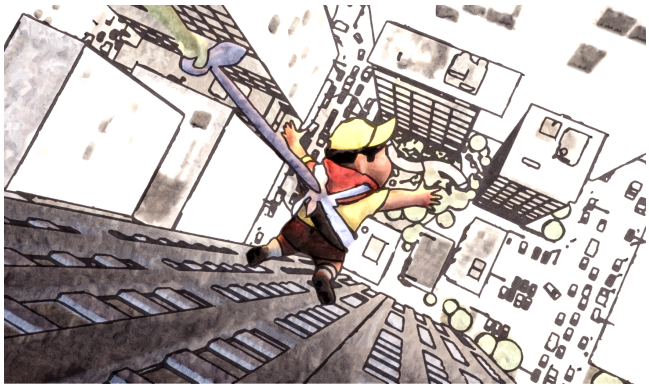
(a) Style input and output (line style as in Figure 6)



(b) Input images: shading and lines



(c) Output images: texture and strokes



(d) Composite with color overlay

Figure 8: *Watercolor style with lines. Grayscale images are stylized with a watercolor style; the edge map is stylized using strokes. The image is composited with colors from the original rendering.*

and achieve running times for still images (1920 × 1080 pixels) of one and a half minutes or less on a NVIDIA Quadro 5000 (Fermi) GPU. Note that computation time scales with image resolution and not scene complexity.

Most notably, TCIA provides strong temporal continuity during animation, especially compared to independent frame-by-frame stylization. Please see the two accompanying videos to fully appreciate this quality for various styles on simple and complex animations.

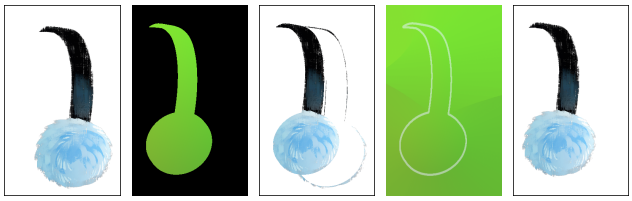
4 Keyframe Constraints

The analogy metaphor is a powerful, versatile and intuitive way to specify a style, but it lacks the degree of control and art direction that a direct interaction system can provide. Direct control is particularly important for stylizing animated characters, and TCIA can easily be extended to allow direct control by incorporating painted keyframes as hard constraints.

(a) Input image (b) Full keyframe (c) Layer decomposition

(d) Closeup (overdraw and partial transparency)

Figure 9: *A painted keyframe and its decomposition into five layers: suit, skin (face and hands), eyes, muffs and skate. This decomposition allows overdraw and partial transparency without compromising the quality of the advection.*



(a) Colors (frame *i*) (b) Vel. field (frame *i*+1) (c) Advection (frame *i*+1) (d) Vel. field extrapolated (*i*+1) (e) Advection extrapolated (*i*+1)

Figure 10: *Advection of colors (a) using a standard velocity field (b) can leave overpainted regions behind (c). Our approach extrapolates the velocity field into the background of each layer (d, original outline in white), allowing advection outside the borders of the shape (e).*

The artist selects a subset of the input frames that she considers to be characteristic poses (one every 10 to 20 frames), then draws over those frames the desired final result using her favorite painting system (e.g., Adobe Photoshop®, Corel Painter®). For best results, shading variations in the input should be respected in the stylized painting (Figure 9). These keyframes form both hard constraints that are exactly matched during the optimization, and pairs of style inputs/outputs for the modified TCIA algorithm (Section 4.1).

We are particularly interested in painterly visual styles, which are generally incompatible with the perfect boundaries of computer graphics rendering. To enable the artist to paint loosely across boundaries, we decompose our scene into separate layers that can be painted separately (Figure 9(c)). For each element, we extrapolate the image-space velocity fields using the distance transform of the input layer. As a result, if the artist chooses to apply paint beyond the rendered boundary of an element in the input animation, the overpainted regions will be carried along sensibly by the extrapolated velocity field (Figure 10(e) vs. 10(c)). This decomposition

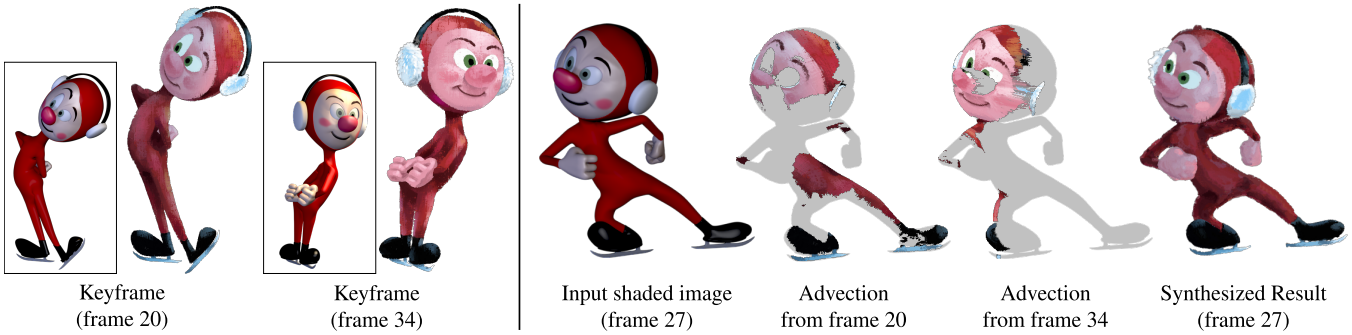


Figure 11: *Comparison with keyframe advection. An artist paints keys at frames 20 and 34, using the input rendering (boxed) as reference. Advecting the keyframes to the intermediate frame 27 gives incomplete results due to occlusions (character body) and conflicting results due to color choices by the artist (face). Our method fills incomplete areas and chooses an appropriate mix in conflicting areas.*

into layers allows overdraw and partial transparency (Figure 9(d)), and reduces the ambiguity of the analogy by constraining the optimization search space to colors of the same layer.

4.1 Optimization Changes

The use of keyframes in the TCIA algorithm raises new challenges. The stylized animation has to match exactly the hard painted constraints while smoothly changing in time and respecting the analogy. As illustrated in Figure 11, the naïve solution of advecting the colors from the two nearest keys and cross-fading does not produce satisfying results. Occlusions give rise to incomplete advection and holes that need to be in-painted, while complete two-way advection gives rise to conflicting colors that need to be blended. Linear blending neither respects the shading variations of the input animation, nor produces sharp textured outputs. To address these issues, we turn the TCIA algorithm into a nonlinear in-betweening technique by adding new temporal terms in the optimization and facilitating transitions between keyframes.

Modified goal function. The temporal coherence term defined in Section 3.2 is asymmetric with respect to the time direction (forward or backward pass). This property enhances temporal continuity by penalizing any changes in time, which is generally suitable for unconstrained synthesis. However, with keyframes, this goal tends to privilege the most recently visited key, which prevents the interpolation from smoothly matching the next key. To solve this problem, we replace g_{tc} by two new goals in the optimization that correspond to the time derivatives of g_{in} and g_{out}.

The **input time-derivative** goal encourages the synthesized animation to change the same way the input animation does. For the visible pixels at the previous frame, it measures the temporal variation of the input images I_t and I_{t-1} and compares it with the corresponding variation in the style input S at the offsets $\mathbf{C}_t$ and $\mathbf{C}_{t-1}$.

$$\begin{aligned} g^-_{\partial in} = &\chi^-(\mathbf{p}+\Delta\mathbf{p},t)|(I_t(\mathbf{p}+\Delta\mathbf{p}) \\ &- I_{t-1}(\mathbf{p}+\Delta\mathbf{p}-V_t^-(\mathbf{p}+\Delta\mathbf{p}))) \\ &- (S(\mathbf{C}_t(\mathbf{p},\Delta\mathbf{p})) - S(\mathbf{C}_{t-1}(\mathbf{p}-V_t^-(\mathbf{p}),\Delta\mathbf{p})))|^2 \end{aligned} \tag{10}$$

By replacing I_t and S in $g^-_{\partial in}$ with $\hat{I}_t$ and $\hat{S}$, we have an **output time-derivative** term $g^-_{\partial out}$. This term allows the output to change consistently with spatial variation in the style, but we found that its influence was limited in practice.

We define similar terms $g^+_{\partial in}$ and $g^+_{\partial out}$ using velocities V_t^+ for the backward sweeps.

In-Betweening algorithm. Apart from these new terms in the goal function, we make three important changes to the algorithm described in Section 3.3.

First, when a forward or backward sweep reaches a keyframe, the content of the key is copied and replaces the currently synthesized image. This ensures that keyframes are perfectly matched, and gives the best possible starting point for advection to the next frame. Partial keyframes can be defined by providing an extra mask as input, or using the alpha channel of the image. Semi-transparency of partial keys is an open problem that we leave for future work.

Second, we modify the random merging scheme to bias towards the closest keyframe, which encourages smooth convergence. Given frame t and the indices of the last and next keys $t_l \leq t$ and $t_n \geq t$, we randomly choose pixels from the forward pass with probability $(t_n - t)/(t_n - t_l)$ and from the backward pass with probability $(t - t_l)/(t_n - t_l)$.

Finally, we modify the propagation and random search steps of the PatchMatch algorithm to handle multiple keyframes. As a preprocess, we follow the 3D velocity field from one keyframe to the next to create *ribbons* that connect a keyframe pixel $\mathbf{p}_t$ to its corresponding pixels $\mathbf{p}_{t_l}$ and $\mathbf{p}_{t_n}$ in the neighboring keyframes. Each ribbon is an image-space representation of the 3D path of the associated point, e.g.,, the tip of the character's nose. During the PatchMatch optimization, the ribbons extend the search space to include both the spatial neighbors in the current keyframe and the spatial neighbors of the corresponding point in neighboring keyframes. This approach greatly eases and smooths transitions between keyframes and allows efficient exploration of the most relevant parts of the search space.

4.2 Results

Figure 1 shows the result of in-betweening two keyframes of an animation sequence with strong shape and shading variations. Thanks to the image analogy approach, our algorithm finds a non-linear interpolation path between the keys to match the non-linear change in brightness. It also adapts to the "squash and stretch" distortions and 3D motion (e.g., head rotation) of the character.

Figure 12 shows final composites of keyframe in-betweening for two polished 3D character animations and three styles painted by different artists (please see the accompanying videos for the full sequences and breakdowns). On the bottom row, only the first and last overlays are keyframes. Our method successfully handles occlusions of the legs and their variations in orientation. The top row shows a more "three-dimensional" example with complex occlusions and disocclusions and stronger lighting variations. Our

Figure 12: *Results of keyframe in-betweening. Upper left: first and third overlays are keyframes. Upper right: third overlay is a keyframe. Bottom: first and last overlays are keyframes.*

algorithm finds convincing in-betweens that faithfully interpolate the keyframes even when advection only provides a limited starting point.

These sequences demonstrate how our system might be used in a production context, and thus include a range of different effects, not all produced by our system. The skater is rendered entirely with our method, including changes in illumination, as are the snowdrifts and the pools of light on the surface of the ice for the top row sequences. The snowdrifts and background of the second row are hand-painted. The reflection of the skater is added using a standard compositing software. The shafts of light are rendered using a conventional 3D rendering package. The surface of the ice is texture mapped, as is the sky background. The scraping effect around the skates is generated with a particle system and composited.

5 Limitations, Future Work and Discussion

This paper presents the first method for automatically in-betweening 2D painted keyframes based on 3D character animation. We believe the method, as presented, is a practical technique for animation production. We have extended the utility of Image Analogies by generalizing the examples over 2D rotations. We have made it possible for image analogies-based algorithms to handle line stylization. We have introduced controls to prevent over-use of small regions of the input examples. We have provided means for dealing with irregular boundaries. And we have developed an optimization algorithm that produces high-quality temporally coherent results using practical amounts of computing resources.

In the course of this work, our goal was to provide adequate solutions to each of the constituent sub-problems. In no case would we claim that these solutions are optimal; there are considerable opportunities to push the work further, improving the quality of the result, the ease of use, or reducing the resources required.

Goal function weights. Finding an adequate balance between the different goals in the optimization requires some experimentation. The parameters are style specific; for instance, structured patterns (e.g., hatching or paint strokes) need higher spatial coherence than stochastic or smooth textures (e.g., watercolor wash) to produce outputs that match the target style (see the supplementary video for examples). There is also an important tradeoff between the desired amount of spatial continuity and temporal coherence. Pushing one or the other to its extreme leads either to patches of texture abruptly appearing and disappearing, or disconnected pixels cross-dissolving. The parameters used for each sequence are shown in the supplementary video. The values were chosen through experimentation. We found that for most cases, parameters within a factor of two from the default values produced good results.

Improved optimization. We use PatchMatch [Barnes et al. 2009] for optimizing our goal function, as it produces visually pleasing results while mapping easily to GPUs. Our approach behaves well in a broad range of situations with objects moving at moderate to high speeds with varied lighting. It performs less well for large, continuous regions that change gradually with illumination or camera motion, since the optimization is not allowed to blend colors from the style. In areas of gradual change, discrete patches can sometimes appear and disappear, causing distracting artifacts. We believe smoother transitions could be achieved by incorporating a more sophisticated optimization scheme such as gradient domain optimization [Darabi et al. 2012], or search across scales and rotations [Barnes et al. 2010], but have not yet investigated such approaches.

Our algorithm is fully parallel in space but sequential in time. While this allows almost interactive feedback when adjusting parameters to synthesize static frames, the final high-resolution result takes 10 to 12 minutes per final frame (with the ability to see early previews at coarse spatial resolution). Parallelization in time seems a natural extension, but our experiments with a simple temporal coarse-to-

fine scheme have shown non-trivial obstacles. The sequential nature of the current algorithm plays an important role in producing temporally coherent results, and is difficult to match with a fully time-parallel synthesis.

Better analogy. In the current work, the only data we use from our input animation is the rendered image intensities and the velocity field. This creates situations where the proper stylization is ambiguous. In many cases, better results could probably be obtained by augmenting the input with additional data. We colored the cheeks and nose of our 3D character in red and his lips in black to reduce the ambiguity of the proper stylistic correspondences, but one can certainly imagine providing additional rendered data besides the rendered image intensities to improve the algorithm's ability to establish the desired correspondence. Normals, curvature information and texture coordinates are examples of additional data that may be used effectively.

Histogram. The histogram term provides an indirect means of controlling stylization by analogy. However, it only considers the distribution of the output offsets without taking into account the statistics of the input image and style input. If the input is especially dark, then it makes sense to bias the offsets towards the dark areas of the style input. Automatically estimating an approximate desirable target distribution based on the inputs could significantly improve the effectiveness of the histogram control. The histogram control might also be modified to take into account distance to nearby keyframes.

Layers. Our simple decomposition into layers has the important benefit of matching the traditional pipeline of digital artists, allowing them to use regular drawing software. However, this decomposition is not always possible – especially for temporally self-occluding parts such as the legs and arms of the character – which prevents overpaint in those regions. Taking inspiration from Bonneel et al. [2010], it might be possible to generate a *guide* with fuzzy boundaries respecting self-occlusions. This would, however, require the artist to specify extra depth or inequality relations when painting the keys.

Additional inputs. Our pipeline is generic enough to apply on any animated input with known velocities. This makes the approach directly applicable to live-action videos where accurate forward and backward optical flows can be extracted using vision algorithms. Spatio-temporal segmentation would also be required for layer decomposition. In principle, traditional 2D animations could be handled as well, using an estimated frame-to-frame correspondence (e.g., computed with the method of Sýkora et al. [2011]).

Aside from any details of our approach, we hope a few of its underlying ideas will be key enablers of future temporally-coherent image stylization. Creating temporal coherence by optimization rather than construction allows a rich variety of visual styles. Close attention to occlusion, disocclusion and irregular boundaries enables high-quality results that retain the warmth and organic feel of hand-painted imagery. And the proper use of example-based methods puts control in the hands of the artist, where it belongs.

Acknowledgements

We would like to thank the anonymous reviewers, Tony DeRose, Mark Meyer, Aaron Hertzmann and Joëlle Thollot for their comments and advice. Thanks to our many Pixar colleagues who worked on these shots. Teddy Newton, Sanjay Patel, Don Shank, Jamie Frye, Ralph Eggleston and Angus MacLane provided wonderful artwork and art direction, and pushed us to go farther. We appreciate Andrew Schmidt, Brian Tindall, Bernhard Haux and Paul Aichele for creating a character capable of hitting exaggerated poses appropriate for NPR rendering, and we also thank Danny Nahmias, Mach Kobayashi, Fernando de Goes and Sue Kalache for their contributions to this project.

References

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

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *Proceedings of the 7th symposium on Interactive 3D graphics*, 217–226.

BARGTEIL, A. W., SIN, F., MICHAELS, J., GOKTEKIN, T., AND O'BRIEN, J. 2006. A texture synthesis method for liquid animations. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 345–351.

BARNES, C., SHECHTMAN, E., FINKELSTEIN, A., AND GOLDMAN, D. B. 2009. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics 28*, 3, 24:1–24:11.

BARNES, C., SHECHTMAN, E., GOLDMAN, D. B., AND FINKELSTEIN, A. 2010. The generalized PatchMatch correspondence algorithm. In *Proceedings of the 11th European conference on computer vision conference on Computer vision*, 29–43.

BÉNARD, P., BOUSSEAU, A., AND THOLLOT, J. 2009. Dynamic solid textures for real-time coherent stylization. *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, 121–127.

BÉNARD, P., LAGAE, A., VANGORP, P., LEFEBVRE, S., DRETTAKIS, G., AND THOLLOT, J. 2010. A dynamic noise primitive for coherent stylization. *Computer Graphics Forum 4*, 29, 1497–1506.

BÉNARD, P., BOUSSEAU, A., AND THOLLOT, J. 2011. State-of-the-Art Report on Temporal Coherence for Stylized Animations. *Computer Graphics Forum 30*, 8, 2367–2386.

BÉNARD, P., JINGWAN, L., COLE, F., FINKELSTEIN, A., AND THOLLOT, J. 2012. Active Strokes: Coherent Line Stylization for Animated 3D Models. In *NPAR 2012 - 10th International Symposium on Non-photorealistic Animation and Rendering*, ACM, 37–46.

BONNEEL, N., VAN DE PANNE, M., LEFEBVRE, S., AND DRETTAKIS, G. 2010. Proxy-guided texture synthesis for rendering natural scenes. In *15th International Workshop on Vision, Modeling and Visualization*, 87–95.

BOUSSEAU, A., NEYRET, F., THOLLOT, J., AND SALESIN, D. 2007. Video watercolorization using bidirectional texture advection. *ACM Transactions on Graphics 26*, 3, 104.

CAO, C., CHEN, S., ZHANG, W., AND TANG, X. 2011. Automatic motion-guided video stylization and personalization. In *Proceedings of the 19th ACM international conference on Multimedia*, 1041–1044.

CHEN, J., AND WANG, B. 2009. High quality solid texture synthesis using position and index histogram matching. *The Visual Computer 26*, 4, 253–262.

DANIELS, E. 1999. Deep canvas in Disney's Tarzan. In *ACM SIGGRAPH 99 Conference abstracts and applications*, 200.

DARABI, S., SHECHTMAN, E., BARNES, C., GOLDMAN, D. B., AND SEN, P. 2012. Image melding. *ACM Transactions on Graphics 31*, 4, 82:1–82:10.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *Proceedings of SIGGRAPH '01*, 341–346.

GRANLUND, G. H., AND KNUTSSON, H. 1995. *Signal Processing for Computer Vision*. Springer.

HAN, J., ZHOU, K., WEI, L.-Y., GONG, M., BAO, H., ZHANG, X., AND GUO, B. 2006. Fast example-based surface texture synthesis via discrete optimization. *The Visual Computer 22*, 9-11.

HARO, A. 2003. *Example based processing for image and video synthesis*. PhD thesis. AAI3117934.

HASHIMOTO, R., JOHAN, H., AND NISHITA, T. 2003. Creating various styles of animations using example-based filtering. In *Computer Graphics International*, 312–317.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *Proceedings of SIGGRAPH '01*, 327–340.

KABUL, I., PIZER, S. M., ROSENMAN, J., AND NIETHAMMER, M. 2011. An Optimal Control Approach for Texture Metamorphosis. *Computer Graphics Forum 30*, 8, 2341–2353.

KAGAYA, M., BRENDEL, W., DENG, Q., KESTERSON, T., TODOROVIC, S., NEILL, P. J., AND ZHANG, E. 2011. Video painting with space-time-varying style parameters. *IEEE Transactions on Visualization and Computer Graphics 17*, 1, 74–87.

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. In *ACM Transactions on Graphics*, vol. 21, 755–762.

KASS, M., AND PESARE, D. 2011. Coherent noise for non-photorealistic rendering. *ACM Transactions on Graphics 30*, 4, 30:1–30:6.

KULLA, C., TUCEK, J., BAILEY, R., AND GRIMM, C. 2003. Using texture synthesis for non-photorealistic shading from paint samples. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, 477–481.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *ACM Trans. Graph. 24*, 3, 795–802.

KWATRA, V., ADALSTEINSSON, D., KIM, T., KWATRA, N., CARLSON, M., AND LIN, M. C. 2007. Texturing fluids. *IEEE Transactions on Visualization and Computer Graphics 13*, 5, 939–952.

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

LAI, C.-H., AND WU, J.-L. 2007. Temporal texture synthesis by patch-based sampling and morphing interpolation. *Computer Animation and Virtual Worlds 18*, 4-5, 415–428.

LASRAM, A., AND LEFEBVRE, S. 2012. Parallel patch-based texture synthesis. In *Proceedings of the 4th ACM SIGGRAPH / Eurographics conference on High-Performance Graphics*, 115–124.

LEE, H., SEO, S., AND YOON, K. 2011. Directional texture transfer with edge enhancement. *Computers & Graphics 35*, 1, 81–91.

LEFEBVRE, S., AND HOPPE, H. 2006. Appearance-space texture synthesis. *ACM Transactions on Graphics 25*, 3, 541–548.

LIN, L., ZENG, K., LV, H., WANG, Y., XU, Y., AND ZHU, S.-C. 2010. Painterly animation using video semantics and feature correspondence. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, no. 212, 73–80.

LIU, Z., LIU, C., SHUM, H.-Y., AND YU, Y. 2002. Pattern-based texture metamorphosis. *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications 1*, 184–191.

MATUSIK, W., ZWICKER, M., AND DURAND, F. 2005. Texture design using a simplicial complex of morphable textures. *ACM Transactions on Graphics 24*, 3, 787–794.

O'DONOVAN, P., AND HERTZMANN, A. 2011. AniPaint: Interactive Painterly Animation from Video. *IEEE Transactions on Visualization and Computer Graphics 18*, 3, 475–487.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. *Proceedings of SIGGRAPH '01*, 581.

RAY, N., LÉVY, B., WANG, H., TURK, G., AND VALLET, B. 2009. Material Space Texturing. *Computer Graphics Forum 28*, 6, 1659–1669.

RUITERS, R., SCHNABEL, R., AND KLEIN, R. 2010. Patch-based Texture Interpolation. *Computer Graphics Forum 29*, 4, 1421–1429.

SCHMID, J., SENN, M. S., GROSS, M., AND SUMNER, R. W. 2011. OverCoat: an implicit canvas for 3D painting. In *ACM Transactions on Graphics*, vol. 30, 28:1–28:10.

SHECHTMAN, E., RAV-ACHA, A., IRANI, M., AND SEITZ, S. 2010. Regenerative morphing. In *IEEE Conference on Computer VIsion and Pattern Recognition (CVPR)*, 615–622.

SÝKORA, D., BEN-CHEN, M., ČADÍK, M., WHITED, B., AND SIMMONS, M. 2011. TexToons. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, 75–84.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of SIGGRAPH 2000*, 479–488.

WEI, L.-Y., LEFEBVRE, S., KWATRA, V., AND TURK, G. 2009. State of the Art in Example-based Texture Synthesis. In *Eurographics 2009 - State of the Art Report*, 93–117.

WHITED, B., DANIELS, E., KASCHALK, M., OSBORNE, P., AND ODERMATT, K. 2012. Computer-assisted animation of line and paint in Disney's Paperman. In *ACM SIGGRAPH 2012 Talks*, SIGGRAPH '12, 19:1.

WINNEMÖLLER, H., OLSEN, S. C., AND GOOCH, B. 2006. Real-time video abstraction. *ACM Transactions on Graphics 25*, 3, 1221–1226.

ZHANG, J., ZHOU, K., VELHO, L., GUO, B., AND SHUM, H.-Y. 2003. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Transactions on Graphics 22*, 3, 295–302.