Automatic Inbetweening for Stroke-Based Painterly Animation

Nicolas Barroso, Amélie Fondevilla, David Vanderhaeghe

DOI: 10.1111/cgf.15201

COMPUTER GRAPHICS *forum*
Volume 0 (2024), number 0, e15201

Automatic Inbetweening for Stroke-Based Painterly Animation

Nicolas Barroso,[1] Amélie Fondevilla[2] and David Vanderhaeghe[1]

[1]IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3, Toulouse, France
{nicolas.barroso, david.vanderhaeghe}@irit.fr
[2]Les Fées Spéciales, Montpellier, France
afondevilla@laposte.net

Abstract

Painterly 2D animation, like the paint-on-glass technique, is a tedious task performed by skilled artists, primarily using traditional manual methods. Although CG tools can simplify the creation process, previous works often focus on temporal coherence, which typically results in the loss of the handmade look and feel. In contrast to cartoon animation, where regions are typically filled with smooth gradients, stroke-based stylized 2D animation requires careful consideration of how shapes are filled, as each stroke may be perceived individually. We propose a method to generate intermediate frames using example keyframes and a motion description. This method allows artists to create only one image for every five to 10 output images in the animation, while the automatically generated intermediate frames provide plausible inbetween frames.

Keywords: animation, 2D techniques, rendering, non-photorealistic rendering

CCS Concepts: • Computing methodologies → Non-photorealistic rendering; Animation

1. Introduction

Stylized animation is widely used by artists to convey expressive stories. Computer graphics tools have greatly enhanced productivity and have been utilised by artists for many years to create 2D animations. Most of these tools are tailored for cartoon rendering and assist in creating animations using the keyframe approach, which typically features smooth colour gradients and simple strokes for contours and details. In this paper, we focus on stroke-based styles, such as paint-on-glass, pastels and charcoal, which have more texture and where each individual stroke might be noticeable. These styles benefit from explicit stroke modeling, allowing for fine user control. Our implementation specifically addresses stroke-based painterly rendering. Recent hand-made animations that inspire our work include *Loving Vincent* [KW17] and *La Traversée* [Mia21].

The keyframe approach is the *de facto* standard for cartoon animation creation. First, the artist draws the keyframes of the animation, and then the inbetweening process involves creating the intermediate frames between each pair of keyframes. This process is facilitated by the use of motion interpolation and skeletal animation in 2D for automatic intermediate frame synthesis. In contrast, paint-on-glass styles are traditionally created by hand using the so-called *under-the-camera* setup, where the artist draws under an acquisition device. Each frame of the animation is drawn sequentially, using the previous frame as a starting point. This method requires the artist to have the entire animation sequence in mind and lacks the flexibility of keyframe or skeletal animation.

We propose an automatic inbetweening method that combines the look and feel of the paint-on-glass style with the ease of control provided by keyframe approaches. Our work follows an example-based approach, capturing the style from examples drawn by the artist (Figure 1). This type of control aligns with what professional artists excel at: drawing efficiently to convey their unique style. The proposed method does not require additional skills from professional artists and integrates seamlessly into the production pipeline.

We propose generating intermediate frames by replicating the drawing process, with strokes rendered in the same manner as the keyframes. The artist retains fine control over the generated frames, as the frame description includes each stroke's curve path. This contrasts with other approaches that directly generate image pixels, such as those using neural networks [JYF*20, HTT*20].

To drive the generation of an intermediate frame, a user-provided motion field is required. We demonstrate three examples of motion field sources: motion rendered from a 3D scene that roughly captures the intended animation motion which is then extrapolated to

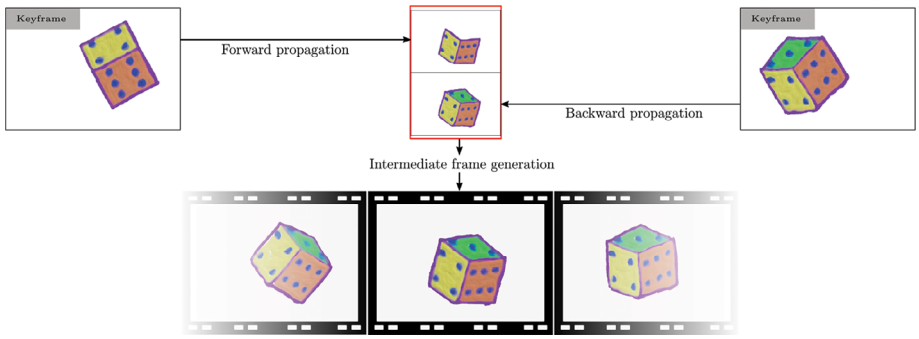

Figure 1: *Our method generates inbetween frames using the propagation of strokes from keyframes. A motion field, given as input, drives the propagation process. The high-level description of the generated strokes enables the artist to edit the resulting frames if desired.*

allow keyframe strokes to extend beyond 3D object boundaries; motion field extracted from video and sketched motion field defined by a set of curves drawn by the artist.

We implement our method for painterly rendering [VC12], including brush and paint simulation. Our method is built on a paint simulator that accurately renders the bi-directional paint exchange between a virtual brush and the canvas. Keyframes are painted using brushes and strokes, describing how the paint is applied to the canvas along a path. The artist has the ability to edit the stroke curves and brush properties to modify a frame.

Our method preserves the natural hand-made look of animated painted frame sequences and does not prioritize temporal coherence, as traditional hand-made stroke-based style animations are not temporally coherent.

Our contributions are as follows:

- The extrapolation of motion field to obtain a 2D motion field suitable for stroke propagation (Section 6).
- An optimization scheme for stroke propagation that preserves geometric features of keyframe strokes (Section 4),
- An intermediate frame generator that leverages candidate strokes from the propagation (Section 5),
- Stroke grouping and mixing to create intermediate frames from two candidate stroke sets (Section 5.1),

A preliminary version of our approach was previously presented at a non-peer-reviewed national conference [BFV21], featuring a simpler stroke optimization step and lacking stroke grouping and mixing, which resulted in an excessive quantity of paint in the generated inbetweens.

2. Related Works

Synthesis of intermediate frames. The automatic synthesis of intermediate frames, also known as inbetweening, plays a significant role in computer-assisted creation of 2D animations. Most methods operate in image space [BCK*13, JST*19], using texture synthesis [BSFG09, SJT*19] and depicting as-rigid-as-possible deformations [SDC09, DBB*17]. Other approaches use triangulation of the image space and apply deformation directly to this triangulated mesh, employing 2D skinning techniques [BKLP16]. These texture synthesis methods are agnostic to the tools used to produce the examples. While they offer parameters to control the generated raster images, they do not allow the result to be edited beyond the direct pixel modifications. In contrast, our approach generates intermediate frames composed of parametric strokes. This representation provides the artist with high-level control, enabling efficient post-editing.

Other approaches operate at the stroke level, either by directly using the motion of an underlying 3D animation [WDK*12], or by deducing the deformation between two frames through rasterized [MFXM21] or vectorized stroke pairing [WNS*10, YSC*18, JSL22]. Even *et al.* [EBB23] propose using transient embeddings to match groups of strokes that share similar motion, addressing rough sketches where keyframes present highly dissimilar number of strokes. While most stroke-based approaches focus on sketches, a few address the filling of regions. In the latter case, the inbetweening problem can be addressed by embedding the strokes in a 3D space where the strokes follow 3D surface's motion [SSGS11, BBS*13]. These approaches produce strong motion coherence between the strokes and the underlying animated 3D scene. This strong coherence suggests that the scene objects are not merely painted onto a 2D canvas but the animation is a 3D drawing. Our primary focus is to reproduce the look and feel of a hand-drawn animation in 2D. To achieve this style, we generate strokes directly in 2D from example keyframes.

O'Donovan *et al.* [OH11] propose to generate intermediate frames by energy minimization, controlling the size, the shape and the density of strokes. The generated animation is refined by the artist with special orientation strokes in keyframes. The refinements are propagated on next frames using an optical flow. In contrast, we define the style of the keyframes by the artist drawn inputs and our approach reproduces this style in the generated intermediate frames.

Generation of strokes by example. Example-based style transfer was widely discussed in literature, and covers a large panel of topics, among which stroke-based rendering methods. Haeberli [Hae90] and Hertzmann [Her98] have set the basis for painterly rendering by representing strokes by attributes such as width, colour or orientation. They generate strokes by computing their attributes from the local characteristics of an underlying image. This principle is re-used for example-based methods to compute stroke attributes from real paintings. Early example-based methods compute strokes

attributes by establishing dictionaries from real examples [ZZ11] or by learning [WCHG13]. In Lu *et al.*[LYFD12], parts of stroke are real-time predicted and refined by energy minimization to match perfectly a stroke in a dataset. These methods cannot be used for keyframes interpolation and rely on large datasets, making it challenging to obtain the necessary data from keyframes alone. Strokes are predicted frame by frame by establishing fuzzy correspondences through similarity analysis between the current and previous frames [XWSY15]. We introduce a fully automatic approach, where the artist only needs to draw keyframes.

Todo *et al.*[TKK*22] de-couple style-specific elements related to the 3D scene's geometry, illumination and view from user examples and transform them into stroke attributes that vary in image space. Their method builds a stroke field that handles temporal coherence through an optimization penalizing spatial and temporal variation. Once stroke positions are determined, strokes are synthesized using a texture and attributes queried from the generated stroke field. Our approach uses 2D motion field and keyframe interpolation to generate intermediate frames. This allows us to provide a suitable pipeline without the dependency on any 3D data which are complex to provide for 2D artists.

Closer to our method, Chen *et al.*[CZBB20] use rigid transformation on strokes given as examples in keyframes to compose intermediate frames. These intermediate frames are built by filling a target density map with selected and modified strokes. The density map is estimated by interpolating keyframes densities with the image registration method [SDC09]. Combined with transformed strokes, the animation sequence shows a hand-drawn look. We share the same approach, but in our case, the transformation of strokes is guided by an underlying motion field as Ellsworth [Ell18]. In addition, our approach modifies the curve shape using an optimization scheme that preserves geometric properties from the keyframes.

Stroke ordering. Hays *et al.*[HE04] and Northam *et al.*[NIK10] discuss how strokes should be distributed on the canvas to preserve details or salient edges. Yang *et al.*[YXD*20] order strokes in a way that produces a natural result for medium synthesis, such as Chinese ink. However, these approaches have not been applied to inbetweening, which poses new challenges for stroke ordering, especially when keyframes have different numbers of strokes and different stroke orders. Our inbetweening process involves mixing strokes from two keyframes while accounting for the colour blending inherent in the rendering process.

3. Overview

Starting with two hand-drawn keyframes and corresponding motion field, our method automatically generates all the intermediate frames in the sequence. When the artist draws a keyframe, our system captures an ordered list of strokes. Each stroke is recorded as a polyline, along with brush information, including brush radius, paint colour and initial paint amount. Each point of the polyline also captures tool pressure and orientation, if the input device provides this data.

The provided motion field describes how the keyframe's content should move from one frame to another. Each motion field frame is defined in image space, independent of the actual keyframe's strokes. Since our intermediate frame generation relies on strokes from keyframes both before and after on the timeline, we need forward and backward motion vectors. We present various examples of motion field sources that depict the motion between keyframes: motion from animated 3D scenes, optical flows extracted from videos and 2D motion curves drawn by the artist (Section 7). In the spirit of paint-on-glass animation, we assume that complex motions, such as overlapping objects, are decomposed into layers. Each layer is animated independently, using its own keyframes and motion field.

We model the style of a frame based on the low-level characteristics of its strokes: location, curve shape and brush information. Our algorithm is designed to generate frames whose strokes exhibit characteristics similar to those in the keyframes and whose locations follow the motion field.

Intermediate frame generation algorithm consists in three main parts. First, it advects and regularises the keyframes' strokes to generate *candidate strokes* (Section 4). Second, it builds an expected density of strokes, represented by a target paint amount map (Section 4.2). Finally, it computes the intermediate frame stroke list (Section 5), keeping only relevant pieces of each candidate strokes.

The generated frames use the same data structure as the keyframes, specifically a polyline representation of the strokes. Although generating intermediate frames can be time-consuming (taking up to several minutes per frame) and may not provide interactive user control, the artist can modify these frames with the same tools and level of control as the keyframes. Additionally, any generated frame can serve as a new keyframe and be modified by the artist to create further refined intermediate frames.

4. Keyframe Propagation

The generation of an intermediate frame takes as input a set of candidate strokes and a target paint amount map. This section presents how these two inputs are computed from the input keyframes.

Each intermediate frame is situated between two keyframes in the timeline, referred to as the previous keyframe and the next keyframe. To generate candidate strokes for an intermediate frame, the strokes from the previous keyframe are advected using the forward motion field, while the strokes from the next keyframe are advected using the backward motion field. Each advection step moves the polyline points of the strokes from one frame to the adjacent frame, resulting in candidate strokes for the intermediate frame. To control the curve shape of the advected stroke, we propose a stroke shape regularisation algorithm. Slight divergences in the motion field can significantly impact the stroke curve, potentially distorting its original shape. However, the artist may wish to maintain the original stroke curve shape throughout the animation, allowing for nearly rigid transformations only.

4.1. Stroke advection and regularisation

The regularisation allows stroke curve shape to roughly follow the motion while preserving the overall shape of the stroke drawn in the keyframe. For the remainder of this section, we consider the

previous keyframe and the forward motion field. Each stroke of the keyframe is transformed into one candidate stroke for each intermediate frame. These candidate strokes for an intermediate frame are computed iteratively from the candidate strokes of the previous intermediate frame in the timeline, or from the keyframe for the first intermediate frame.

We denote a stroke curve as a function $f(x)$, with $x \in [0, 1]$ the curve parameter. The jth point of the polyline curve is $f(x_j)$ and x_j the corresponding parameter.

To compute the candidate stroke regularised curve f_n in intermediate frame t_i, of a stroke curve f_p coming from the previous frame t_{i-1}, we consider two other curves: the advected curve f_a and the reference curve f_r.

The advected curve f_a is defined as

$$f_a(x_j) = f_p(x_j) + M(t_{i-1} \to t_i, f_p(x_j))$$

where $M(t_{i-1} \to t_i, p)$, the motion vector from t_{i-1} to t_i at pixel position p.

The regularisation process is controlled by a blending weight that blends the shape between the advected curve and the reference curve. The blending weight of a stroke is defined as the function $\sigma(x) \in [0, 1]$, where $x \in [0, 1]$ is the parametric value of the polyline curve. $\sigma(x)$ equals zero, respectively, one, corresponds to stroke location where we expect rigid transformation, respectively, advection, of the curve. $\sigma(x)$ is automatically computed along f_p according to the motion field (see Section 6). Additionally, we allow the artist to edit the blending weight at the stroke level to enable fine-grained control of the regularisation process.

The reference curve f_r is computed by applying a 2D rigid transformation Y^* to f_p:

$$f_r(x_j) = Y^*(f_p(x_j))$$

We define Y^* by the least-squares rigid motion using SVD approach [SHR17] as

$$Y^* = \arg\min_Y \sum_j w_j \left\| Y(f_p(x_j)) - f_a(x_j) \right\|^2$$

where the weights $w_j = \lfloor \sigma(x_j) \rfloor$, such that only points where we expect advection are used to define Y^* so f_r will *follow* these points. If no such point exists, we use $w_j = 1$ for all points so that f_r follows f_a, still using rigid transform.

We compute f_n through an optimisation to obtain a mix between f_a and f_r according to the blending weights $\sigma(x)$. Since the curve is represented by a polyline that may contain hundreds of points, we first fit a piecewise cubic Bézier curve onto the polyline [DP73] to reduce the number of variables to optimise. We formulate the problem as an optimisation on the control points of the Bézier curve and we use a combination of three energy functions for minimisation. These energies are measured along the piecewise cubic Bézier curve for K evenly distributed sample points x_k, $k \in [1, K]$.

The first energy function, E_f, corresponds to the data fidelity term, which ensures that, without other constraints, the optimisation will spatially converge to the linear interpolation f_b between

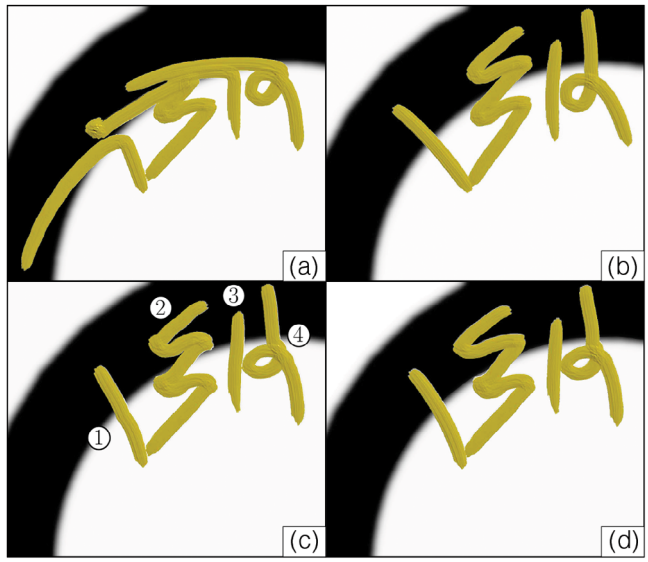

Figure 2: *An example of four strokes and the curves involved during the regulation process: (a) advected curve f_a, (b) the reference curve f_r, (c) linear blended curve f_b and (d) regularised curve f_n. The motion field is the one of Figure 3. Background intensity corresponds to blending weight $\sigma(x)$ and varies from zero to one. Linear blended curve artifacts are emphasized in (c) with cracks (1, 4), collapse (2) and elongation (3).*

the advected curve and the rigid curve, according to $\sigma(x)$:

$$f_b(x) = f_r(x) + \sigma(x)(f_a(x) - f_r(x))$$

$$E_f = \frac{1}{K} \sum_{k=1}^{K} \| f_b(x_k) - f_n(x_k) \|^2$$

While it provides a good initialization, the linear blended curve might exhibit artifacts, such as introducing sharp corners that are not present in the reference stroke curve (see Figure 2c).

The next two energy functions, E_c and E_l, penalize variations between the reference and the regularised curve by minimizing, respectively, the difference in curvature and the difference in distance between every consecutive sample pairs.

$$E_c = \frac{1}{K} \sum_{k=1}^{K} (1 - \sigma(x_k))(\gamma_r(x_k) - \gamma_n(x_k))^2$$

where γ_λ is the curvature of f_λ, $\lambda \in \{r, n\}$ and

$$E_l = \frac{1}{K-1} \sum_{k=1}^{K-1} \left(1 - \frac{\sigma(x_k) + \sigma(x_{k+1})}{2}\right)(\delta_r - \delta_n)^2$$

where

$$\delta_\lambda = \| f_\lambda(x_k) - f_\lambda(x_{k+1}) \|$$

We use the Levenberg–Marquardt algorithm [Mar63] to solve the optimization problem. The control points of the advected curve f_a serve as the initialization, as they provide an initial approximation

Figure 3: *When strokes (in purple) from a keyframe (left) are advected by a motion field, forward advected strokes (middle) exhibit strong deformations, assuming that the strokes extend slightly beyond the shape they depict, and thus are deformed by the counterclockwise motion. Our regularisation of strokes (right) captures the motion while maintaining strokes shape from the keyframe, according to a blending weight. The underlying motion field (shown as background colour and arrows) depicts a central clockwise rotation and an external counterclockwise rotation. The blending weights (shown in Figure 2) are zero in the transitional region between the two rotation.*

of the regularised curve. The minimized energy is a weighted combination of the three energies functions:

$$E = E_f + \alpha E_c + \beta E_l$$

In our results, we use $\alpha = 25$ and $\beta = 50$. Figure 3 illustrates how the regularisation process handles strokes that are highly deformed by the advection.

Similarly, regularised strokes are computed from the next keyframe using the backward motion field. After stroke advection and regularisation, we have two lists: one from each keyframe, containing candidate strokes for each intermediate frame. Each stroke list maintains the stroke order of its originating keyframe.

4.2. Paint amount propagation

The rendering process of a keyframe involves a media simulation. Our implementation utilises a paint simulator which mixes paint colour onto a virtual canvas (see Section 7). The canvas also represents paint amount for mixing paint computations and for rendering (*e.g.* to render paint thickness). We leverage this paint amount to drive the intermediate frame generation, such that once rendered the paint amount of the intermediate frame is similar to the one in the keyframe, without paint accumulation or missing paint.

First, the process advects paint amount map from the two keyframes to each intermediate frame by applying successive motion field frame. To this end, we define an inverse mapping to ensure that every pixel of the intermediate frame receives a paint amount, unless the motion field moves outside the domain defined by the previous frame:

$$A_i^0(p) = A_{i-1}^0\left(p + M(t_i \to t_{i-1}, p)\right)$$
$$A_i^N(p) = A_{i+1}^N(p + M(t_i \to t_{i+1}, p))$$

where A_i^0 and A_i^N which are, respectively, the advected paint amount from the previous keyframe t_0 and next keyframe t_N, to the intermediate frame t_i. We define the target paint amount map T_i as the pixel-wise maximum of both paint amount, where the maximum function acts as a union operator:

$$T_i = \max\left(A_i^0, A_i^N\right)$$

At the end of paint amount propagation, we have a target paint amount map for each intermediate frame.

5. Intermediate Frame Generation

The final step of our algorithm involves selecting strokes from the candidate stroke lists to compose a frame that best matches the target paint amount map. Each candidate stroke is rated using a scoring system (Section 5.2), and strokes that score above an artist-defined threshold are selected.

The score associated with a stroke must consider the strokes already selected for the intermediate frame. This is because the final colour is determined by a blending process from the paint simulator. Therefore, the order of strokes is important, and a stroke cannot be evaluated on its own. Typical animations in our tests contain more than 100 strokes. Exhaustively testing all possible stroke combinations, which involves exploring all subsets of candidate strokes, is intractable. We design the intermediate frame generation algorithm to select a subset of candidate strokes in a greedy fashion. The algorithm randomly picks a candidate stroke and selects pieces of this stroke to add to the intermediate frame according to the score along its curve. The process ends when there are no more candidate strokes for the current intermediate frame.

5.1. Stroke ordering

To define the intermediate frame candidate stroke list, we merge the two candidate stroke lists and define a total order on the resulting list. When mixing strokes from different keyframes, stroke ordering has a strong impact on the output appearance, as the colour blending between strokes can produce unexpected results (see Figure 4 for example). Therefore, special care needs to be taken to define the order of strokes during intermediate frame generation.

The way each stroke acts on the canvas depends on the amount and colour of paint in the brush before drawing the stroke. A stroke

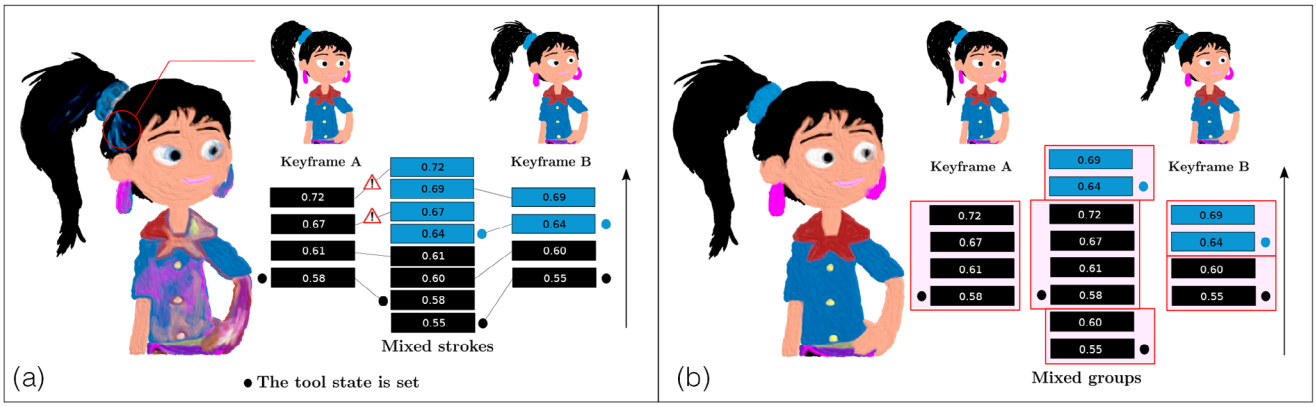

Figure 4: *A comparison between the ordering of candidate strokes based on their individual drawing positions (left) versus the ordering of stroke groups based on their group positions (right). Sorting strokes by their drawing positions can impact the tool state used for painting, resulting in differences in colour and paint amount during the simulation. Keyframes are shown at time t_{10} and t_{20}, and the intermediate frame is shown at time t_{11}.*

can start with a tool that is freshly refilled with colour, freshly cleared for smudges or keeping its state from the end of the previous strokes. In the latter case, the tool's colour depends on the previous brush/canvas interaction during the paint simulation. We refer to the 'clear', 'refill' or 'keep' state as the *tool state* of the stroke.

Each stroke of a keyframe has a drawing order index, which corresponds to its index in the list of stroke. To ease the mixing of two stroke list, the stroke ordering process assigns a drawing position by mapping the drawing indices to [0,1], independently in each stroke list. Sorting candidate strokes by drawing positions is not sufficient to merge the two lists since strokes with a 'keep' tool state have their colour changed (Figure 4a). Hence, we define stroke groups to create coherent drawing batches within each list, and merge the two lists while preserving the groups and their order. To achieve this, we assign each group a unique ordering index based on the drawing order position of the first stroke within the group, considering groups from both candidate stroke lists. The drawing order of strokes within a group follows that of the keyframe, and the tool state of these strokes acts in the same way as in the keyframe (Figure 4b). This process defines the merged list of candidate strokes and establishes the drawing order for the intermediate frame.

We provide an auto-grouping mechanism based on the tool state of the strokes. Each stroke with a tool state of 'clear' or 'refill' starts a new group, whereas strokes with a 'keep' state do not. If necessary, the artist can manually adjust the auto-grouping and group indices to finely tune how the keyframe groups are interleaved, although none of the examples shown in the paper utilise this feature.

During candidate stroke evaluation, strokes are randomly selected from the merged list and inserted in the intermediate frame list of strokes for evaluation. Stroke insertion follows the established drawing order, and the tool state of the first stroke of a group within the intermediate frame list is set to match the state of the first stroke of the corresponding group from the merged list. Consider the picked stroke S: If S is inserted as the first stroke in its group within the intermediate frame list, its tool state is set to the state of the first stroke in its group from the merged list. If any other stroke is inserted in the first position of this group thereafter, the tool state of S is reset to its initial value. The algorithm then evaluates S as described in the following section. If S is rejected, we revert the tool state of the first stroke in its group within the intermediate frame list. This process is detailed in Algorithm 1.

5.2. Candidate stroke evaluation

Let us consider a stroke, randomly picked from the merged list of candidate strokes. We define a score to represent how well the stroke improves the current intermediate frame's paint amount, taking into account the target paint amount map.

Stroke evaluation. To determine the stroke's score, the evaluation process first computes the score of each pixel whose paint amounts have been modified by the addition of the picked candidate stroke. The distance to the expected paint amount $T(p)$ before and after picking the candidate stroke is computed using current paint amount $C(p)$ and new paint amount $U(p)$:

$$\Delta_C(p) = |C(p) - T(p)|$$
$$\Delta_U(p) = |U(p) - T(p)|$$

Then the variation to the expected paint amount gives the pixel score:

$$\Delta(p) = \Delta_C(p) - \Delta_U(p) + \lambda$$

A positive score means that the changes in paint amount between C and U get closer to the target paint amount map T. We add a bonus λ to the score if the pixel is not covered yet while paint is expected, *i.e.* $C(p) = 0 \wedge T(p) > 0$. The value of λ depends on the underlying paint system, we set it empirically to two times the typical paint amount of covered pixels in our paint simulator.

Stroke cutting. Computing the stroke score as the sum of pixel scores to accept strokes with a positive score and reject others can lead to excessive paint overflow in the results (Figure 5c). A stroke

Algorithm 1. Intermediate frame stroke insertion.

```
Input/Output:
vector<Stroke> pool;      // Merged list of candidate strokes
vector<Stroke> c;         // Intermediate frame list of strokes
begin
    // Pick a stroke and remove it from merged list
    Stroke s = randomPick ( pool );
    Group g = getGroup ( s );
    Stroke f_p ← getFirstStrokeInGroup ( pool, g );
    // Get the first stroke in c if g is already present, return an invalid stroke if not.
    Stroke f_c ← getFirstStrokeInGroup ( c, g )// Save tool state, if reset needed anytime
    s.oldToolState ← s.toolState;
    // If picked stroke is inserted at first position of its group in c
    if (¬ valid ( f_c )) ∨ ( valid ( f_c ) ∧ f_c.drawOrder > s.drawOrder ) then
        if valid ( f_c ) then
            // Reset f_c tool state since it isn't the first anymore
            f_c.toolState ← f_c.oldToolState;
        s.toolState ← f_p.toolState;
    c.insert ( s );
    // Evaluate new frame list of strokes, and reset the list and state if the score is not improved
    c.newScore ← evaluate ( c );
    if c.newScore < c.oldScore then
        c.remove ( s );
        if valid ( f_c ) then
            // f_c returns to the first position of the group, set its state from f_p
            f_c.toolState ← f_p.toolState;
    else
        c.oldScore ← c.newScore
```

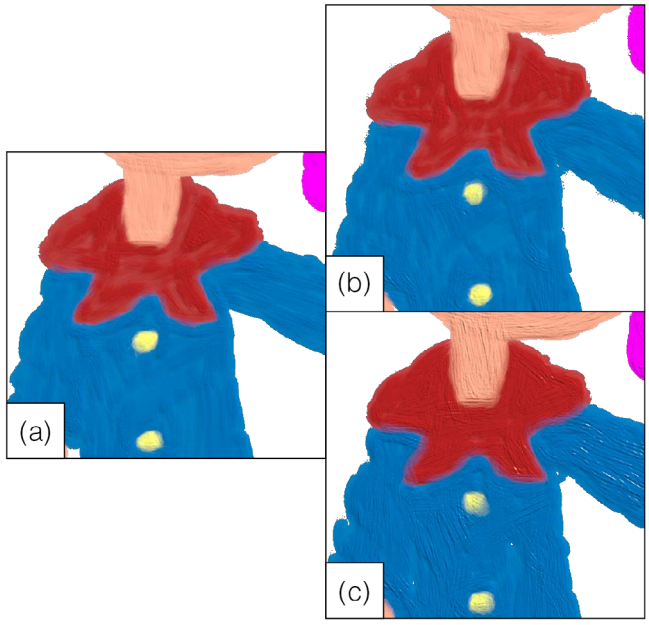

Figure 5: *A comparison of the frame generation with and without stroke cutting. (a) shows one of the two inputs keyframes used for stroke generation, while (b) and (c) show the results of stroke generation for the same intermediate frame. In (b), with stroke cutting enabled, the paint amount is more consistent with the keyframe. In (c), with stroke cutting disabled, there is an overflow of paint in some regions.*

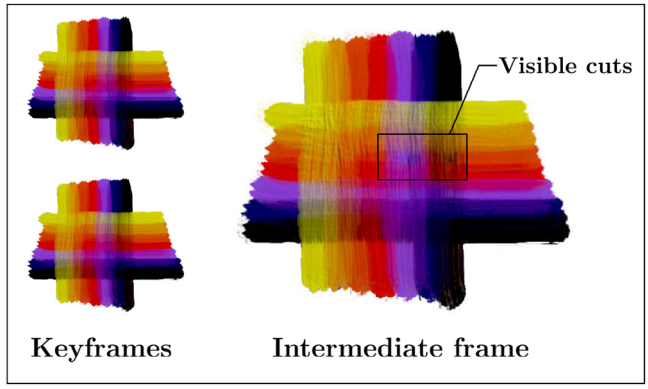

Figure 6: *Impact of stroke cutting on the appearance of generated frames when using keyframes with complex coloured regions. For illustration purpose, the two keyframes are the same.*

may have some pieces with a high score while others have a low score. To better match the target paint amount of the canvas, we propose a mechanism to cut a stroke into pieces according to the scores of pixels that belong to the rendered surface of the stroke. Cutting strokes has no perceptible impact on uniformly painted regions, as shown in Figures 5(a) and (b). This is not the case for regions painted with different colours (Figure 6).

We associate per-pixel scores to the curve parameters of the stroke. The stroke curve is then divided into a fixed number of segments (*i.e.* 100) corresponding to evenly distributed parameter ranges along the curve. Each segment represents a quadrilateral region of the stroke on the canvas. We compute the score of each segment as the sum of all pixel scores within its quadrilateral region and retain only those segments with a positive score.

The selected stroke segments are added to the stroke list of the intermediate frame as new strokes. The first segment takes the tool state of the stroke, while the tool state of subsequent segments is set to 'keep' state.

Smudge effects. Smudge strokes are used by the artist to mix colours and smooth out colour changes on the canvas. These strokes correspond to a tool state set to 'clear' and do not add additional paint to the canvas, unlike filled strokes. Therefore, it is not

necessary to cut them in order to ensure a consistent distribution of paint on the canvas. While cutting filled strokes can enhance overall paint distribution, cutting smudge strokes offers limited benefits and may introduce more visible discontinuities on the canvas.

Nevertheless, our experiments show that accepting all smudge strokes from both keyframes leads to excessive smoothing in the intermediate frames. Therefore, we have designed a specific selection mechanism for smudge strokes that avoids cutting. We filter candidate smudge strokes to consider as candidate only a subset of them from the previous and next keyframe, based on the timeline position of the intermediate frame being generated. To ensure that the closer the intermediate frame is to a keyframe the more smudge strokes from this keyframe are likely to be considered as candidates, we select smudge strokes according to the ratio ρ:

$$\rho = \frac{t}{N}$$

and take $(1-\rho)\cdot\kappa_0$ smudge strokes from the previous keyframe and $\rho\cdot\kappa_N$ smudge strokes from the next keyframe. κ_i is the number of smudge strokes in the keyframe at time t_i.

6. Extrapolation of Motion Field

In the examples presented in this paper, we demonstrate three different sources of motion fields for generating painted animations using our method: video, 3D scenes and artist-drawn motion field.

Dense optical flow extraction from a video defines the motion of each pixel in the image domain. This motion field can be directly used in our method. It provides information about the motion of the entire scene, though splitting this motion into layers can be challenging.

The motion field from a 3D scene only defines motion within the rendered area of the animated objects, making it easy to split into layers. However, strokes that extend beyond the objects' area may not be properly advected, requiring the motion field to be extrapolated as explained later.

We propose a simple method to obtain a motion field by using parametric curves drawn by the artist to represent motion across frames. Each parametric curve is divided into segments by matching their parameters to the timestamps of the intermediate frames in the sequence. Each segment defines the motion of the underlying pixels, but this sparse motion field also requires extrapolation. Authoring complex motion field from drawings and annotations is beyond the scope of this paper. Motion field authoring approaches such as Hu *et al.*[HXF*19] could be directly integrated into our approach.

A sparse or incomplete motion field needs to be extrapolated to provide a motion vector for every pixel, ensuring proper advection of painted strokes even in regions without initial motion information. To this end, we use a bi-harmonic diffusion method adapted from Baster *et al.*[BBA09]. The bi-harmonic diffusion is based on minimizing gradient differences within a mesh that discretizes the sparse motion field, producing a smooth extrapolation of known vectors. First, the extrapolation process builds a triangular 2D mesh T that tessellates the image plane. Then, it assigns known motion

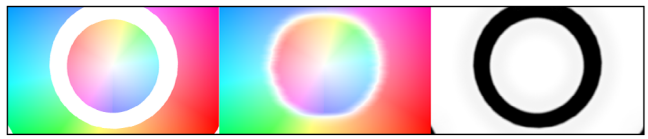

Figure 7: *Extrapolating a partial motion field (right) results in a global motion field where each pixel is assigned a motion vector (middle) along with a confidence value (right).*

vectors to the nearest vertices. To compute the motion vectors for the unassigned vertices, the process minimizes the Jacobian of the motion vectors of adjacent triangles in the mesh. This corresponds to minimizing the following energy:

$$\sum_{j,k\in\zeta(T)} \|J(j)-J(k)\|^2 + \epsilon \sum_{i\in F} \|J(i)\|^2,$$

where

$$J(f) = \begin{bmatrix} V(f_a)-V(f_c) \\ V(f_b)-V(f_c) \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \omega(f_a) \\ \omega(f_b) \\ \omega(f_c) \end{bmatrix}$$

with F the set of faces, V the vertices and ω the motion vectors of the vertices, *i.e.* the variables of the optimization. Each face f has three indices, f_a, f_b and f_c, which index V and ω. $\zeta(T)$ represents the pairs of all faces in T that share an edge.

The motion vector for each pixel in the image plane is computed using barycentric interpolation from the motion vectors of the vertices of T. In addition, we define a confidence value for each motion vector. To compute this value, the process first creates a binary mask where each pixel is set to one if it corresponds to an input motion vector in the motion field and zero if it was extrapolated. Then, a Gaussian filter is applied to smooth the transition between the original and extrapolated areas (see Figure 7). This confidence value is used as blending weights σ in the stroke regularisation process (Section 4.1).

7. Implementation and Results

We implemented our prototype in C++/OpenGL using RadiumEngine [MRB*21]. All of our illustrations are rendered using our own implementation of a paint simulator inspired by Baxter *et al.*[BWL04] and Chu *et al.*[CBWG10]. This paint simulator computes bi-directional paint exchanges between the brush and the canvas, on the GPU.

We extract the rendered motion field using Blender's AOV rendering [Com18], employ the Triangle library's Python wrapper [She96, R*20] and use our implementation of bi-harmonic diffusion to obtain the interpolated motion field as a pre-processing step.

Regarding the propagation of keyframe strokes, we advect them on the GPU using compute shaders. The rigid transform registration and the Levenberg–Marquardt optimisation are implemented using the Eigen library [GJ*10].

The computational cost of our method is directly proportional to the number of candidate strokes used. Our method utilises a greedy

Table 1: *Measurements in seconds taken when generating the beach-ball animation (see supplemental data) using keyframes at t_{10} and t_{20}, generating nine intermediate images consisting of approximately 600 strokes each. Computation times were measured from a workstation equipped with an Intel® Xeon® CPU E5-1630 v4@3.70GHz and an NVIDIA GeForce GTX 1080.*

Step	Per image (s)	Per stroke (s)
Generation	≃1140	
- Propagation	≃1.4	
* *Stroke advection*	≃0.0839	
* *Regularisation*	≃0.6	≃0.0007
* *Paint advection*	≃0.0182	
- Ordering	<0.0000	
- Evaluation	≃1122	
* *Paint simulation*	≃1026	≃1
* *Score*	≃84	≃0.0889
* *Cutting*	≃6.5	≃0.0068

stroke-picking algorithm, which involves rendering a list of strokes that expands at each step of the evaluation process. As a result, the simulation cost increases linearly with each accepted evaluation of a candidate stroke. For instance, keyframes composed of about 100 strokes require only a few minutes per intermediate frame to render. However, for more complex keyframes consisting of a thousand strokes, the rendering time can extend to several minutes per intermediate frame (Table 1).

We provide examples of generated animations in the supplemental data accompanying this paper. Depending on the scene's complexity, the artist decomposes the scene into multiple layers, which reduces the number of strokes required per keyframe and significantly improves computational efficiency. Each layer has its own keyframes and motion field. To guide the drawing of keyframes, the artist typically uses a background animation, either from 3D rendering, from video or from an animated storyboard. To ensure compelling intermediate frames, the advection from one keyframe to the next should align their content. In practice, relying solely on a pose-to-pose approach can make achieving this alignment challenging, as the artist must anticipate the motion applied during advection. To assist with motion anticipation, we find useful to advect the first keyframe to predict where its content will appear on the second keyframe. The advected result can then serve as a guide for the artist when drawing the second keyframe.

Our method exhibits a level of temporal coherence comparable to paint-on-glass animations, such as *Loving Vincent* [KW17]. However, this temporal *in*-coherence have fuzzy contours, with strokes only following the motion of the scene, and some frame to frame flickering. However, the main difference between our approach and traditional under-the-camera technique is that our method utilises keyframes whilst manual approaches typically involve repainting only the moving parts of the frame over the previous frame. Our generative scheme uses random stroke picking to create slight variations in each intermediate frame. This is useful for breaking temporal continuity and enforcing the flatness described by Bénard *et al.* [BBT11], which breaks the perception of synthetic moving strokes.

Our method was tested on a variety of scenes with challenging animation, such as occlusions, camera zoom and boil line effect. As

Figure 8: *This intermediate frame was generated using dense optical flow from the Sintel video, employing the OpenCV's Farnebäck implementation [Far03]. Due to the low quality of the motion field, strokes around the character are distorted by its movement.*

stated in Section 3, we assume that animations involving multiple objects, and occlusions, are decomposed into layers. We evaluated our method on a scene involving a zoom effect, which typically presents challenges due to rapid changes in scale and perspective that can lead to unfilled areas. Our approach leverages strokes from keyframes to ensure shape consistency in each intermediate frame. While this approach assumes that the keyframes capture the necessary scaling information, it aligns with the fundamental principles of pose-to-pose animation, where keyframes are defined by key positions and extreme poses [Wil09]. Our algorithm effectively adapts to the changing scale of the scene, leveraging the bi-directional motion field to handle zooming in and out seamlessly. Our method can also handle animations with no explicit motion between the frames, resulting in line boil effects. In this case, our algorithm generates intermediate frames based on two keyframes that represent the same state of the scene. By using random stroke picking, our algorithm creates slight variations of these keyframes for each intermediate frame. This produces line boils, similar to what we observe in traditional hand-drawn animation.

8. Limits and Perspectives

The quality of our results largely depends on the motion field provided as input. Rendered motion fields capture instantaneous speed motion vectors accurately but result in non-invertible motion fields as they do not account for acceleration. The problem can be overcome by using Runge–Kutta differential equations [But87] to get a better approximation of the rendered motion [Ell18]. Moreover, the motion field only capture the motion within object shapes, necessitating extrapolation for regions outside these shapes. Optical flows extracted from video can have poor quality depending on the extraction method used (Figure 8). Our method is compatible with approaches that enhance optical flow to improve the resulting video animation [DBH19].

Intermediate frames generated by our system may contain empty regions where paint is missing. While our paint advection scheme effectively preserves the paint distribution from the corresponding

keyframes, the advection of strokes can sometimes lead to areas that lack candidate strokes to fill them.

The performance of our method is currently not suitable for real-world usage, primarily due to the greedy algorithm that requires numerous paint simulations, which cannot be cached in memory. While keyframe creation is performed in real time by the artist, the computation of intermediate frames is still an offline process in its current form. We believe that achieving generation times on the order of seconds, while maintaining the same level of quality in results, will greatly benefit our workflow and the animation industry as a whole. We describe some avenues for improvement.

Genetic algorithms could potentially provide improved performance compared to greedy approaches. The intermediate frames generation could work as follows: First, the algorithm selects a subset of candidate strokes, which defines a solution for the intermediate frame, and computes associated costs according to the target paint amount. Then, the subset undergoes a mutation step, and the subset is re-evaluated. We anticipate convergence after a few mutations. This approach would allow the evaluation of the state of intermediate frames at each mutation step instead of at each stroke selection, significantly reducing the cost of the paint simulation.

To improve our method, a more advanced approach to mixing and grouping keyframe strokes could be explored. One potential lead is to use machine learning techniques to identify strokes that share similar semantic properties, such as those depicting shape contours or elements in the foreground or background. This could allow strokes to be processed differently depending on the groups they belong to, leading to more accurate and efficient animation.

9. Conclusion

The proposed method offers the flexibility of the pose-to-pose workflow combined with the look and feel of paint-on-glass animations. The approach is compatible with both full 2D and mixed 2D-3D animation workflows. We demonstrate that motion fields require specific processing to be usable for interpolating keyframes in a painterly style. The necessary processing depends on whether the motion field is rendered, estimated or hand-drawn, to be readily usable by the artist. Our method also includes a novel approach to frame interpolation that considers filling constraints. This represents a step forward in bringing the traditional animation style to computer-generated animation, and opens up new possibilities for artists.

Acknowledgements

This work was funded by the Grant ANR-19-CE38-0009-01 (ANR project *Structures*). We thank Pascal Barla, Mathias Paulin and Nicolas Mellado for their valuable comments.

References

[BBA09] Baxter W., Barla P., Anjyo K.: N-way morphing for 2D animation. *Computer Animation and Virtual Worlds 20*, 2-3 (2009), 79–87. https://doi.org/10.1002/cav.310.

[BBS*13] Bassett K., Baran I., Schmid J., Gross M., Sumner R. W.: Authoring and animating painterly characters. *ACM Transactions on Graphics 32*, 5 (Oct. 2013). https://doi.org/10.1145/2484238.

[BBT11] Bénard P., Bousseau A., Thollot J.: State-of-the-art report on temporal coherence for stylized animations. *Computer Graphics Forum 30*, 8 (2011), 2367–2386. https://doi.org/10.1111/j.1467-8659.2011.02075.x.

[BCK*13] Bénard P., Cole F., Kass M., Mordatch I., Hegarty J., Senn M. S., Fleischer K., Pesare D., Breeden K.: Stylizing animation by example. *ACM Transactions on Graphics 32*, 4 (July 2013). https://doi.org/10.1145/2461912.2461929.

[BFV21] Barroso N., Fondevilla A., Vanderhaeghe D.: Automatic intermediate frames for stroke-based animation. In *Journées Françaises d'Informatique Graphique (JFIG 2021)* (Sophia Antipolis, France, Nov. 2021). https://hal.science/hal-03454288.

[BKLP16] Bai Y., Kaufman D. M., Liu C. K., Popović J.: Artist-directed dynamics for 2D animation. *ACM Transactions on Graphics 35*, 4 (July 2016). https://doi.org/10.1145/2897824.2925884.

[BSFG09] Barnes C., Shechtman E., Finkelstein A., Goldman D. B.: PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics 28*, 3 (July 2009). https://doi.org/10.1145/1531326.1531330.

[But87] Butcher J. C.: *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods*. Wiley-Interscience, Chichester, 1987.

[BWL04] Baxter W., Wendt J., Lin M. C.: IMPaSTo: A realistic, interactive model for paint. In *NPAR'04: Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering* (New York, NY, USA, 2004), Association for Computing Machinery, pp. 45–148. https://doi.org/10.1145/987657.987665.

[CBWG10] Chu N., Baxter W., Wei L.-Y., Govindaraju N.: Detail-preserving paint modeling for 3D brushes. In *NPAR'10: Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering* (New York, NY, USA, 2010), Association for Computing Machinery, pp. 27–34. https://doi.org/10.1145/1809939.1809943.

[Com18] Community B. O.: *Blender—a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam. (2018).

[CZBB20] Chen J., Zhu X., Bénard P., Barla P.: Stroke synthesis for inbetweening of rough line animations. In *Pacific Graphics Short Papers, Posters, and Work-in-Progress Papers* (2020), S.-H. Lee, S. Zollmann, M. Okabe and B. Wuensche (Eds.), The Eurographics Association. https://doi.org/10.2312/pg.20201233.

[DBB*17] Dvorožňák M., Bénard P., Barla P., Wang O., Sýkora D.: Example-based expressive animation of 2D rigid

bodies. *ACM Transactions on Graphics 36*, 4 (July 2017). https://doi.org/10.1145/3072959.3073611.

[DBH19] Delanoy J., Bousseau A., Hertzmann A.: Video motion stylization by 2D rigidification. In *Expressive'19: Proceedings of the 8th ACM/Eurographics Expressive Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering* (Goslar, DEU, 2019), Eurographics Association, pp. 11–19. https://doi.org/10.2312/exp.20191072.

[DP73] Douglas D. H., Peucker T. K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization 10*, 2 (1973), 112–122. https://doi.org/10.3138/FM57-6770-U75U-7727.

[EBB23] Even M., Bénard P., Barla P.: Non-linear rough 2D animation using transient embeddings. *Computer Graphics Forum 42*, 2 (2023), 411–425. https://doi.org/10.1111/cgf.14771.

[Ell18] Ellsworth T. S.: *The Bird and The Fish: Motion Field-Based Frame Interpolation in the Context of a Story*. Brigham Young University, 2018.

[Far03] Farnebäck G.: Two-frame motion estimation based on polynomial expansion. In *Image Analysis: 13th Scandinavian Conference, SCIA 2003 Halmstad, Sweden, June 29–July 2, 2003 Proceedings 13* (2003), Springer, pp. 363–370. https://doi.org/10.1007/3-540-45103-X_50.

[GJ*10] Guennebaud G., Jacob B.: Eigen v3. http://eigen.tuxfamily.org (2010).

[Hae90] Haeberli P.: Paint by numbers: Abstract image representations. In *SIGGRAPH'90: Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1990), Association for Computing Machinery, pp. 207–214. https://doi.org/10.1145/97879.97902.

[HE04] Hays J., Essa I.: Image and video based painterly animation. In *NPAR'04: Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering* (New York, NY, USA, 2004), Association for Computing Machinery, pp. 113–120. https://doi.org/10.1145/987657.987676.

[Her98] Hertzmann A.: Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH'98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), Association for Computing Machinery, pp. 453–460. https://doi.org/10.1145/280814.280951.

[HTT*20] Hauptfleisch F., Texler O., Texler A., Krivánek J., Sýkora D.: StyleProp: Real-time example-based stylization of 3D models. *Computer Graphics Forum 39*, 7 (2020), 575–586. https://doi.org/10.1111/cgf.14169.

[HXF*19] Hu Z., Xie H., Fukusato T., Sato T., Igarashi T.: Sketch2VF: Sketch-based flow design with conditional generative adversarial network. *Computer Animation and Virtual Worlds 30*, 3-4 (2019), e1889. https://doi.org/10.1002/cav.1889.

[JSL22] Jiang J., Seah H. S., Liew H. Z.: Stroke-based drawing and inbetweening with boundary strokes. *Computer Graphics Forum 41*, 1 (2022), 257–269. https://doi.org/10.1111/cgf.14433.

[JST*19] Jamriška O., Sochorová V., Texler O., Lukáč M., Fišer J., Lu J., Shechtman E., Sýkora D.: Stylizing video by example. *ACM Transactions on Graphics 38*, 4 (July 2019). https://doi.org/10.1145/3306346.3323006.

[JYF*20] Jing Y., Yang Y., Feng Z., Ye J., Yu Y., Song M.: Neural style transfer: A review. *IEEE Transactions on Visualization & Computer Graphics 26*, 11 (Nov. 2020), 3365–3385. https://doi.org/10.1109/TVCG.2019.2921336.

[KW17] Kobiela D., Welchman H.: *Loving vincent*. BreakThru Productions, Trademark Films. (2017).

[LYFD12] Lu J., Yu F., Finkelstein A., DiVerdi S.: HelpingHand: Example-based stroke stylization. *ACM Transactions on Graphics 31*, 4 (July 2012). https://doi.org/10.1145/2185520.2185542.

[Mar63] Marquardt D. W.: An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics 11*, 2 (1963), 431–441. https://doi.org/10.1137/0111030.

[MFXM21] Miyauchi R., Fukusato T., Xie H., Miyata K.: Stroke correspondence by labeling closed areas. In *2021 Nicograph International (NicoInt)* (2021), pp. 34–41. https://doi.org/10.1109/NICOINT52941.2021.00014.

[Mia21] Miailhe F.: *La traversée*. Gebeka Films. (2021).

[MRB*21] Mourglia C., Roussellet V., Barthe L., Mellado N., Paulin M., Vanderhaeghe D.: Radium-engine. Apache-2.0 license. https://doi.org/10.5281/zenodo.5101334 (2021).

[NIK10] Northam L., Istead J., Kaplan C. S.: Brush Stroke Ordering Techniques for Painterly Rendering. In *Computational Aesthetics in Graphics, Visualization, and Imaging* (2010), P. Jepp and O. Deussen (Eds.), The Eurographics Association. https://doi.org/10.2312/COMPAESTH/COMPAESTH10/059-066.

[OH11] O'Donovan P., Hertzmann A.: AniPaint: Interactive painterly animation from video. *IEEE Transactions on Visualization and Computer Graphics 18*, 3 (2011), 475–487. https://doi.org/10.1109/TVCG.2011.51.

[R*20] Rufat D.: Triangle. https://rufat.be/triangle/ (2020).

[SDC09] Sýkora D., Dingliana J., Collins S.: As-rigid-as-possible image registration for hand-drawn cartoon animations. In *NPAR'09: Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering* (New York, NY, USA, 2009), Association for Computing Machinery, pp. 25–33. https://doi.org/10.1145/1572614.1572619.

[She96] Shewchuk J. R.: Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*. Lecture Notes in Computer Science (May 1996), M. C. Lin and D. Manocha (Eds.), Springer-Verlag, vol. *1148*, pp. 203–222. From the First ACM Workshop on Applied Computational Geometry. https://doi.org/10.1007/BFb0014497.

[SHR17] Sorkine-Hornung O., Rabinovich M.: *Least-Squares Rigid Motion Using SVD*. Tech. Rep., ETH Department of Computer Science, Zürich, Switzerland, 2017.

[SJT*19] Sýkora D., Jamriška O., Texler O., Fišer J., Lukáč M., Lu J., Shechtman E.: StyleBlit: Fast example-based stylization with local guidance. *Computer Graphics Forum 38*, 2 (2019), 83–91. https://doi.org/10.1111/cgf.13621.

[SSGS11] Schmid J., Senn M. S., Gross M., Sumner R. W.: Overcoat: An implicit canvas for 3D painting. In *SIGGRAPH'11: ACM SIGGRAPH 2011 Papers* (New York, NY, USA, 2011), Association for Computing Machinery. https://doi.org/10.1145/1964921.1964923.

[TKK*22] Todo H., Kobayashi K., Katsuragi J., Shimotahira H., Kaji S., Yue Y.: Stroke transfer: Example-based synthesis of animatable stroke styles. In *SIGGRAPH'22: ACM SIGGRAPH 2022 Conference Proceedings* (New York, NY, USA, 2022), Association for Computing Machinery. https://doi.org/10.1145/3528233.3530703.

[VC12] Vanderhaeghe D., Collomosse J.: Stroke based painterly rendering. In *Image and Video-Based Artistic Stylisation. Computational Imaging and Vision*. P. Rosin and J. Collomosse (Eds.). Springer, London (2012), vol. *42*, pp. 3–21. https://doi.org/10.1007/978-1-4471-4519-6_1.

[WCHG13] Wang T., Collomosse J. P., Hunter A., Greig D.: Learnable stroke models for example-based portrait painting. In *British Machine Vision Conference, BMVC 2013, Bristol, UK, September 9-13, 2013* (2013), T. Burghardt, D. Damen, W. W. Mayol-Cuevas and M. Mirmehdi (Eds.), BMVA Press. https://doi.org/10.5244/C.27.36.

[WDK*12] Whited B., Daniels E., Kaschalk M., Osborne P., Odermatt K.: Computer-assisted animation of line and paint in Disney's paperman. In *SIGGRAPH'12: ACM SIGGRAPH 2012 Talks* (New York, NY, USA, 2012), Association for Computing Machinery. https://doi.org/10.1145/2343045.2343071.

[Wil09] Williams R.: *The Animator's Survival Kit–Revised Edition: A Manual of Methods, Principles and Formulas for Classical, Computer, Games, Stop Motion and Internet Animators*. Faber & Faber, Inc., London, UK, 2009.

[WNS*10] Whited B., Noris G., Simmons M., Sumner R. W., Gross M., Rossignac J.: BetweenIT: An interactive tool for tight inbetweening. *Computer Graphics Forum 29*, 2 (2010), 605–614. https://doi.org/10.1111/j.1467-8659.2009.01630.x.

[XWSY15] Xing J., Wei L.-Y., Shiratori T., Yatani K.: Autocomplete hand-drawn animations. *ACM Transactions on Graphics 34*, 6 (Nov. 2015). https://doi.org/10.1145/2816795.2818079.

[YSC*18] Yang W., Seah H.-S., Chen Q., Liew H.-Z., Sýkora D.: FTP-SC: Fuzzy topology preserving stroke correspondence. *Computer Graphics Forum 37*, 8 (2018), 125–135. https://doi.org/10.1111/cgf.13518.

[YXD*20] Yang L., Xu T., Du J., Zhang H., Wu E.: Brushwork master: Chinese ink painting synthesis for animating brushwork process. *Computer Animation and Virtual Worlds 31*, 4-5 (2020), e1949. https://doi.org/10.1002/cav.1949.

[ZZ11] Zhao M., Zhu S.-C.: Portrait painting using active templates. In *NPAR'11: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering* (New York, NY, USA, 2011), Association for Computing Machinery, pp. 117–124. https://doi.org/10.1145/2024676.2024696.

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Supporting Information