DOI: 10.1111/cgf.14946
Pacific Graphics 2023
R. Chaine, Z. Deng, and M. H. Kim
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 42 (2023), Number 7

Efficient Interpolation of Rough Line Drawings

J. Chen[1], X. Zhu[1], M. Even[2], J. Basset[2], P. Bénard[2] and P. Barla[2]

[1]Zhejiang University of Technology, China [2] Inria, Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, France

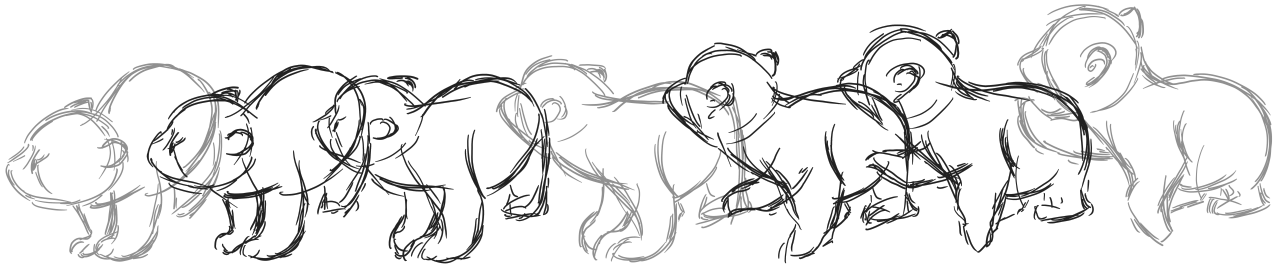


Figure 1: *Our interpolation technique takes as input a series of rough vector key drawings (shown in light gray), and produces intermediate drawings (in black) that preserve the input drawing style(s). It is based on an interactive stroke distribution synthesis algorithm that introduces a minimum amount of temporal artifacts (here with the* $\{3,3,0.05\}$ *strategy).*

Abstract

In traditional 2D animation, sketches drawn at distant keyframes are used to design motion, yet it would be far too labor-intensive to draw all the inbetween frames to fully visualize that motion. We propose a novel efficient interpolation algorithm that generates these intermediate frames in the artist's drawing style. Starting from a set of registered rough vector drawings, we first generate a large number of candidate strokes during a pre-process, and then, at each intermediate frame, we select the subset of those that appropriately conveys the underlying interpolated motion, interpolates the stroke distributions of the key drawings, and introduces a minimum amount of temporal artifacts. In addition, we propose quantitative error metrics to objectively evaluate different stroke selection strategies. We demonstrate the potential of our method on various animations and drawing styles, and show its superiority over competing raster- and vector-based methods.

CCS Concepts

• ***Computing methodologies*** → ***Non-photorealistic rendering; Animation;***

1. Introduction

Hand-made 2D line animations have the power to convey motion in a particularly expressive manner, at the expense of an extremely labor-intensive process. A line animation is traditionally produced through at least two stages: a *rough* stage, where sketched strokes are drawn at relatively distant keyframes, giving the gist of the motion; and a *clean* stage, where clean line drawings are produced between keyframes relying on the rough drawings for consistency, yielding smooth lines drawn at a high frame rate that convey a fluid motion. The focus of this paper is on the interpolation of rough line drawings at a high frame rate. It may be used as a form of pre-visualization for motion design prior to the clean stage, to give the animation a sketchy look instead of a clean look for artistic purposes, or even to augment a collection of sketchy drawings. We target an interactive solution to grant sufficiently fast interpolation feedback when modifying the keyframes or their timing and spacing, which makes our approach compatible with recent computer-assisted 2D animation systems (e.g., [XWSY15, EBB23]).

What makes rough line inbetweening a difficult problem is that rough key drawings (i.e., line drawings in keyframes) are located far apart in time and space. Moreover, they are usually made of a different number of strokes with different lengths and at different locations along the depicted shape. An additional difficulty arises due to the sketchy nature of rough drawings: the production of rough animations at high frame rates is prone to visual artifacts that may disturb the perception of the underlying motion, such as popping, flickering, and residual motion. Precisely defining, measuring, and controlling those artifacts represents an extra challenge.

As detailed in Section 2, most existing computer-aided inbetweening techniques work on clean line drawings provided at close keyframes in time (i.e., tight inbetweening [WNS*10]). Since the key drawings do not differ dramatically in terms of shape, and the number of lines is similar from one keyframe to the next, inbetweening is performed directly on clean lines. This requires solving two problems: a *correspondence* problem, where each line in a key drawing must be matched to a line in the next key drawing; and an *interpolation* problem, where a single clean line must be generated at each intermediate frame that interpolates a pair of matched lines. These two problems must be restated in the case of rough drawings inbetweening: correspondences must be established between groups of strokes representing the same (part of a) shape, and interpolation must be performed between heterogeneous groups of strokes. In the rest of this paper, **we only address the interpolation problem**, assuming groups of strokes have already been matched with each other with existing techniques [XWSY15, EBB23].

To guide the design of our interpolation algorithm, we analyzed several rough 2D animations. Our main observation is that each drawing necessarily uses a slightly different set of strokes, as it would be extremely tedious for an artist to do otherwise. Even in the case of static scenes, artists routinely introduce variations in strokes to keep the animation alive, a technique called "faux-fixe" in the field. We thus make the crucial design choice that each interpolated drawing should be synthesized from scratch, in order to match the look-and-feel of rough hand-made 2D animations.

Formally, we consider the strokes of a rough key drawing as a vectorial texture that follows an underlying (but unspecified) contour. In contrast with methods that synthesize strokes patterns (e.g., [LGH13, MWLT13, TWZ22]), our approach does not attempt to reproduce relationships among strokes (proximity, orientation, etc), as strokes in key drawings may follow highly irregular distributions and tend to significantly overlap. Instead, the goal of the interpolation process is to generate a set of strokes for each intermediate frame that (1) appropriately conveys the interpolated contour motion, (2) interpolates the stroke distributions of the key drawings, and (3) introduces a minimum amount of temporal artifacts. One possible solution for such a problem would be to rely on example-based raster synthesis (e.g., [JST*19, TFK*20, FKL*21]). However, as discussed in Section 5.1 and shown in Figure 11, ignoring the vectorial nature of strokes does not preserve stroke consistency, which leads to severe visual artifacts in the resulting animations.

We instead rely on vector strokes since stroke consistency is preserved by construction, and fine control is granted especially when interpolating key drawings with different styles and stroke attributes. In particular, this allows us to make sure that every frame is synthesized as if it had been drawn by hand, with each stroke drawn independently on the canvas. Our method works in two main steps (see Figure 2). In pre-process, we generate for each key drawing many *candidate strokes* via transformations of the input drawn strokes. At runtime, we select subsets of these candidate strokes to synthesize intermediate frames, carefully monitoring potential visual artifacts to reduce their occurrences. More specifically, we make the following three contributions:

- We introduce in Section 3 an interactive GPU-based stroke distribution synthesis framework for rough line drawings interpolation. It preserves stroke consistency by construction and conveys interpolated contours with a coherent style while reducing visual artifacts. It is also easily art-directable, allowing artists to strike different balances between visual goals through different stroke selection strategies.
- We describe in Section 4 several error metrics tailored to our stroke distribution synthesis framework of rough line animations. These metrics allow us to evaluate different stroke selection strategies in a more objective manner.
- We present in Section 6 a user study that evaluates the effectiveness of our stroke selection strategies, and compares our results with the temporal noise reduction technique of Noris et al. [NSC*11].

2. Previous work

2.1. Inbetweening

Vector drawings. Inbetweening for clean line animations provided as vector inputs has been investigated for decades [BW75]. These methods usually proceed in two steps which correspond to the two aforementioned problems: (1) they build *correspondences* between strokes of consecutive keyframes either by construction [DRvdP15] or semi-automatically [LCY*11, YBS*12, ZLWH16, BBM*16, CMV17, YSC*18, MFXM21]; (2) they *interpolate* each pair of matched strokes to generate intermediate frames [Ree81, SGWM93, Kor02, LWZ*04, WNS*10, Yan18]. The most recent method in this body of work [JSL22] additionally allows the specification and control of occlusions. Even though these methods work well for clean line drawings with some manual interventions, they do not extend easily to rough animations, as each keyframe may contain a very different number of strokes and varying spatial distributions.

To the best of our knowledge, only three previous vector animation techniques accept rough inputs. The drawing prediction system of Xing et al. [XWSY15] detects similarities and establishes correspondences between rough drawings to suggest future drawings with temporal and spatial consistency. However, it does not generate novel rough inbetween frames interpolating two key drawings.

Closer to our work, the method of Noris et al. [NSC*11] selects a subset of the frames of a manually-drawn (and hence "noisy") rough animation, and then matches and interpolates those to generate a less noisy animation. More precisely, it first estimates the global motion between each pair of selected drawings by performing As-Rigid-As-Possible (ARAP) registration [SDC09] between their rasterized distance fields, hence abstracting their topological differences. Then, it matches each stroke of the first drawing, deformed by the ARAP transformation, with the most similar stroke in the second one, and vice versa, potentially duplicating strokes. The intermediate frames are eventually generated with the stroke interpolation technique of Whited et al. [WNS*10]. This results in noise-free but overly smooth stroke trajectories that break the original hand-made appearance. To reintroduce a controllable amount of noise, these steps are repeated between the noise-free and original drawings, the stroke interpolation factor effectively modulating the noise level. In this work, we take the opposite approach: we start from a sparse set of rough drawings instead of a full rough

input animation, and we interpolate groups of strokes instead of individual strokes to avoid creating unnatural motion trajectories. We compare the merits and results of both approaches in Section 5.2.

The rough animation system of Even et al. [EBB23] proposes vectorial ARAP registration tools to directly match groups of strokes embedded into lattices between key drawings. Those are then deformed forward and backward in time using ARAP interpolation [ACOL00] and cross-faded to generate intermediate frames. This method additionally provides control over timing and continuity at keyframes, and enforces connectivity among different sets of strokes through hierarchical constraints. We follow the same approach to match and deform key drawings, but we want to replace the simple cross-fading that produces obvious ghosting artifacts. The problem left to solve is thus how to synthesize intermediate groups of strokes while keeping temporal noises under control.

Raster drawings. When processing raster drawings, the stroke correspondence step becomes an image registration problem, whereas the stroke interpolation step is turned into a morphing (deformation and blending) problem. The method of Baxter et al. [BBA09] registers the outer boundary of 2D shapes enclosing the drawings and uses uniform alpha-blending to mix the intermediate warped images, which leads to ghosting artifacts especially noticeable for rough line drawings. Zhu et al. [ZPBK17] extend this kind of approach to handle extreme shape deformations and topological changes, but their method involves an expensive numerical optimization that prevents its use in an interactive system.

Focusing on cartoon animations, learning-based techniques [LZLS21, SZY*21, CZ22] manage to inbetween colored keyframes with complex content. To register a pair of key images, they first estimate dense motion flows between them (e.g., [SLL*22]). They then generate an intermediate image by warping and non-linearly blending the input key frames (e.g., [HZH*22]), which produces ghosting artifacts similar to other raster methods. In addition, these methods were not trained on rough drawings, and would thus require a large training database that must encompass different drawing styles. In contrast, example-based stylization approaches [TFK*20, FKL*21] can transfer arbitrary styles to a video sequence with only a few stylized exemplars. Nevertheless, as shown in our experiments (Section 5.1), when applied to rough drawings, their results look more like "advected" textures than hand-drawn animations.

Closer to our input styles, the method of Arora et al. [ADN*17] interpolates concept sketches using a multi-image dense matching algorithm tailored to line drawings, and a non-linear alpha-blending scheme based on the confidence of the matching. Even though non-linear blending successfully reduces ghosting artifacts, it cannot reproduce the distinctive appearance of rough animations where stroke distribution varies from frame to frame.

2.2. Strokes distribution synthesis

Raster texture synthesis. Since a rough drawing can be seen as a distribution of strokes, it is tempting to represent it as a texture. Non-parametric raster texture synthesis [WLKT09] has been extensively used to transfer the style of an exemplar to images [HJO*01, FJL*16, SJT*19] and animations [BCK*13, FLJ*14, DLKS18, JST*19]. Yet, rough drawings have a very specific style which requires preserving the spatial continuity of strokes. This constraint turns out to be overly challenging for those methods, as demonstrated by our experiments (Section 5.1).

Vector pattern synthesis. To ensure the integrity of the strokes or of more general vector elements, example-based pattern synthesis approaches directly distribute discrete elements to meet some distribution constraints [BBT*06, IMIM08, HLT*09, LGH13, MWT11, MWLT13, TWZ22], potentially combined with continuous structures [RÖM*15, TWY*20]. Alternatively, procedural techniques [GAM*21, RSP22] can achieve similar patterns by grammars or procedures. However, both families of approaches only support repetitive patterns, at best with branching or graph-like structures. Consequently, they are not appropriate for representing stroke distributions with a large amount of overlap between strokes, large spatial variations and a strong underlying structure dictated by the depicted contours. In addition, only the work of Ma et al. [MWLT13] considers temporal variations but solely with repetitive movements.

The specific problem of rough drawing inbetweening does not necessitate to take into account precise stroke-to-stroke relationships such as proximity or orientation, as done in stroke pattern synthesis techniques. To the contrary, maintaining precise stroke patterns through interpolation would put too much emphasis on the motion of strokes themselves, whereas our goal is to communicate the motion of the underlying contour. In this work, our objective is instead to synthesize drawings that interpolate ensemble properties of stroke distributions: their spatial density (i.e., how many strokes overlap along the depicted contour) and their total stroke length. In addition, we provide a simple way to interpolate stroke attributes (e.g., thickness) in case they vary along a contour.

Stroke synthesis. A large body of work aims at synthesizing individual strokes in a given style taken from a single [HOCS02, KMM*02, LA15] or a set of exemplars [FTP03, LYFD12]. These methods reproduce complex drawing styles (e.g., curls), but they do not capture distributions of strokes. Relying on a large database of hand-made portrait sketches (about 8000 strokes per artist), Berger et al. [BSM*13] synthesizes new drawing of faces from photographs, capturing three stroke distribution properties: their amount of overlapping, their length and their type (i.e., simple or complex strokes). Ben-Zvi et al. [BBM*16] extends this approach to videos with temporal coherence. Similarly to those two methods, we generate sketches in a given drawing style, but we cannot rely on an external library of drawings since the inbetween frames that we synthesize must be drawn in the specific style used to depict the enclosing key drawings.

3. Rough inbetweening

Figure 2 shows an overview of our interpolation framework for rough line drawings, with notations summarized in Table 1. We assume that groups of strokes in key drawings have been registered beforehand (we use lattice-based ARAP deformation [SDC09, EBB23]), such that they can be warped in alignment.

In pre-process, a large amount of candidate strokes is generated

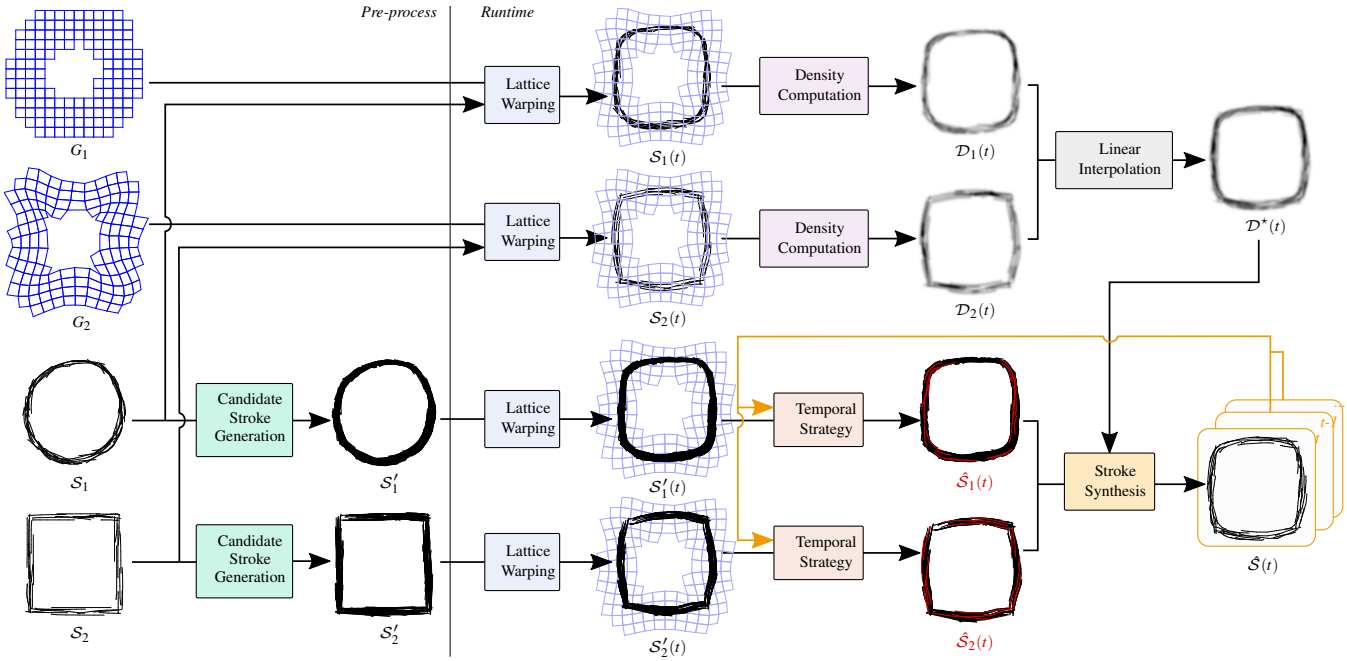


Figure 2: ***Overview of our stroke distribution synthesis method.*** *In pre-process (left), we generate a large number of candidate strokes for each input key drawing. Then, at runtime (right), key strokes are warped in alignment to generate an interpolated density map (top), which is used as a target density for the selection of relevant candidate strokes. All candidate strokes are warped in alignment, then progressively added until density criteria are met (bottom). Temporal strategies provide control over temporal artifacts by directly including (in red) some strokes and excluding (not shown) others.*

for each keyframe separately using linear perturbations with random parameters (Section 3.1). The rationale is that pre-generating strokes makes the synthesis process a mere question of selecting the most adequate candidates at each frame, which is fast enough to run interactively.

At runtime, we must first estimate the target stroke density at the current frame. This is done by warping key strokes (i.e., strokes in both keyframes, as shown at the top of Figure 2) with ARAP interpolation [ACOL00]. We then compute their blended density map on the GPU (Section 3.2). We also compute attribute maps (not shown) such as stroke thickness (i.e., their width) at this stage.

Candidate strokes must then be selected according to the target density map. We first warp all candidate strokes from both key drawings, then add them progressively to the synthesized drawing until the target density is met (Section 3.3). This process can also be customized using temporal strategies, which select candidates that should be directly included, excluded, or considered for synthesis according to previously synthesized frames (Section 3.4).

$\mathcal{S}_k$	set of strokes from the k^{th} key drawing
$\mathcal{S}'_k$	set of candidate strokes from the k^{th} key drawing
$\hat{\mathcal{S}}_k$	set of selected candidate strokes from the k^{th} key drawing
$\mathcal{D}_k$	density map of the k^{th} key drawing
$\mathcal{D}^\star$	target density map
$\mathcal{D}_S$	density map of a single stroke S
$\mathcal{C}(S)$	covered density for a single stroke S
$L(\mathcal{S})$	total length of all stokes in $\mathcal{S}$
$L^\star$	target total stroke length
$L^\star_{1/2}$	target stroke length ratio
Δ_d	minimum frame interval before a stroke may disappear
Δ_a	minimum frame interval during which a stroke must remain
β	amount of stroke perturbation

Table 1: ***Main notations****, from top to bottom: stroke sets, density maps, covered density, stroke lengths, and temporal parameters.*

3.1. Candidate strokes generation

Our stroke distribution synthesis algorithm relies on the availability of two sets of candidate strokes, one per key drawing to interpolate. Even though the generation of these candidate strokes is performed in pre-process, we provide an efficient solution so that new candidate strokes can be quickly re-generated when either of the key drawings is modified. As shown in Figure 3, candidate strokes are generated in three steps: (a) linear perturbations, (b) stroke sliding, and (c) density-based filtering.

Linear perturbations. Let us write $S \in \mathcal{S}$ a stroke from a key drawing (also called *key stroke* in the following). It is defined by a sequence of 2D points $\mathbf{x}_i, \forall i \in \{1, \dots, |S|\}$, where $|S|$ stands for the number of points in S. A candidate stroke S' is generated by copying S to $\mathcal{S}'$ while applying small linear perturbations to its po-

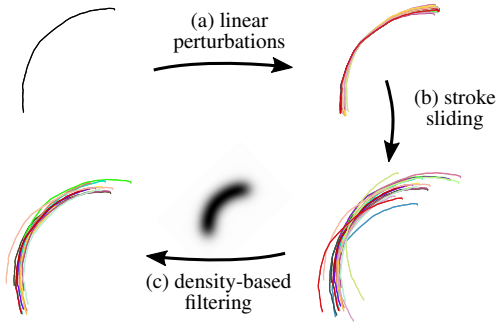


Figure 3: ***Generation of candidate strokes.*** *For each input key stroke (top left), (a) N candidate strokes are first generated by linear perturbations around the stroke median point, and then (b) slide along their own length with an orientation adaption to produce larger perturbations. Finally, (c) candidate strokes that do not sufficiently cover the density map of the corresponding key stroke (inset) are filtered out.*

sitions (see Figure 3(a)):

$$\mathbf{x}'_i = (\mathrm{I}+\mathrm{L})\cdot(\mathbf{x}_i - \mathbf{x}_m) + \mathbf{x}_m \tag{1}$$

where $\mathbf{x}_m$ is the median point of S (i.e., $m = |S|/2$), I is the identity matrix, and L is a matrix of random linear perturbations in the $[-0.1, 0.1]$ range. We use small perturbations around the median point $\mathbf{x}_m$ since each candidate stroke S' is further perturbed in the next step. We write N the total number of candidate strokes initially generated with linear perturbations (we use $N = 20$).

Stroke sliding. Each candidate stroke $S' \in \mathcal{S}'$ is then subjected to a new perturbation process, whereby it is made to slide along its own path. This is done by picking a random point $\mathbf{x}'_j \in S'$, and translating/rotating the whole stroke so that its center of gravity after sliding $\mathbf{x}'_m$ is $\mathbf{x}'_j$ and its tangent $\mathbf{t}(\mathbf{x}'_m)$ at that point is aligned with the tangent $\mathbf{t}(\mathbf{x}'_j)$.Formally, for all $\mathbf{x}'_i \in S'$, we compute:

$$\mathbf{x}'_i \leftarrow \mathrm{R}_{m\to j}(\mathbf{x}'_i - \mathbf{x}'_m) + \mathbf{x}'_j \tag{2}$$

where $\mathrm{R}_{m\to j}$ denotes the rotation that aligns $\mathbf{t}(\mathbf{x}'_m)$ with $\mathbf{t}(\mathbf{x}'_j)$. In practice, we restrict the randomly picked point $\mathbf{x}'_j$ to points whose arc-length distance to $\mathbf{x}'_m$ is less than a quarter of the stroke length, hence preventing the candidate stroke from overly deviating from the original stroke. We end up with a new set of N candidate strokes that are much more spread apart (see Figure 3(b)).

Density-based filtering. The last step ensures that the generated candidate strokes do not lie too far away from their original key stroke, and thus have sufficient chances to be selected during synthesis. This is done by computing a density map $\mathcal{D}_S$ for the corresponding key stroke S (as explained in the next section), and enforcing that a minimum proportion of stroke points (95% in our implementation) overlaps a non-zero region of the density map (see Figure 3(c)). We add to this set of filtered candidate strokes the original key stroke, which is necessary to produce coherent interpolations of key drawings, as explained in Section 3.3. We found empirically that near-circle strokes need more perturbations to yield valid candidate strokes, which are mostly offset versions of the key stroke, as rotation tends to make candidate strokes filtered out by the density map.

(a) *Input strokes* ***(b)*** *Density map* ***(c)*** *Thickness map*

Figure 4: ***Generation of density and thickness maps.*** *The density map (b) is computed by a Gaussian convolution of the input stroke paths (a). Stroke thickness is then rendered to a raster buffer, which is normalized by the density map to obtain the thickness map (c).*

3.2. Density and attribute interpolation

Before synthesizing a new drawing between a pair of keyframes, we must establish a target stroke density that will be achieved through synthesis. Such a density map is also used in pre-process for each individual key stroke to filter out generated candidate strokes that have little chance to be selected for synthesis.

All strokes of a key drawing are first warped to the current frame. Drawing inspiration from the painterly rendering technique of Lu et al. [LSF10], they are then rendered with additive blending on the canvas. To spatially average and spread their contribution, they are eventually convolved with a Gaussian kernel to yield a density map, as shown in Figure 4(b). One might think that such a convolution could be done in a pre-process for each key drawing; unfortunately, warping a stroke density map is not equivalent to computing a density map from warped strokes (except when warping boils down to a rigid transformation). The two resulting density maps $\mathcal{D}_1(t)$ and $\mathcal{D}_2(t)$ – one for each key drawing – are finally linearly interpolated according to the current interpolation time t to yield the target density map $\mathcal{D}^\star = (1-t)\mathcal{D}_1(t) + t\mathcal{D}_2(t)$ (see Figure 2). The same process is used to build attribute (e.g., thickness) maps. The only difference is that attributes are normalized by density after convolution to store averaged attribute values (see Figure 4(c)).

GPU implementation. In practice, density and attribute maps are simultaneously built on the GPU for efficiency, since they must be recomputed at each frame. The Gaussian convolution is approximated by a stamping process to allow fast density updates that are required by our synthesis algorithm (Section 3.3). The warped strokes are first resampled uniformly along their arc length (with a 2.5 pixels step size in our implementation) and stored in a Vertex Buffer Object along with their stroke indices; a Gaussian footprint texture is then rendered at each point location in parallel with additive blending. We use a Gaussian with a standard deviation that is 3/4 of the ARAP lattice cell size ℓ ($\ell = 32$ pixels by default), and a low image resolution for the density or attribute maps to speed up GPU read-back.

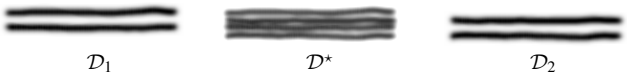


Figure 5: ***Local density issue.*** *The interpolated target density $\mathcal{D}^\star$ may be spread out when the key strokes and thus their corresponding densities $\mathcal{D}_1$ and $\mathcal{D}_2$ are not properly aligned. Without a global density criterion, a naive algorithm will tend to synthesize too many strokes to cover $\mathcal{D}^\star$.*

3.3. Stroke distribution synthesis

Equipped with a target density map $\mathcal{D}^\star$, the synthesis algorithm consists in selecting a subset of warped candidate strokes coming from two key drawings. We denote the two sets of warped candidate strokes $\mathcal{S}'_1$ and $\mathcal{S}'_2$. Since we target interactivity, a global optimization is not an option; we instead devise a fast greedy algorithm.

In its most naive version, this algorithm would randomly pick a stroke S' from either $\mathcal{S}'_1$ or $\mathcal{S}'_2$, check whether it sufficiently covers $\mathcal{D}^\star$; if so, update $\mathcal{D}^\star$ by subtracting the density covered by S', remove it from its set, and repeat this process until there is no remaining candidate stroke that can improve density coverage. This naive algorithm would give equal chances to all candidates which is not desirable for two reasons: (1) strokes with small density coverage could be inserted early on, preventing the selection of strokes with larger density coverage, sometimes even hampering convergence; (2) many strokes would be rejected before the density coverage criterion is met, which would be inefficient. To improve on both aspects, candidate strokes are sorted according to their covered density $\mathcal{C}(S')$, and the candidate stroke with the highest covered density in $\mathcal{S}'_1 \cup \mathcal{S}'_2$ is always selected. After each stroke selection, the target density map $\mathcal{D}^\star$ should be updated, the covered densities of all remaining candidate strokes must be recomputed, and the strokes re-ordered. Formally, the density $\mathcal{C}(S')$ covered by a stroke S' is the integral of the target density in the stroke footprint. Repeatedly evaluating this integral would be prohibitively expensive. However, since the size of this footprint is constant, $\mathcal{C}(S')$ is correlated to the density covered by the stroke vertices, and we can thus use the following crude, but very fast, approximation: $\mathcal{C}(S') \approx \sum_i \mathcal{D}^\star(\mathbf{x}'_i)$, which proved to be sufficient for stroke sorting. We accelerate this computation further by leveraging GPU parallelism (see below).

The second issue of the naive algorithm is its termination criterion: only relying on *local* density coverage might end up synthesizing too many strokes at intermediate frames. This can be illustrated with a simple example (see Figure 5). When strokes in key drawings are slightly offset (which is commonplace in rough drawings), the interpolated target density map $\mathcal{D}^\star$ at an intermediate frame covers a larger amount of space than $\mathcal{D}_1$ or $\mathcal{D}_2$, albeit with a reduced magnitude. As a result, the naive algorithm will tend to synthesize twice the number of strokes. To solve this issue, we need to consider a *global* density criterion: we must ensure that the total density of synthesized strokes is close to the total density in the initial target density map $\mathcal{D}^\star$. Denoting $I[\mathcal{D}] = \int \mathcal{D}(\mathbf{x})d\mathbf{x}$ the integrated density, we need to enforce the following constraint:

$$\sum_{\hat{S}\in\hat{\mathcal{S}}} I[\mathcal{D}_{\hat{S}}] \approx I[\mathcal{D}^\star], \tag{3}$$

where $\hat{\mathcal{S}}$ denotes the set of synthesized strokes and $\mathcal{D}_{\hat{S}}$ is the density map for a single synthesized stroke $\hat{S}$. Unfortunately, such a pixel-wise comparison would be too time-consuming in practice. Since the density map of a stroke $\mathcal{D}_{\hat{S}}$ is correlated with its length, we approximate that constraint by $L(\hat{\mathcal{S}}) \approx (1-t)L(\mathcal{S}_1) + tL(\mathcal{S}_2) = L^\star$, where $L(\mathcal{S})$ denotes the total length of strokes in $\mathcal{S}$.

A final additional criterion is that inbetweening should produce the smoothest possible transition between the two keyframes. Intuitively, it implies that more strokes from $\mathcal{S}'_1$ (resp. $\mathcal{S}'_2$) should be selected in the temporal vicinity of the first (resp. second) keyframe. In between, the proportion of selected strokes should progressively change from one keyframe to the next. This would be sufficient if the two key drawings had similar densities, but this is not the case in general (i.e., $\mathcal{D}_1 \neq \mathcal{D}_2$). At interpolation time $t \in [0,1]$, we must thus enforce a target density ratio:

$$\frac{\sum_{\hat{S}_1\in\hat{\mathcal{S}}_1} I[\mathcal{D}_{\hat{S}_1}]}{\sum_{\hat{S}_2\in\hat{\mathcal{S}}_2} I[\mathcal{D}_{\hat{S}_2}]} \approx \frac{(1-t)\, I[\mathcal{D}_1]}{t\, I[\mathcal{D}_2]}, \tag{4}$$

where $\hat{\mathcal{S}}_1$ (resp. $\hat{\mathcal{S}}_2$) is the set of synthesized strokes coming from the first (resp. second) key drawing. As before, we rely on an approximation based on total length for efficiency reasons. At time t, we thus instead target $\frac{L(\hat{\mathcal{S}}_1)}{L(\hat{\mathcal{S}}_2)} \approx \frac{(1-t)L(\mathcal{S}_1)}{tL(\mathcal{S}_2)} = L^\star_{1/2}$.

Algorithm 1 recaps our stroke synthesis procedure. It takes as input the two sets of warped candidate strokes $\mathcal{S}'_1$ and $\mathcal{S}'_2$, the target density $\mathcal{D}^\star$, the target total length $L^\star$ and the target length ratio $L^\star_{1/2}$ which implicitly depends on the interpolation time t. It outputs a set of synthesized strokes $\hat{\mathcal{S}}$, which is also given as input to the algorithm for the next intermediate frame. As of now, we input $\hat{\mathcal{S}} = \emptyset$, but we will see in Section 3.4 that different initial choices for $\mathcal{S}'_1$, $\mathcal{S}'_2$ and $\hat{\mathcal{S}}$ grant temporal control over the synthesis process. Note that the covered density $\mathcal{C}(S')$ for all warped candidate strokes S' must be recomputed at the beginning of every step (line 3), since the latest addition of a selected stroke always affects the target density $\mathcal{D}^\star$ (line 10). The two tests at lines 5 and 7 are fail-safes: they fail only when we run out of candidate strokes with sufficient density coverage, which terminates the algorithm since $\mathcal{S}' = \emptyset$. However, in practice, in all of our experiments, our approach never runs out of valid candidate strokes and thus never fails those tests.

The resulting synthesized strokes may be drawn as is (see Figure 6), or with attribute variations, typically thickness (see Figure 8). In the latter case, attribute values for each stroke point are simply fetched in screen space from interpolated attribute maps.

GPU implementation. To update the target density (line 10), we simply render the new selected stroke $\hat{S}$ with subtracting blending into $\mathcal{D}^\star$ using the stamping process described in Section 3.2. We then recompute the covered density for all candidate strokes in parallel on the GPU. Each point in the previously defined Vertex Buffer Object reads its corresponding density value in $\mathcal{D}^\star$ and writes it with additive blending into a 1D array indexed by stroke indices, hence paralleling the sum at line 3 over the points of all strokes in $\{\mathcal{S}'_1 \cup \mathcal{S}'_2\}$. To fetch this array more easily from the GPU memory, it is stored into a 2D texture with the same dimensions as the density map. The stroke index i is just remapped to the 2D coordinates $(i/h, i\%h)$, where h is the height of the density map. We

Algorithm 1: stroke distribution synthesis algorithm.

Input: $\mathcal{S}'_1, \mathcal{S}'_2, \mathcal{D}^\star, L^\star_{1/2}, L^\star, \hat{\mathcal{S}}$

Output: $\hat{\mathcal{S}}$

begin
  repeat
    ▷ Compute covered density for all candidate strokes
    $\mathcal{C}(S') = \sum_i \mathcal{D}^\star(\mathbf{x}'_i),\ \forall S' \in \{\mathcal{S}'_1 \cup \mathcal{S}'_2\}$
    ▷ Select a set of candidates w.r.t. the target length ratio
    $\mathcal{S}' \leftarrow L(\hat{\mathcal{S}}_1) \leq L^\star_{1/2} L(\hat{\mathcal{S}}_2)\ ?\ \mathcal{S}'_1 : \mathcal{S}'_2$
    if $\mathcal{S}' \neq \emptyset$ **then**
      ▷ Select the candidate stroke with maximum coverage
      $\hat{S} \leftarrow \arg\max_{S' \in \mathcal{S}'} \mathcal{C}(S')$
      if $\mathcal{C}(\hat{S}) > 0$ **then**
        $\hat{\mathcal{S}} \leftarrow \hat{\mathcal{S}} \cup \hat{S}$ ▷ Add it to the output set
        $\mathcal{S}' \leftarrow \mathcal{S}' \setminus \hat{S}$ ▷ Remove it from the candidate set
        $\mathcal{D}^\star \leftarrow \mathcal{D}^\star - \mathcal{D}_{\hat{S}}$ ▷ Update the target density
      else
        $\mathcal{S}' \leftarrow \emptyset$ ▷ Stop if no valid candidate is available
  until $L(\hat{\mathcal{S}}) \geq L^\star$ ***or*** $\mathcal{S}' = \emptyset$

only need to read back the first $\frac{|\mathcal{S}'_1 \cup \mathcal{S}'_2|}{h} + 1$ rows of this texture to CPU memory to update all candidate strokes covered density.

The cost of those repeated memory transfers can be too prohibitive for certain applications. We describe in Appendix A a more complex version of this algorithm that selects a batch of candidate strokes without updating the target and covered densities.

Synthesis results. Figure 6 shows the result of this stroke distribution synthesis algorithm at a single intermediate frame for a variety of sketchy looks, using long or short strokes, with large or small overlaps, aligned or not to the contour. Our method starts to show its limits when the drawing style includes gaps (last row). This is due to the linear interpolation of key density maps, which fills in gaps when they do not occur at the same locations. A solution might be to use a more sophisticated interpolation technique (e.g., mass transport) on densities for such specific drawing styles.

Our approach also handles changes of drawing style from one keyframe to the next, as shown in Figure 7. This is made possible by the target length ratio $L^\star_{1/2}$ which properly balances selected strokes coming from the key drawings. Since our method only synthesizes *distributions* of strokes, the style of individual strokes is not itself interpolated. Whether this should be done is a matter of artistic choice. We leave that specific option to future work, as it raises additional technical challenges, especially in an interactive setting.

Stroke attributes such as thickness may also be varied between key drawings, either globally or locally. Our approach produces adequate interpolation results in both cases, as shown in Figure 8.

The supplemental videos show the full interpolated sequences for those results. Although the style is well preserved throughout the sequence, temporal artifacts are substantial, often excessive to properly depict the motion of the underlying contour. In the next section we analyse the reasons of those temporal artifacts and extend the synthesis algorithm with strategies to control them.

3.4. Temporal strategies

The stroke distribution synthesis algorithm described in the previous section works independently at each frame. Therefore, some candidate strokes that are selected at the current frame may not have been selected at the previous frame and/or may disappear at the next frame, yielding temporal artifacts. At the same time, selecting the same candidate stroke for all inbetweened frames might not be desired either, since it will attract attention towards the motion of the stroke instead of the motion of the depicted contour. Indeed, as previously mentioned, we consider strokes as part of a vectorial texture that represents an underlying contour, and the goal of the synthesis process is to generate new, but similar, vectorial textures while reducing temporal artifacts as much as possible.

One solution would be to apply the method of Noris et al. [NSC*11] to reduce the amount of temporal artifacts, which they call "noise". Unfortunately, the method has two main drawbacks: (1) it is far too computationally demanding to yield interactive feedback, and (2) reducing noise has the drawback of attracting attention toward individual stroke motion as discussed in Section 5.2. We propose an alternative approach based on temporal strategies whose objective is to find a compromise between different sources of artifacts: flickering, popping and residual motion.

Flickering. When a candidate stroke is not selected at the current frame but is selected in previous *and* next frames, this leads to a flickering artifact. A solution to this problem is to forbid a disappearing stroke S' to reappear in the next Δ_d frames. This is simply done by removing S' from its set $\mathcal{S}'_1$ or $\mathcal{S}'_2$ prior to stroke distribution synthesis.

Popping. When a candidate stroke appears or disappears at a given frame, it will inevitably create a popping artifact. As previously mentioned, such artifacts cannot be avoided, but they can be reduced; in particular, they can be spaced apart in time as much as possible. Our solution is thus to enforce the selection of an appearing stroke S' in the next Δ_a subsequent frames, so that two appearing/disappearing popping artifacts do not occur too close in time. This is done by adding S' to the initial set of selected strokes $\hat{\mathcal{S}}$ prior to stroke distribution synthesis and updating the target density $\mathcal{D}^\star$ accordingly. To avoid synchronized stroke appearance every Δ_a, we assign an initial lifetime randomly chosen in the $[0, \Delta_a)$ range to the strokes of the first synthesized frame.

Residual motion. Selecting the same candidate stroke S' for several frames has an inherent drawback though: the motion of S' will be smooth throughout that time interval, attracting attention to its own motion instead of the motion of the underlying contour, which we call a residual motion. When this happens, the stroke appears to "slide", which yields a subjectively less natural look and feel. Residual motion increases with larger Δ_a, but smaller Δ_a yield more popping artifacts. A work-around consists in slightly perturbing the

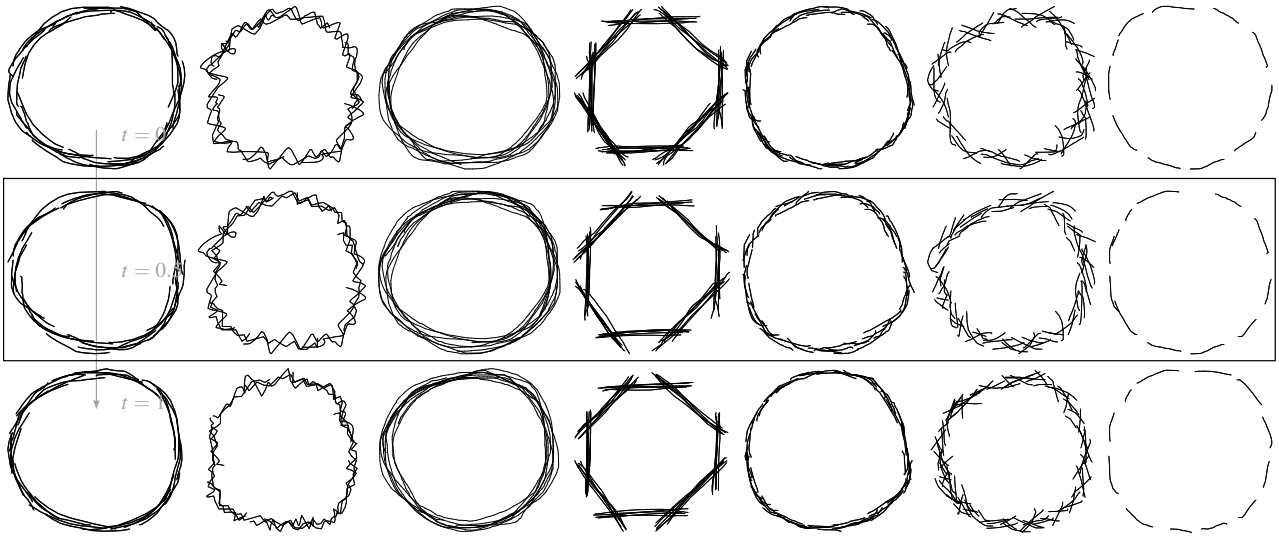


Figure 6: ***Style variations.*** *Our interpolation results (middle row, framed) reproduce the styles of keyframes (top and bottom rows). We use the same key shapes (i.e., with identity transformations) to focus on style reproduction.*

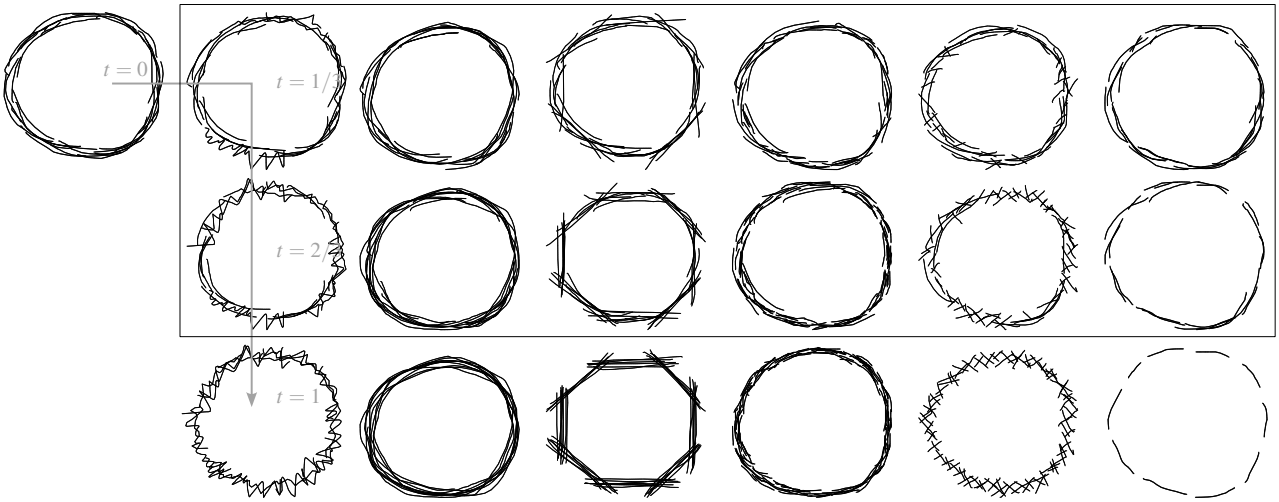


Figure 7: ***Interpolation across different styles.*** *Starting from a same input style for the first keyframe (left-most drawing), we interpolate with keyframes drawn in different styles (bottom row). Our results (top and middle rows, framed) interpolate stroke distributions in a way that transitions between styles.*

strokes selected by the temporal strategies in $\hat{\mathcal{S}}$ prior to stroke distribution synthesis using linear perturbation (see Section 3.1), with a user-controlled parameter β.

A temporal strategy in our approach then consists in a triplet $\{\Delta_d, \Delta_a, \beta\}$, where the $\{0,0,0\}$ strategy selects candidate strokes at each frame independently without perturbation (i.e., no temporal control). The next section introduces error metrics to help decide which strategy is most adapted.

4. Error metrics

How should one choose among different temporal strategies? There is no ideal solution since the (potentially subtle) look-and-feel produced by different strategies might appeal differently to different artists. Nevertheless, we present in this section error metrics tailored to our approach that help make a relevant choice.

Each metric is computed locally in time around the current frame. We define a pair of *temporal incoherence metrics* related to flickering and popping artifacts in Section 4.1, a pair of *motion metrics* to capture "sliding" artifacts and to ensure that the underlying contour motion is depicted properly in Section 4.2, and finally a *total density metric* that tracks integrated density in Section 4.3. We evaluate our temporal strategies using these error metrics in Section 4.4 on the example of Figure 2. Since some of these metrics require stroke-level correspondences between frames, we first describe a simple stroke matching technique.

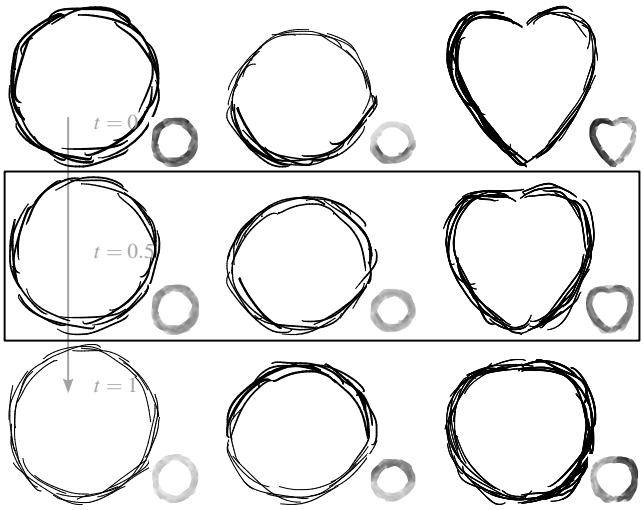


Figure 8: ***Thickness variations.*** *Key drawings (top and bottom rows) with thickness variations are properly interpolated with our approach (middle row, framed). From left to right, we show variations from thick to thin strokes; from thick at the bottom to thick at the top; and from thick at left to thick at right with two different key shapes. Key and interpolated thickness maps are shown as insets.*

Stroke matching. The matching of strokes between consecutive frames is simplified by our stroke distribution synthesis algorithm since for each synthesized stroke we can identify the key stroke it comes from. We thus make here the simplifying assumption that the viewer will only be able to visually track strokes originating from the same key stroke. Let $\hat{S}_{t-\Delta t}$ and $\hat{S}_t$ be two synthesized strokes from two consecutive frames, which are generated from the same key stroke S. The *spatial distance* between this pair of strokes is defined by:

$$d(\hat{S}_{t-\Delta t}, \hat{S}_t) = \frac{1}{|S|} \sum_{i=1}^{|S|} ||\mathbf{x}_i(t-\Delta t) - \mathbf{x}_i(t)|| \tag{5}$$

where $\mathbf{x}_i(t)$ (resp. $\mathbf{x}_i(t-\Delta t)$) is a point from $\hat{S}_t$ (resp. $\hat{S}_{t-\Delta t}$), and $|S| = |\hat{S}_t| = |\hat{S}_{t-\Delta t}|$ by construction. If only one $\{\hat{S}_{t-\Delta t}, \hat{S}_t\}$ pair exists, then stroke matching is trivial. When several pairs are found, we apply the following simple, brute-force matching algorithm: among all possible pairs, we find the one with minimum spatial distance (Equation 5) to establish a stroke matching; then we remove this pair and iterate until no further pair can be found. Note that some synthesized strokes might be left unmatched, which is taken into account in temporal incoherence metrics.

4.1. Temporal incoherence metrics

In our approach, a temporal artifact is always related to the appearance or disappearance of a synthesized stroke. We distinguish two types of temporal artifacts and organize strokes in two corresponding sets: the flickering set, holding synthesized strokes that disappear for a few frames then reappear; and the popping set, which gathers all other appearing or disappearing synthesized strokes. We provide formal definitions for these two sets below and derive simple metrics from them.

Flickering strokes. A synthesized stroke is considered to flicker if it disappears and reappears within a sufficiently short distance in both space and time. A flickering event is assumed to occur once a stroke reappears; hence we need to look in the past to determine when it last disappeared. In practice we consider strokes that have disappeared a maximum of three frames in the past, hence we compare all pairs of synthesized strokes $\hat{S}_{t-k\Delta t}$ and $\hat{S}_t$ coming from the same key stroke, with $k \in \{2,3\}$. $\hat{S}_t$ is then added to the flickering set $\hat{\mathcal{S}}_f(t)$ if $d(\hat{S}_{t-k\Delta t}, \hat{S}_t)$ is below a fraction of it length (in practice we use $d(\hat{S}_{t-k\Delta t}, \hat{S}_t) < L(\hat{S}_t)/5$ pixels). In other words, Equation 5 is used to decide whether synthesized strokes in *non-contiguous* frames are close enough to be considered flickering. The flickering metric is then simply defined by:

$$E_f(t) = \frac{1}{L(\hat{\mathcal{S}}_t)} \sum_{\hat{S} \in \hat{\mathcal{S}}_f(t)} |\hat{S}|, \tag{6}$$

where $L(\hat{\mathcal{S}}_t)$ is the total length of synthesized strokes at time t.

Popping strokes. A stroke $\hat{S}_t$ is added to the popping set $\mathcal{S}_p(t)$ if it appears or disappears at time t, but is not part of the flickering set *at any other frame*. The popping metric is then defined similarly to the flickering metric:

$$E_p(t) = \frac{1}{L(\hat{\mathcal{S}}_t)} \sum_{\hat{S} \in \hat{\mathcal{S}}_p(t)} |\hat{S}|. \tag{7}$$

Note that increasing the allowable time gaps for the flickering set (values taken by k) will increase its size and decrease the size of the popping set.

4.2. Motion metrics

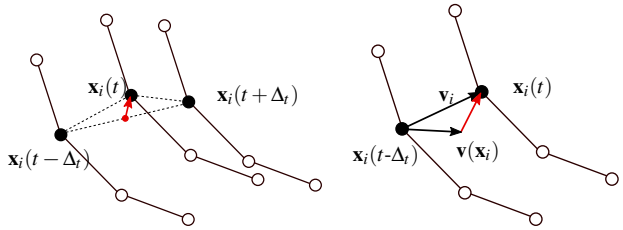
The synthesized animation should not attract attention toward the motion of strokes themselves, but instead convey the motion of the underlying contour. We design two metrics to verify to which extent this is the case.

Smoothness of stroke motion. Strokes that neither appear nor disappear and last for several frames may produce individual motions that attract attention away from the motion of the depicted underlying contour: they may appear to "slide" autonomously. We thus measure to which extent these strokes exhibit a smooth trajectory, which makes the hypothesis that the smoothest motion yields more significant sliding.

Let us first identify the set of strokes that exist on three consecutive frames (i.e., at times $t-\Delta t$, t and $t+\Delta t$) by $\hat{\mathcal{S}}_s(t)$. Formally, the stroke motion metric measures the discrete curvature of the trajectory of stroke points at time t. This is done by computing the deviation of a stroke from a linear trajectory (see Figure 9(a)):

$$E_s(\hat{S}) = \sum_{i=1}^{|\hat{S}|} \left(1 - \frac{1}{\Delta_i(t)} \left\| \mathbf{x}_i(t) - \frac{\mathbf{x}_i(t+\Delta t) + \mathbf{x}_i(t-\Delta t)}{2} \right\| \right), \tag{8}$$

where $\Delta_i(t) = \sqrt{\frac{||\mathbf{x}_i(t)-\mathbf{x}_i(t-\Delta t)||^2 + ||\mathbf{x}_i(t)-\mathbf{x}_i(t+\Delta t)||^2}{2}}$ is used to remap the discrete curvature to the $[0,1]$ range.



(a) Smoothness of stroke motion (b) Motion incoherence

Figure 9: ***Illustration of our motion error metrics.*** *(a) The smoothness of stroke motion is computed per vertex $\mathbf{x}_i$ as the magnitude of the deviation (red vector) from a straight motion on 3 consecutive frames. (b) The motion incoherence is computed as the magnitude of the deviation (red vector) between the vertex velocity vector $\mathbf{v}_i$ and the expected motion vector $\mathbf{v}(\mathbf{x}_i)$.*

The stroke motion metric is then defined as:

$$E_s(t) = \frac{1}{L(\hat{\mathcal{S}}_t)} \sum_{\hat{S} \in \hat{\mathcal{S}}_s(t)} E_s(\hat{S}). \tag{9}$$

Motion incoherence. Lastly, we need to make sure that the motion conveyed by synthesized strokes is coherent with the motion of the underlying contour. To this end, we introduce a motion incoherence metric that relies on matched strokes and compares their motion vectors to the underlying motion computed with ARAP interpolation. Formally, all strokes that have a matching stroke in the previous frame are gathered in a set $\hat{\mathcal{S}}_m$, and we measure the deviation of those stroke motion vectors from the reference motion flow, as illustrated in Figure 9(b). However simply adding up the norm of these deviations would overestimate the error since it would ignore the local cancellation of those vectors. To take this effect into account, we average the deviations in a local neighborhood (with radius $\ell = 32$ pixels by default):

$$E_m(\hat{S}) = \sum_{i=1}^{|\hat{S}|} \left\| \mathbf{v}(\mathbf{x}_i) - \frac{1}{|\mathcal{N}_\ell(\mathbf{x}_i)|} \sum_{j \in \mathcal{N}_\ell(\mathbf{x}_i)} \mathbf{v}_j \right\|, \tag{10}$$

where $\mathbf{v}(\mathbf{x}_i)$ denotes the reference motion flow, $\mathcal{N}_\ell(\mathbf{x}_i)$ is the set of point indices such that $||\mathbf{x}_i - \mathbf{x}_j|| < \ell$ and $\mathbf{v}_j = \mathbf{x}_j(t) - \mathbf{x}_j(t-\Delta t)$ is the motion vector of the jth stroke point.

The motion incoherence metric is eventually defined by:

$$E_m(t) = \frac{1}{L(\hat{\mathcal{S}}_t)} \sum_{\hat{S} \in \hat{\mathcal{S}}_m(t)} E_m(\hat{S}). \tag{11}$$

4.3. Total density metric

A linear interpolation between two drawings of different densities $\mathcal{D}_1$ and $\mathcal{D}_2$ should result in interpolated drawings whose density is a linear interpolation of $\mathcal{D}_1$ and $\mathcal{D}_2$ (i.e., $\mathcal{D}^\star = (1-t)\mathcal{D}_1 + t\mathcal{D}_2$). However, we cannot directly compare the density of synthesized strokes with $\mathcal{D}^\star$, since a mere offset of strokes would result in an arbitrarily large difference. Instead, we rely on a metric that compares integrated densities $I[\mathcal{D}]$:

$$E_d(t) = \frac{|I[\mathcal{D}^\star] - \sum_{\hat{S} \in \hat{\mathcal{S}}_t} I[\mathcal{D}_{\hat{S}}]|}{I[\mathcal{D}^\star]}. \tag{12}$$

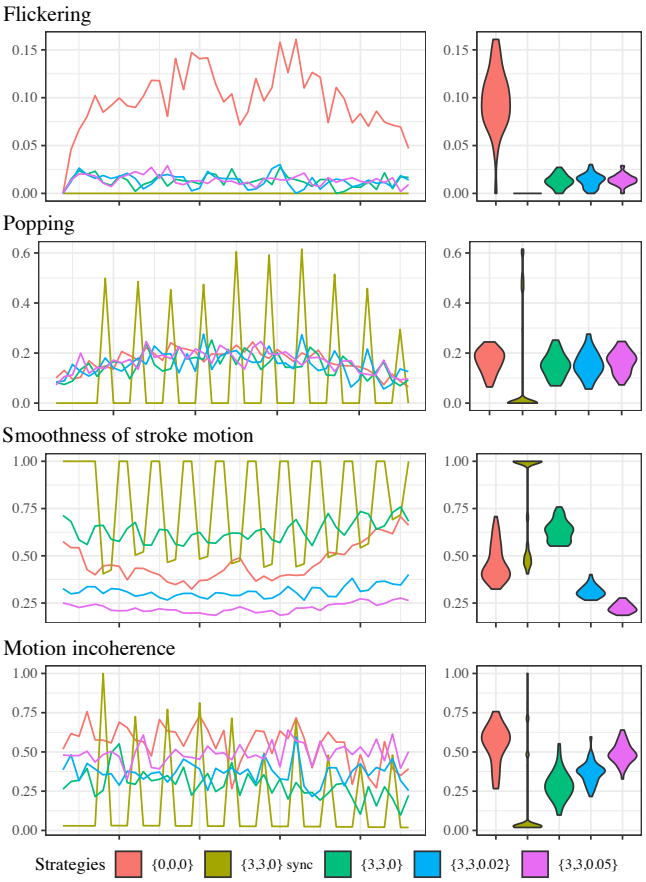


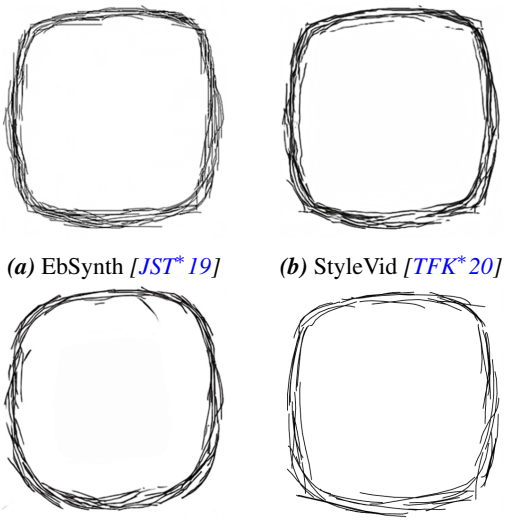
Figure 10: ***Evaluation of temporal strategies.*** *We measure how five strategies perform on the error metrics introduced in Section 4. Curves in the left column show metric results as a function of time. Violin plots in the right column offer temporal statistical metric summaries per strategy. We do not show the density metric as it is close to zero for all strategies.*

Note that $E_d \approx 0$ will occur if Equation 3 is met. Since we do not directly use that equation in our algorithm but an approximation based on total length, E_d is a good metric to estimate the accuracy of our approximation.

4.4. Strategies evaluation

Now that we are equipped with several error metrics, we are ready to evaluate the temporal strategies $\{\Delta_a, \Delta_d, \beta\}$ of Section 3.4. Besides the $\{0,0,0\}$ strategy, we consider a variety of alternatives in supplemental material, all computed on the example of Figure 2. The motion incoherence metric is normalized by the maximum error found across all frames and strategies. Even though all metrics are normalized in the $[0,1]$ range, their relative visual impact cannot easily be predicted: each metric should thus be considered independently when comparing temporal strategies.

We compare a subset of strategies in details in Figure 10. The flickering metric (first row) clearly shows that the $\{0,0,0\}$ strategy is unacceptable compared to the other ones. If we instead use



(a) EbSynth *[JST*19]* *(b)* StyleVid *[TFK*20]*

(c) STAPL *[FKL*21]* *(d)* *Our approach*

Figure 11: ***Comparison with example-based raster synthesis.*** *Neither patch-based texture synthesis methods (a) nor learning-based methods (b-c) preserve stroke consistency, whereas our approach (d) does so by construction as it synthesizes vector strokes.*

$\Delta_a = \Delta_d = 3$ frames, flickering is strongly reduced. The importance of desynchronizing those parameters in the first synthesized frame is demonstrated by the popping metric (second row): the synchronized strategy $\{3,3,0\}$ sync (in olive) produces groups of strokes that appear and disappear together, as seen in the supplemental video. Moreover, they tend to exhibit smooth motion inbetween those frames (third row), which results in sliding artifacts. The effect of the perturbation parameter β is to break the smoothness of stroke motion, at the expense of slightly increasing motion incoherence (last row). As discussed in the supplemental document, we have found that choosing β in the $[0.02, 0.05]$ range offers the best trade-off in practice. We do not show results for the density metric as it is close to zero for all strategies ($\forall t,\ E_d(t) < 0.0464$), which confirms the accuracy of our total length approximation. However, we use it to compare to the method of Noris et al. [NSC*11] in Section 5.2.

5. Results

In this section, we expose the benefits of our approach over existing approaches (Sections 5.1 and 5.2), before demonstrating its capabilities on complex animations (Section 5.3) and discussing its performance (Section 5.4).

5.1. Comparison with raster-based approaches

Figure 11 compares our method with three state-of-the-art example-based stylization approaches: *EbSynth* [JST*19], *StyleVid* [TFK*20] and *STAPL* [FKL*21]. Instead of synthesizing strokes, they generate raster images: *EbSynth* synthesizes bitmap texture patches that are extracted from key drawings, whereas the latter two methods use an image translation network trained with a few pairs of source / stylized images. With *EbSynth*, individual strokes are implicitly broken during patch extraction, and cannot be re-assembled during texture synthesis. In addition, it tends to create ghosting artifacts. *StyleVid* and *STAPL* slightly better preserve stroke consistency, but the results look more like shape morphing than hand-drawn animations, as is clearly visible in the accompanying videos. We conclude that current raster-based approaches are not adapted to the interpolation of rough line drawings.

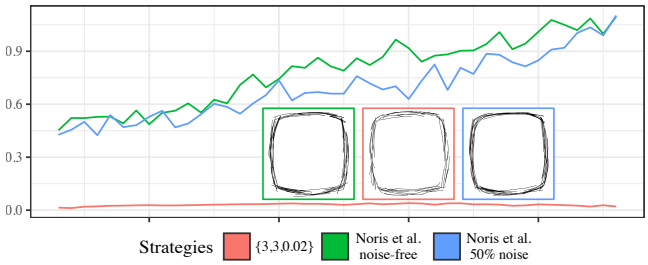


Figure 12: ***Comparison with noise reduction [NSC*11].*** *The total density metric shows that stroke density is not preserved by the noise reduction technique, which is confirmed by visual inspection at an intermediate frame (insets). Creating the noise-free version from the sequence synthesized with our $\{0,0,0\}$ strategy takes about 15ms per frame, but an additional 25ms per frame is required to generate any intermediate noise levels (here 50%).*

5.2. Comparison with noise reduction

The method of Noris et al. [NSC*11] is originally intended to control noise given a hand-drawn rough line animation as input. For the purpose of comparison, we instead apply it on a set of interpolated drawings obtained using our approach with the $\{0,0,0\}$ strategy, with the intent to compare it to our temporal strategies. In their approach a noise-free interpolation is first computed, then matched to the noisy input using ARAP registration and stroke-to-stroke Hausdorff distance such that a result with intermediate noise reduction may be produced.

Our implementation relies only on the Hausdorff distance for matching (no ARAP registration), which makes the method way more efficient. Nevertheless, it is still approximately 2 times slower than our solution. More importantly, as shown in Figure 12, stroke density is not well preserved by either of their results as shown by our total density metric. This was expected since in their method, strokes are initially duplicated to generate the noise-free version of the animation. Our solution matches density closely, which validates our choice of approximation based on total stroke length.

Our other metrics are tailored to our stroke distribution synthesis algorithm, and are thus not easily adaptable to the method of Noris et al. One exception is the smoothness of stroke motion, which may be trivially computed on the noise-free result. As shown in supplemental material, this reveals the presence of many sliding artifacts, which is to be expected since the noise-free version exhibits no popping nor flickering by construction. A subjective comparison is provided through the user study of Section 6.

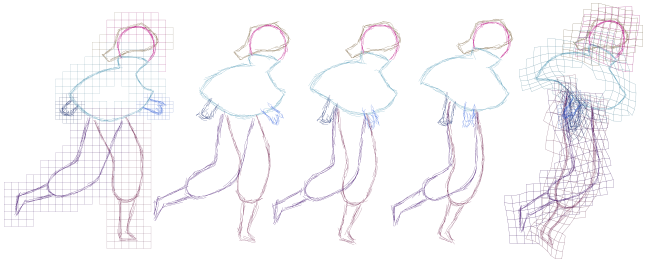


Figure 13: ***Multi-layer animations.*** *For complex animations, we transform and interpolate each drawing part separately in a different layer. In this walk cycle, the left-most and right-most drawings show the transformation lattices assigned to each part, depicted with a specific color. We show our interpolation results with the same colors. Here we use the* $\{3,3,0.05\}$ *strategy.*

Figure 14: ***Complex animation with thickness variations.*** *As in Figure 13, we use several layers to animate this leopard (keyframes in gray, interpolation in black). In addition, we decrease stroke thickness based on motion speed. Our solution properly interpolates such stroke attributes. Here we use the* $\{3,3,0.02\}$ *strategy.*

5.3. Complex results

As shown in Figure 1, our method handles animations composed of multiple key-frames. For simplicity and efficiency, each pair of consecutive keyframes is processed independently, implying that the selection strategy is reset at each keyframe. For such complex animations, we use multiple layers, one for each part of the animated shape. As shown in Figure 13, each layer is controlled by its own ARAP lattice and interpolated independently, even though hierarchical constraints between layers [EBB23] could be added. We use a lattice size of $\ell = 32$ pixels by default, but it may be adjusted according to the size of the layer. Stroke thickness may be varied throughout keyframes, as seen in Figure 14 where we decrease thickness with motion speed.

5.4. Performances

Our prototype implementation achieves interactive performance even on modest configurations (e.g., laptops) thanks to both GPU acceleration and the use of stroke batches for synthesis (see Appendix). A timing breakdown is provided in Table 2 for the example of Figure 2: its key drawings have 70 and 48 key strokes respectively, and 1048 candidate strokes are generated in pre-process in about 1.5s. We indicate the mean computation time per frame, averaged over 46 intermediate frames, which are then cached for playback. The GPU-batch implementation is required for interactive feedback in this case, with most of the time spent in the computation of stroke densities through splatting. In comparison, the CPU implementation is much longer, with most of the time spent in the update of the density map. The last column of Table 2 provides timings for the GPU-batch algorithm on the same animation but with key drawings having 215 and 218 key strokes respectively, leading to 4330 candidates strokes. Timings scale reasonably well with the number of candidate strokes, most of the overhead being due to under-optimized CPU parts of the code.

	CPU	**GPU**	**GPU-batch**	
# candidate strokes	1048	1048	1048	4330
$\mathcal{S}_k(t)$: warping & resampling (CPU)	5.4	5.4	5.4	20.8
$\mathcal{D}^\star(t)$: target density interpolation	25	2.1	2.1	9.3
$\mathcal{D}(S)$: stroke densities computation	4.3	10.4	2.2	4.1
$\hat{S}$: strokes sorting (CPU)	5.7	5.7	1.7	1.6
$\mathcal{D}^\star$: density map update	997.4	6	2	2.9
other CPU computation	2.2	3.1	1.2	30.9
Total (per frame)	1040	32.7	14.6	69.6

Table 2: ***Performance breakdown.*** *Timings per frame (in milliseconds) for each implementation, detailed for the most computationally expensive steps of our method. Candidate strokes are generated beforehand during a separate CPU pre-process in about 1.5s (resp. 2.5s) for 1048 (resp. 4330) strokes. Performance averaged over the 46 frames of the example in Figure 2, measured on a Intel I5 7300HQ CPU and a NVidia GeForce GTX1050Ti* ***Mobile*** *GPU.*

6. User Study

To validate the effectiveness of our selection strategies and compare it with noise reduction [NSC*11], we conducted a user study involving 38 undergraduate art students that followed at least one course on animation. The code to run the study was developed using *PsychoPy2* [PGS*19], and is provided with the stimuli in the supplemental materials.

6.1. Experiment

The user study is a force-choice experiment asking the participants to choose which one of two animations looks more hand-drawn (A/B testing).

Stimuli. The stimuli are two-second animations generated from five sets of keyframes with similar drawing styles but various degrees of complexity (e.g., abstract shape vs. character, planar deformation vs. complex motion). We consider four methods to generate the stimuli: our algorithm with the $\{0,0,0\}$ strategy as the baseline, the $\{3,3,0.02\}$ and $\{3,3,0.05\}$ strategies (cf. Section 4.4), the noise-free (N100) and 50% noise level (N50) versions of Noris et al. [NSC*11] (cf. Section 5.2). We consider two values of β to evaluate the compromise between motion incoherence and smoothness of stroke motion.

Protocol. To anchor the hand-drawn look-and-feel, participants are first presented with five rough animations that are not part of the stimuli and were manually drawn by professional artists. Then, for

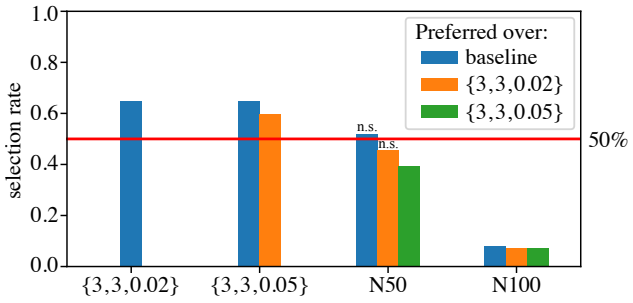


Figure 15: ***Histogram of selection rates of the studied comparisons.*** *Each bar corresponds to the selection rate of a given method when compared to the baseline, ours with the* $\{3,3,0.02\}$ *or the* $\{3,3,0.05\}$ *temporal strategy. 50% chance level is shown with a red line. Selection rates non significantly different than chance level are indicated by 'n.s.'.*

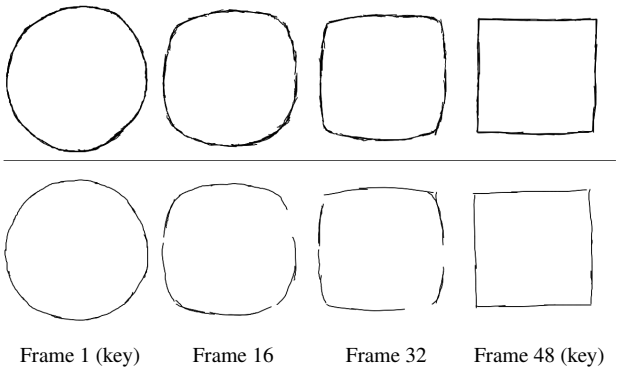


Figure 16: ***Stroke overlap.*** *Our method can handle rather "clean" key drawings (top), but if strokes insufficiently overlap with each others, gaps may appear in the interpolated frames (bottom).*

each trial, two animations generated by a different method starting from the same input keyframes are played consecutively, and may be played back as many times as needed. The participant is then asked to choose which animation looks more hand-drawn in her opinion. The first five trials are training examples to familiarize the participants with the interface and animations. Those are generated with the baseline $\{0,0,0\}$ and simple cross-fading for the five sets of keyframes. After this training phase, we compare $\{3,3,0.02\}$, $\{3,3,0.05\}$, N50 and N100 against the baseline $\{0,0,0\}$, as well as our two strategies against N50 and N100, and against each other, i.e., nine comparisons for the five sets of keyframes. Each comparison was presented twice in a different order, resulting in 90 randomized comparisons per participant.

6.2. Results

For all participants, we compute the selection rate of each of the nine method comparisons by averaging the participant choices over the two repetitions of the five animations. We present the average selection rates obtained for each comparison in Figure 15. Details of the statistical analysis and additional results are presented in the supplemental materials.

As a reminder, if participants were not able to choose which method looked more hand-drawn in a method comparison, the selection rate should be close to the 50% chance level. For each comparison, we thus test if the selection rate was significantly different than the chance level using either a one sample *Student's t-test* or a one sample *Wilcoxon signed rank test* if we cannot assume a normal distribution of the participants' selection rates [Con99]. Similarly, we study which method performed better against the baseline and against each other.

Effectiveness of our temporal strategies. Participants found that animations generated with our two strategies looked more hand-drawn than the $\{0,0,0\}$ baseline. This validates our choice of reducing flickering with $\Delta_a = \Delta_d = 3$. Moreover, they found that animations generated with $\beta = 0.05$ looked more hand-drawn than with $\beta = 0.02$, which helps select the best trade-off between breaking motion smoothness and increasing motion incoherence using the perturbation parameter β.

Comparison with noise reduction. As expected, the selection rates of the noise-free method (N100) against the baseline and our strategies were all found lower than the chance level by a large margin, and N100 performed significantly worse than all other methods against the baseline. Our two strategies performed significantly better against the baseline than the 50% noise version (N50) (65.0% and 64.7% vs. 52.1%), which indicates that they produce animations that look more hand-drawn. Moreover, while participants could not choose which animation looked more hand-drawn when comparing N50 directly to the baseline and to our $\{3,3,0.02\}$ strategy, the selection rate of N50 against our $\{3,3,0.05\}$ strategy was significantly lower than chance level. This further indicates that $\beta = 0.05$ gives the best trade-off between motion smoothness and incoherence in our tested strategies. The impact of density variations of N50 (see Figure 12) on these results remains unclear and would require further user studies and analysis.

7. Discussion

We have presented an efficient stroke distribution synthesis technique that produces compelling interpolations of rough line drawings and reduces temporal artifacts through several user-controllable strategies. Comparisons with previous work have demonstrated that a dedicated solution is required to properly interpolate rough drawings, and we have shown how our approach handles different types of styles, attributes, and motions, with interactive feedback thanks to our GPU-accelerated implementation. The error metrics we have introduced not only help choose a strategy, but also permit the comparison with alternative approaches. We performed a user study that confirms the superiority of our approach from a subjective standpoint.

Limitations. The most straightforward limitations of our approach lie in the adequacy of the input key drawings. This appears in the last column of Figure 6. In general, our approach expects a sketchy drawing style that does not draw attention to individual strokes, which is different from stroke pattern synthesis techniques, as already mentioned in Section 2. Figure 16 further shows how our method behaves as the input key drawings get "cleaner", i.e., with

Figure 17: ***Failure case.*** *When keyframes are not well registered, as shown at right (blue/red strokes from key drawing 1/2), warped candidate strokes will inevitably produce ghosting artifacts.*

less overdraw and strokes closer to the underlying contour. With too few overlaps between strokes, our method may produce gaps. Another issue arises when the registration between key drawings is not accurate enough, as shown in Figure 17. Our approach then logically yields ghosting artifacts.

As noticeable in Figures 1, 13 and 14, our method does not take visibility into account. This is not much of an issue in the early stage of rough animations: in many of the example animations we analyzed, hidden parts are often explicitly drawn. However, when refining the animation, visibility handling will likely become increasingly important. The major issue will then be to solve the matching problem between partially occluded sets of strokes, which to the best of our knowledge is still an open problem. Once solved, our approach could easily be adapted by either providing a different target distribution, by making use of masks that partially hide synthesized strokes, or by a combination of both.

Our error metrics have proved useful to select the best parameters for temporal strategies. However, aside from total density, they are tailored to our stroke distribution synthesis algorithm. It would be interesting to extend them to other stroke synthesis techniques. Ultimately, additional user experiments will be required to evaluate how well they correlate with human perception.

Future work. We have explored a number of temporal strategies. However, once a strategy has been chosen, we rely on it for the whole animation. It would be interesting to explore how strategies could be adjusted to adapt to the motion speed and complexity between two keyframes. Even though we only focused on the interpolation problem in this work, we also relied on rather simple correspondences between key drawings. One might wonder how it might handle more complex cases where drawings become partially hidden by others. Our approach could also be adapted to other stroke-based styles, such as painterly rendering, or maybe even watercolor. An intriguing direction of future work is the extension of our approach to 3D line drawings produced in VR environments.

Acknowledgments

We would like to thank Prof. Gao Fei, Prof. Xu Yuzhong, and student volunteers at the Art School of Zhejiang University of Technology, for helping us with the user experiment. We are grateful to Daniel Sýkora for running the comparisons with raster-based approaches. This work is supported by the ANR MoStyle project (ANR-20-CE33-0002), the National Natural Science Foundation of China (62172367), and the Natural Science Foundation of Zhejiang Province (LGF22F020022).

References

[ACOL00] ALEXA M., COHEN-OR D., LEVIN D.: As-rigid-as-possible shape interpolation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), pp. 157–164. 3, 4

[ADN*17] ARORA R., DAROLIA I., NAMBOODIRI V. P., SINGH K., BOUSSEAU A.: Sketchsoup: Exploratory ideation using design sketches. *Computer Graphics Forum 36*, 8 (2017), 302–312. doi:10.1111/cgf.13081. 3

[BBA09] BAXTER W., BARLA P., ANJYO K.-I.: Compatible embedding for 2d shape animation. *IEEE Transactions on Visualization and Computer Graphics 15*, 5 (2009), 867–879. doi:10.1109/TVCG.2009.38. 3

[BBM*16] BEN-ZVI N., BENTO J., MAHLER M., HODGINS J., SHAMIR A.: Line-drawing video stylization. *Computer Graphics Forum 35*, 6 (2016), 18—-32. doi:10.1111/cgf.12729. 2, 3

[BBT*06] BARLA P., BRESLAV S., THOLLOT J., SILLION F., MARKOSIAN L.: Stroke pattern analysis and synthesis. *Computer Graphics Forum 25*, 3 (2006). doi:10.1111/j.1467-8659.2006.00986.x. 3

[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 Trans. Graph. 32*, 4 (2013), 119:1–119:12. doi:10.1145/2461912.2461929. 3

[BSM*13] BERGER I., SHAMIR A., MAHLER M., CARTER E., HODGINS J.: Style and abstraction in portrait sketching. *ACM Trans. Graph. 32*, 4 (2013). doi:10.1145/2461912.2461964. 3

[BW75] BURTNYK N., WEIN M.: Computer animation of free form images. In *Proceedings of the 2nd Annual Conference on Computer Graphics and Interactive Techniques* (1975), ACM, p. 78–80. doi:10.1145/563732.563743. 2

[CMV17] CARVALHO L., MARROQUIM R., VITAL BRAZIL E.: Dilight: Digital light table – inbetweening for 2d animations using guidelines. *Computers & Graphics 65* (2017), 31–44. doi:https://doi.org/10.1016/j.cag.2017.04.001. 2

[Con99] CONOVER W. J.: *Practical nonparametric statistics*, vol. 350. john wiley & sons, 1999. 13

[CZ22] CHEN S., ZWICKER M.: Improving the perceptual quality of 2d animation interpolation. In *Proceedings of the European Conference on Computer Vision* (2022). 3

[DLKS18] DVOROŽNÁK M., LI W., KIM V. G., SÝKORA D.: Toonsynth: example-based synthesis of hand-colored cartoon animations. *ACM Trans. Graph. 37*, 4 (2018), 167. doi:10.1145/3197517.3201326. 3

[DRvdP15] DALSTEIN B., RONFARD R., VAN DE PANNE M.: Vector graphics animation with time-varying topology. *ACM Trans. Graph. 34*, 4 (2015). doi:10.1145/2766913. 2

[EBB23] EVEN M., BÉNARD P., BARLA P.: Non-linear rough 2d animation using transient embeddings. *Computer Graphics Forum* (2023). doi:10.1111/cgf.14771. 1, 2, 3, 12

[FJL*16] FIŠER J., JAMRIŠKA O., LUKÁČ M., SHECHTMAN E., ASENTE P., LU J., SÝKORA D.: StyLit: Illumination-guided example-based stylization of 3d renderings. *ACM Trans. Graph. 35*, 4 (2016). doi:10.1145/2897824.2925948. 3

[FKL*21] FUTSCHIK D., KUČERA M., LUKÁČ M., WANG Z., SHECHTMAN E., SÝKORA D.: Stalp: Style transfer with auxiliary limited pairing. *Computer Graphics Forum 40*, 2 (2021), 563–573. doi:10.1111/cgf.142655. 2, 3, 11

[FLJ*14] FIŠER J., LUKÁČ M., JAMRIŠKA O., ČADÍK M., GINGOLD Y., ASENTE P., SÝKORA D.: Color me noisy: Example-based rendering of hand-colored animations with temporal noise control. *Computers Graphics Forum 33*, 4 (2014), 1–10. doi:10.1111/cgf.12407. 3

[FTP03] FREEMAN W. T., TENENBAUM J. B., PASZTOR E. C.: Learning style translation for the lines of a drawing. *ACM Trans. Graph. 22*, 1 (2003), 33–46. doi:10.1145/588272.588277. 3

[GAM*21] GIESEKE L., ASENTE P., MECH R., BENES B., FUCHS M.: A survey of control mechanisms for creative pattern generation. *Computer Graphics Forum 40*, 2 (2021), 585–609. doi:10.1111/cgf.142658. 3

[HJO*01] HERTZMANN A., JACOBS C. E., OLIVER N., CURLESS B., SALESIN D. H.: Image analogies. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (2001), SIGGRAPH '01, ACM, p. 327–340. doi:10.1145/383259.383295. 3

[HLT*09] HURTUT T., LANDES P.-E., THOLLOT J., GOUSSEAU Y., DROUILHET R., COEURJOLLY J.-F.: Appearance-guided synthesis of element arrangements by example. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering (NPAR)* (2009), ACM, pp. 51–60. doi:10.1145/1572614.1572623. 3

[HOCS02] HERTZMANN A., OLIVER N., CURLESS B., SEITZ S. M.: Curve analogies. In *Proceedings of the 13th Eurographics Workshop on Rendering* (2002), Eurographics Association, p. 233–246. 3

[HZH*22] HUANG Z., ZHANG T., HENG W., SHI B., ZHOU S.: Real-time intermediate flow estimation for video frame interpolation. In *Computer Vision – ECCV 2022: 17th European Conference* (2022), Springer-Verlag, p. 624–642. doi:10.1007/978-3-031-19781-9_36. 3

[IMIM08] IJIRI T., MECH R., IGARASHI T., MILLER G.: An example-based procedural system for element arrangement. *Computer Graphics Forum 27*, 2 (2008), 429–436. doi:10.1111/j.1467-8659.2008.01140.x. 3

[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. doi:10.1111/cgf.14433. 2

[JST*19] JAMRIŠKA O., SOCHOROVÁ Š., TEXLER O., LUKÁČ M., FIŠER J., LU J., SHECHTMAN E., SỲKORA D.: Stylizing video by example. *ACM Trans. Graph. 38*, 4 (2019), 1–11. doi:10.1145/3306346.3323006. 2, 3, 11

[KMM*02] KALNINS R. D., MARKOSIAN L., MEIER B. J., KOWALSKI M. A., LEE J. C., DAVIDSON P. L., WEBB M., HUGHES J. F., FINKELSTEIN A.: Wysiwyg npr: Drawing strokes directly on 3d models. *ACM Trans. Graph. 21*, 3 (2002), 755–762. doi:10.1145/566654.566648. 3

[Kor02] KORT A.: Computer aided inbetweening. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering* (2002), ACM, pp. 125–132. doi:10.1145/508530.508552. 2

[LA15] LANG K., ALEXA M.: The markov pen: Online synthesis of free-hand drawing styles. In *Non-Photorealistic Animation and Rendering* (2015), The Eurographics Association. doi:10.2312/exp.20151193. 3

[LCY*11] LIU D., CHEN Q., YU J., GU H., TAO D., SEAH H. S.: Stroke correspondence construction using manifold learning. *Computer Graphics Forum 30*, 8 (2011), 2194—-2207. doi:10.1111/j.1467-8659.2011.01969.x. 2

[LGH13] LANDES P.-E., GALERNE B., HURTUT T.: A shape-aware model for discrete texture synthesis. *Computer Graphics Forum 32*, 4 (2013), 67–76. doi:10.1111/cgf.12152. 2, 3

[LSF10] LU J., SANDER P. V., FINKELSTEIN A.: Interactive painterly stylization of images, videos and 3d animations. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2010), ACM, p. 127–134. doi:10.1145/1730804.1730825. 5

[LWZ*04] LIU L., WANG G., ZHANG B., GUO B., SHUM H.-Y.: Perceptually based approach for planar shape morphing. In *12th Pacific Conference on Computer Graphics and Applications* (2004), pp. 111–120. doi:10.1109/PCCGA.2004.1348341. 2

[LYFD12] LU J., YU F., FINKELSTEIN A., DIVERDI S.: Helpinghand: Example-based stroke stylization. *ACM Trans. Graph. 31*, 4 (2012). doi:10.1145/2185520.2185542. 3

[LZLS21] LI X., ZHANG B., LIAO J., SANDER P.: Deep sketch-guided cartoon video inbetweening. *IEEE Transactions on Visualization and Computer Graphics* (2021). doi:10.1109/TVCG.2021.3049419. 3

[MFXM21] MIYAUCHI R., FUKUSATO T., XIE H., MIYATA K.: Stroke correspondence by labeling closed areas. In *2021 Nicograph International* (2021), IEEE Computer Society, pp. 34–41. doi:10.1109/NICOINT52941.2021.00014. 2

[MWLT13] MA C., WEI L.-Y., LEFEBVRE S., TONG X.: Dynamic element textures. *ACM Trans. Graph. 32*, 4 (2013). doi:10.1145/2461912.2461921. 2, 3

[MWT11] MA C., WEI L.-Y., TONG X.: Discrete element textures. *ACM Trans. Graph. 30*, 4 (2011). doi:10.1145/2010324.1964957. 3

[NSC*11] NORIS G., SÝKORA D., COROS S., WHITED B., SIMMONS M., HORNUNG A., GROSS M., SUMNER R. W.: Temporal noise control for sketchy animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering* (2011), ACM, p. 93–98. doi:10.1145/2024676.2024691. 2, 7, 11, 12

[PGS*19] PEIRCE J., GRAY J. R., SIMPSON S., MACASKILL M., HÖCHENBERGER R., SOGO H., KASTMAN E., LINDELØV J. K.: Psychopy2: Experiments in behavior made easy. *Behavior research methods 51* (2019), 195–203. 12

[Ree81] REEVES W. T.: Inbetweening for computer animation utilizing moving point constraints. In *Proceedings of the 8th Annual Conference on Computer Graphics and Interactive Techniques* (1981), ACM, p. 263–269. doi:10.1145/800224.806814. 2

[RÖM*15] ROVERI R., ÖZTIRELI A. C., MARTIN S., SOLENTHALER B., GROSS M.: Example based repetitive structure synthesis. *Computer Graphics Forum 34*, 5 (2015), 39–52. doi:10.1111/cgf.12695. 3

[RSP22] RISO M., SFORZA D., PELLACINI F.: pop: Parameter optimization of differentiable vector patterns. *Computer Graphics Forum 41*, 4 (2022), 161–168. doi:https://doi.org/10.1111/cgf.14595. 3

[SDC09] SÝKORA D., DINGLIANA J., COLLINS S.: As-rigid-as-possible image registration for hand-drawn cartoon animations. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering* (2009), ACM, p. 25–33. doi:10.1145/1572614.1572619. 2, 3

[SGWM93] SEDERBERG T. W., GAO P., WANG G., MU H.: *2-D Shape Blending: An Intrinsic Solution to the Vertex Path Problem*. ACM, 1993, p. 15–18. 2

[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. doi:10.1111/cgf.13621. 3

[SLL*22] SIYAO L., LI Y., LI B., DONG C., LIU Z., LOY C. C.: Animerun: 2d animation visual correspondence from open source 3d movies. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track* (2022). 3

[SZY*21] SIYAO L., ZHAO S., YU W., SUN W., METAXAS D., LOY C. C., LIU Z.: Deep animation video interpolation in the wild. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 6583–6591. doi:10.1109/CVPR46437.2021.00652. 3

[TFK*20] TEXLER O., FUTSCHIK D., KUČERA M., JAMRIŠKA O., ŠÁRKA SOCHOROVÁ, CHAI M., TULYAKOV S., SÝKORA D.: Interactive video stylization using few-shot patch-based training. *ACM Trans. Graph. 39*, 4 (2020). doi:10.1145/3386569.3392453. 2, 3, 11

[TWY*20] TU P., WEI L.-Y., YATANI K., IGARASHI T., ZWICKER M.: Continuous curve textures. *ACM Trans. Graph. 39*, 6 (2020). doi:10.1145/3414685.3417780. 3

[TWZ22] TU P., WEI L.-Y., ZWICKER M.: Clustered vector textures. *ACM Trans. Graph. 41*, 4 (2022). doi:10.1145/3528223.3530062. 2, 3

[WLKT09] WIE L.-Y., LEFEBVRE S., KWATRA V., TURK G.: State of the art in example-based texture synthesis. In *Eurographics 2009 - State of the Art Reports* (2009), The Eurographics Association, pp. 93–117. doi:10.2312/egst.20091063. 3

[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. doi:10.1111/j.1467-8659.2009.01630.x. 2

[XWSY15] XING J., WEI L.-Y., SHIRATORI T., YATANI K.: Autocomplete hand-drawn animations. *ACM Trans. Graph. 34*, 6 (2015). doi:10.1145/2816795.2818079. 1, 2

[Yan18] YANG W.: Context-aware computer aided inbetweening. *IEEE Transactions on Visualization and Computer Graphics 24*, 2 (2018), 1049–1062. doi:10.1109/TVCG.2017.2657511. 2

[YBS*12] YU J., BIAN W., SONG M., CHENG J., TAO D.: Graph based transductive learning for cartoon correspondence construction. *Neurocomput. 79* (2012), 105–114. doi:10.1016/j.neucom.2011.10.003. 2

[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. doi:10.1111/cgf.13518. 2

[ZLWH16] ZHU H., LIU X., WONG T.-T., HENG P.-A.: Globally optimal toon tracking. *ACM Trans. Graph. 35*, 4 (2016). doi:10.1145/2897824.2925872. 2

[ZPBK17] ZHU Y., POPOVIĆ J., BRIDSON R., KAUFMAN D. M.: Planar interpolation with extreme deformation, topology change and dynamics. *ACM Trans. Graph. 36*, 6 (2017). doi:10.1145/3130800.3130820. 3

Appendix A: Batched stroke distribution synthesis algorithm

Thanks to our GPU-based implementation, each update of both the density map and covered densities can be achieved in real time. However, the numerous transfers from GPU to CPU memory are still too expensive for most applications. To address this issue, we propose an extended version of the synthesis algorithm described in Section 3.3 that selects multiple candidate strokes in batches. This way, the density map and covered densities are only updated and read back from the GPU between batches.

Since we cannot rely on the density map to prevent aggregation of strokes within a batch $\hat{\mathcal{S}}_b$, we need another distribution criterion that should be (1) an approximation of the density already covered by the strokes in $\hat{\mathcal{S}}_b$ in the footprint of a new candidate stroke $\hat{S}$, and (2) very fast to evaluate. For this purpose, we approximate the stroke footprint by their Axis-Aligned Bounding Box (AABB), and we count how many times the AABB of $\hat{S}$ overlaps AABBs from the strokes in $\hat{\mathcal{S}}_b$. The candidate stroke is only accepted if this number is lower than $\gamma\,\mathcal{C}(\hat{S})$, with $\gamma \in [0,1]$ a user parameter. In practice, we use $\gamma = 1/2$ in all our results.

The batch size is dynamically determined as a fraction of the interpolated total length $L^\star$. The total length of the selected strokes in a batch $L(\hat{\mathcal{S}}_{b_1}) + L(\hat{\mathcal{S}}_{b_2})$ must be lower than $\omega\,L^\star$, where the parameter $\omega \in (0,1]$ controls the batch maximum size. When $\omega \to 0$, the selection reverts to the non-batch strategy, which is slow. When $\omega \to 1$, many strokes are selected without updating the density map, which can lead to visual artifacts. Empirically, we found that a good balance is achieved when $\omega = 1/4$.

Algorithm 2 recaps the batched stroke selection process, where we use $\{\mathcal{S}, \mathcal{S}_b\}$ to denote the union of the sets $\mathcal{S}$ and $\mathcal{S}_b$. There are a number of differences with Algorithm 1. First, at each iteration of the outer loop, we initialize a set $\mathcal{S}'_r$ of strokes (r standing for "removed") that keeps track of all the strokes that have already been considered. The innermost loop looks for a stroke that has not yet been considered (i.e., in $\mathcal{S}' \setminus \mathcal{S}'_r$) that can be added according to the AABB criterion. Once it is found, it is not directly added to the set of selected strokes, but instead to $\hat{\mathcal{S}}_b$, the batch set of selected strokes. Only when the batch has been fully processed do we add its selected strokes to $\hat{\mathcal{S}}$ and update the density.

Algorithm 2: Batched stroke distribution synthesis.

Input: $\mathcal{S}'_1, \mathcal{S}'_2, \mathcal{D}^\star, L^\star_{1/2}, L^\star, \hat{\mathcal{S}}, \gamma, \omega$
Output: $\hat{\mathcal{S}}$

begin
  repeat
    $\mathcal{C}(\mathcal{S}') = \sum_i \mathcal{D}^\star(\mathbf{x}'_i),\ \forall \mathcal{S}' \in \{\mathcal{S}'_1 \cup \mathcal{S}'_2\}$
    $\hat{\mathcal{S}}_b \leftarrow \emptyset$
    $\mathcal{S}'_r \leftarrow \emptyset$
    repeat
      if $L(\{\hat{\mathcal{S}}_1, \hat{\mathcal{S}}_{b_1}\}) \le L^\star_{1/2} L(\{\hat{\mathcal{S}}_2, \hat{\mathcal{S}}_{b_2}\})$ **then**
        $\{\mathcal{S}', \hat{\mathcal{S}}_b\} \leftarrow \{\mathcal{S}'_1, \hat{\mathcal{S}}_{b_1}\}$
      else
        $\{\mathcal{S}', \hat{\mathcal{S}}_b\} \leftarrow \{\mathcal{S}'_2, \hat{\mathcal{S}}_{b_2}\}$
      strokeFound $\leftarrow$ False
      repeat
        $\hat{S} \leftarrow \arg\max_{S' \in \mathcal{S}' \setminus \mathcal{S}'_r} \mathcal{C}(S')$
        $\mathcal{S}'_r \leftarrow \mathcal{S}'_r \cup \hat{S}$
        if `OverlapCount` $(\hat{S}, \hat{\mathcal{S}}_b) < \gamma\,\mathcal{C}(\hat{S})$ **then**
          strokeFound $\leftarrow$ True
      until strokeFound ***or*** $\mathcal{S}' = \mathcal{S}'_r$
      if strokeFound ***and*** $\mathcal{C}(\hat{S}) > 0$ **then**
        $\hat{\mathcal{S}}_b \leftarrow \hat{\mathcal{S}}_b \cup \hat{S}$
        $\mathcal{S}' \leftarrow \mathcal{S}' \setminus \hat{S}$
      else
        break
    until $L(\hat{\mathcal{S}}_{b_1}) + L(\hat{\mathcal{S}}_{b_2}) \ge \omega\,L^\star$ ***or*** $L(\hat{\mathcal{S}} \cup \hat{\mathcal{S}}_b) \ge L^\star$
    if $\hat{\mathcal{S}}_b \neq \emptyset$ **then**
      $\hat{\mathcal{S}} \leftarrow \hat{\mathcal{S}} \cup \hat{\mathcal{S}}_b$
      $\mathcal{D}^\star \leftarrow \mathcal{D}^\star - \mathcal{D}_{\hat{\mathcal{S}}},\ \forall \hat{\mathcal{S}} \in \hat{\mathcal{S}}_b$
  until $L(\hat{\mathcal{S}}) \ge L^\star$ ***or*** $\hat{\mathcal{S}}_b = \emptyset$

Function `OverlapCount` $(\hat{S}, \mathcal{S})$**:**
  $n \leftarrow 0$
  forall $S \in \mathcal{S}$ **do**
    if $AABB(\hat{S}) \cap AABB(S) \neq \emptyset$ **then**
      $n \leftarrow n + 1$
  return n