Non-linear Rough 2D Animation using Transient Embeddings

Melvin Even, Pierre Bénard, Pascal Barla

EUROGRAPHICS 2023 / K. Myszkowski and M. Nießner
(Guest Editors)

Volume 42 (2023), Number 2

Non-linear Rough 2D Animation using Transient Embeddings

Melvin Even, Pierre Bénard, Pascal Barla

Inria, Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, France

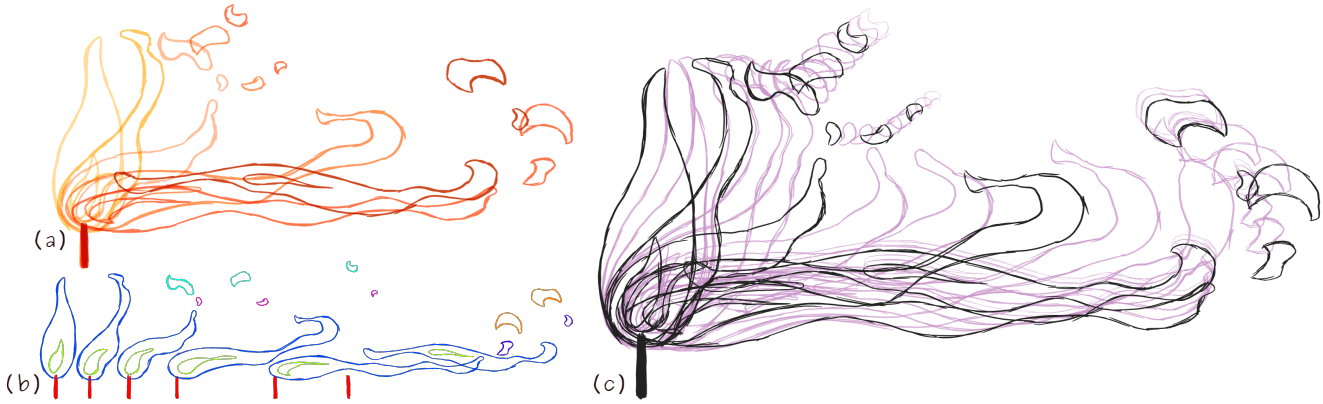


Figure 1: *Our rough animation system supports the inbetweening of complex special effects with multiple topological events, here a flame splitting into smaller components. In (a), we show the input key drawings traced over a reference from Gilland's book [Gil09], while in (b) we show their decomposition into* transient embeddings. *Even though we use a unique color per embedding throughout the animation for visualization, each embedding only exists between a pair of keyframes in this example. This allows the handling of topological changes, which occur at the third and fifth keyframes. In (c), we display in magenta a subset of the inbetween frames generated by our animation system in real-time. Please see the supplemental results video for the full sequence.*

Abstract

Traditional 2D animation requires time and dedication since tens of thousands of frames need to be drawn by hand for a typical production. Many computer-assisted methods have been proposed to automatize the generation of inbetween frames from a set of clean line drawings, but they are all limited by a rigid workflow and a lack of artistic controls, which is in the most part due to the one-to-one stroke matching and interpolation problems they attempt to solve. In this work, we take a novel view on those problems by focusing on an earlier phase of the animation process that uses rough drawings (i.e., sketches). Our key idea is to recast the matching and interpolation problems so that they apply to transient embeddings, *which are groups of strokes that only exist for a few keyframes. A transient embedding carries strokes between keyframes both forward and backward in time through a sequence of transformed lattices. Forward and backward strokes are then cross-faded using their thickness to yield rough inbetweens. With our approach, complex topological changes may be introduced while preserving visual motion continuity. As demonstrated on state-of-the-art 2D animation exercises, our system provides unprecedented artistic control through the non-linear exploration of movements and dynamics in real-time.*

CCS Concepts

• ***Computing methodologies*** → ***Graphics systems and interfaces; Animation;***

1. Introduction

Traditional 2D animation requires a lot of planning, not only at the level of storyboards, but also for the animation itself where rough drawings (i.e., sketches) are used to test out the motion of different characters or special effects (e.g., water, smoke). These tests aim at defining the trajectories of the characters or objects, as well as their dynamics (e.g., their speed and acceleration) through the timing & spacing of drawings [JT95, Wil01]. Even though the rough animation itself is not directly visible in the final movie, its impact on motion design is vividly retained, since it serves as a guide for "inbetweeners" – the artists who draw all the intermediate frames.

In this paper, we introduce a system for the design and exploration of rough 2D animations; a problem which, to the best of our knowledge, has never been addressed in previous work. Our main contribution is in the assembly and adaptation of a set of existing techniques for the previsualization of 2D animations. The main goal is to provide real-time feedback at intermediate frames between rough key drawings, to both significantly speed up the animation process and allow artists to experiment with different creative alternatives. This is useful not only for experienced animators, who may try variations in early tests for discussions with art directors and quickly converge to final rough animations to pass down to inbetweeners; but also for animation students, who may benefit from the ability to observe interactively the look-and-feel of different animation choices. As described in supplemental material, we relied on an observational study of a professional animator at work followed by interviews to guide the design of our system. By working at the rough animation stage, we leverage the fact that drawings are sketchy and the global perception of movement is more important than the appearance of the strokes themselves that will eventually be redrawn at the cleaning stage. However, it brings two fundamental challenges. First, artists may create drawings through very different workflows such as "shift-and-trace" (drawings are traced over deformations of previous ones) or "pose-to-pose" (all key drawings are created in advance then interpolated). Second, the drawings themselves are most often composed of different numbers of strokes and routinely differ in their number of parts.

As described by Fekete et al. [FBC*95], automated inbetweening systems can be divided in two main families: those based on *templates* or *embeddings* (e.g., [BW75]), and those relying on *explicit correspondences* between strokes (e.g., [MIT67]). The former family is mostly well-suited for "cut-out" animations since the movement of the embedded objects or characters is restricted by the motion of their template (e.g., skeleton, control polygon, cage) whose topology is usually fixed throughout the animation. Explicit correspondence systems are more flexible as the stroke-to-stroke mapping is *transient*, changing between each pair of keyframes. However they are restricted to "tight inbetweening" of clean line drawings due to the challenge (or chore) of matching complex networks of strokes. In this work, we propose *transient embeddings* to keep the best of both approaches, hence allowing template-based animation of rough drawings with topological changes. In practice, this requires adapting two common problems to deal with transient embeddings: the *matching problem* where two drawings must be registered, here with drawings having different numbers of strokes; and the *interpolation problem*, where the movement from one key drawing to the next must be generated while providing flexible and interactive artistic control over timing, spacing and trajectories. A key feature of our approach is to enable changes of topology at keyframes (i.e., key drawings may have different numbers of embeddings), while ensuring visual continuity through constrained trajectories that persist over multiple keyframes.

Our main contribution is a novel animation system that relies on transient embeddings to provide full *non-linear* artistic control at the rough animation stage, as described in Section 3. Methods for matching embeddings at keyframes are introduced in Section 4: they work with shift-and-trace and pose-to-pose workflows, or any combination of them. Methods for interpolating between embeddings are presented in Section 5, featuring non-linear control over timing & spacing and direct artistic control over trajectories between and across keyframes. Implementation details including a novel real-time ARAP registration technique for vector strokes are exposed in Section 6. We demonstrate that our system allows users to produce complex 2D animations in Section 7, by reproducing typical animation exercises [Wil01], such as walk cycles, special effects, articulated motion, and some principles of animation [JT95], such as anticipation and follow-through, or squash and stretch.

2. Related work

The design of computer-aided 2D animation systems dates back to the inception of Computer Graphics in the late '60s and early '70s [MIT67, Bae69, BW71]. As already observed by Catmull [Cat78], automatic inbetweening is a central problem tackled by most of such systems, and yet – more than forty years later – current commercial solutions [Ado, Too, CAC, Com] often remain too limited or time consuming for most use cases in production. In addition, by focusing on inbetweening of final clean line drawings, we believe that those systems and most previous work in academia have missed the real potential benefit of computer assistance, that is, using the words of Durand [Dur91], to *"boost user creativity by allowing them to concentrate on the most interesting part of their work"*: the design and exploration of motion.

The key idea of our approach is to shift focus from the animation of individual strokes to the animation of groups of strokes that are only defined between a pair of keyframes, which we call *transient embeddings*. Nevertheless, it requires to revisit the two main stages of explicit correspondence techniques: *matching* and *interpolation*.

Matching. Most early methods require the user to manually identify correspondences between the strokes of consecutive keyframes and do not handle occlusions or topological changes [MIT67, Bae69, Ree81, Dur91]. More recent techniques support such features using manually populated 2.5D [DFSEVR01, RID10] or space-time [DRvdP15] data structures. Despite their appeal, these approaches require ad hoc, rather constraining drawing representations which are not suitable for rough drawings. To partially automatize the stroke correspondence process, a large body of work represents the drawings as a graph of strokes and try to match those graphs at subsequent keyframes [Kor02, WNS*10, LCY*11, YBS*12, CMV17, Yan18, YSC*18, MFXM21]. They differ by the graph matching algorithm they employ, and the way users interact with the system to guide or correct correspondences, especially when strokes appear or disappear. To resolve occlusions in a user-controllable fashion, Jiang et al. [JSL22] introduce "boundary strokes", i.e., strokes with an occluding side that acts as occluding surfaces. However, none of these methods can handle rough drawings with an highly dissimilar number of strokes per keyframe, and despite recent advances in rough sketch cleanup [YVG20], no algorithm is currently able to produce a sequence of clean line drawings that can be automatically inbetweened.

Alternatively, some methods aim at estimating region (rather than stroke) correspondences between consecutive frames based on their appearance (color, shape, distance) [Xie95, MSG96, SBv05, dJB06, BBA09, ZHF12, LMY*13] and motion features [ZLWH16],

but they are limited to polygonal shapes or cel animations (i.e., mostly flat color regions with clean line boundaries). Taking inspiration from As-Rigid-As-Possible (ARAP) shape deformation techniques [IMH05, WXXC08], Sýkora et al. [SDC09] present an image registration algorithm that decouples the matching resolution from the image complexity by embedding it into a square lattice. Noris et al. [NSC*11] use this method to estimate a global warp between two drawings of an existing rough animation, abstracting the input strokes by their rasterized distance fields. Then, each stroke of the first drawing, deformed by the ARAP transformation, is matched with the most similar stroke in the second one. We also embed strokes into square lattices, but extend the registration algorithm to directly take as input vectorial strokes. Furthermore, we make the assumption that stroke-to-stroke correspondences are not required to depict motion in rough animations, which we demonstrate in our results.

Closest to our work, Xing et al. [XWSY15] present an interactive system that combines a global shape similarity metric with an embedding-free ARAP deformation model [SSP07] to match an existing drawing with a new set of hand-drawn guidelines. We discuss the benefits of our explicit embeddings in Section 8.1 and provide visual comparison in the supplemental results video.

Following the current trend in computer science, learning-based techniques [Yag17, NHA19, LZLS21, SZY*21] have also been proposed to estimate per-pixel correspondence between two raster clean line drawings. In the work of Casey et al. [CPL21], line-enclosed segments are first extracted from the two drawings, and then correspondences between segments are estimated using a combination of convolutional and transformer neural networks. Extending such approaches to rough drawings, whose style may considerably vary from one artist to another, seems extremely difficult for such data-driven approaches.

Interpolation. Once the key drawings have been put into correspondence, inbetween frames can be generated by interpolation. As already noted by Burtnyk et al. [BW75], linear interpolation and thus linear trajectories sampled at uniform rates do not produce natural motion in the great majority of cases.

To offer maximum artistic control, the animation system of Reeves [Ree81] allows the user to specify the trajectory and dynamics of a set of "moving points" spanning multiple keyframes. This effectively decomposes the full 2D+t space of the animation into a network of Coons patches, into which interpolation can be performed independently but with continuity at boundaries. However heuristics are required to complete the patch network, and user manipulation of moving points in space and time may be laborious.

Kort [Kor02] models trajectories of stroke vertices by quadratic splines. The user can correct these paths when needed and specify their spacing. Since this simple interpolation scheme does not take the shape of the strokes into account, it may lead to local or global distortions. Sederberg et al. [SGWM93] introduce an intrinsic interpolation technique which minimizes shape distortion. Similar approaches [FTA05, SZGP05] attempt to preserve local differential quantities (Laplacian coordinates or edge deformation gradients). But those three methods only apply to a single polyline. Motivated by classical 2D animation books [JT95, Wil01], Whited et al. [WNS*10] present an interpolation scheme that produces arc trajectories for a full graph of strokes. It first computes motion paths for stroke endpoints along logarithmic spirals, and then deform the intermediate stroke vertices using intrinsic interpolation [SGWM93] followed by curve fitting and a tangent-aligning warp to ensure continuity between adjacent strokes. The trajectory of any stroke vertex can be edited, albeit without considering its dynamics. This scheme was later used by Noris et al. [NSC*11] for generating smooth stroke trajectories between pairs of strokes.

An alternative solution to minimize shape distortion is to rely on ARAP interpolation [ACOL00, XZWB05] of 2D embeddings of the drawings. The interpolated trajectories can be controlled through point and vector constraints [BBA08, KHS*12] or even a full skeleton [YHY19]. However, the boundary polygon of those embeddings must be compatible across keyframes and put into correspondence, and a compatible triangulation of their interior must also be built. Baxter et al. [BBA09] describe the most relevant techniques to solve this challenging problem along with their own solution. Zhu et al. [ZPBK17] extend these approaches to handle extreme shape deformations and topological changes, but it requires significant manual intervention and involves an expensive numerical optimization that prevents its use in an interactive system.

Yang [Yan18] combines the strokes deformation technique of Whited et al. [WNS*10] with a simpler embedding, called "context mesh", that better preserves the global layout of the stroke network. He presents an automatic construction algorithm of these compatible meshes based on the matched strokes, and an edge-based rigid interpolation technique inspired by the method of Igarashi et al. [IMH05] which is robust to degenerated configurations (e.g., collapsing edges) and may be constrained to follow a given trajectory. It is however unclear how such "context meshes" could be built for rough drawings. Instead, we use even simpler lattice embeddings, which are compatible between two keyframes by construction, but do not need to extend further in time.

Dvorožnák et al. [DLKS18] use similar embeddings to build deformable puppets, but since those are connected at fixed junctions driven by a skeleton, their results suffer from the "cut-out" look-and-feel. The animation system of Bai et al. [BKLP16] integrates handle-based shape deformations with example-based simulations to interpolate drawings embedded into triangular meshes. This approach manages to reproduce many of the principles of animation [JT95], supports manual topological changes and local control of the dynamics, but user interaction is restricted to handles manipulation, hence once again following the "cut-out" metaphor rather than hand-drawn animation.

For image-based approaches, interpolation turns into an image morphing (i.e., deformation and blending) problem. Many solutions have been proposed for photographs (e.g., [FZP*20, PSN20]), cartoon animations [LZLS21, SZY*21, CZ22] and, closest to our inputs, concept sketches [ADN*17]. Yet, rough drawings have a very specific style which requires preserving the distribution, spatial continuity and color or gray-level intensity of the strokes. Previous approaches are unlikely to satisfy all three criteria. In this work, we do not attempt to solve this problem, and use simple cross-fading of stroke thicknesses that turns out to be sufficient for motion previsualization in practice.

3. Animation system

Most computer-aided 2D animation systems (e.g., [TVP, Too, Ado, CAC]) provide a timeline which displays the timing (i.e., frame number) and the layers in a simplified and systematic manner. Layers are organized in a stack defining the compositing order of the drawings on the canvas. Each layer is animated independently and populated by keyframes that may hold a drawing and be "exposed" (i.e., repeated) over multiple frames.

The design of our animation system is guided by extended discussions with a professional 2D animator and 2D animation software developers. As detailed in supplemental material, we arrived at the following conclusions and thus guidelines:

G1: Contours in rough drawings are depicted with multiple strokes which implies working with groups of strokes rather than individual elements; the artist must be able to redefine the number of groups according to the complexity of the motion.

G2: Topological changes, such as parts of the drawing appearing, disappearing, splitting or merging, are explicitly drawn at keyframes, hence do not need to be automatically in-betweened.

G3: Artists alternately use pose-to-pose and shift-and-trace workflows that must be both supported; they are used to provide indications to inbetweeners, but their creative flow should be interrupted as least as possible.

G4: Control over timing, spacing and interpolation trajectories is crucial to design motion, yet extremely complex and time-consuming since all intermediate frames must be redrawn.

Based on these guidelines, we extend the structure of a classic animation system to work on groups of strokes, which we call *embeddings*, as detailed in the remainder of this section. Embeddings act as units of motion (e.g., the forearm of a character), which may be refined throughout the animation process when needed (**G1**). Each embedding is transformed from its start keyframe to the next, carrying along its strokes so that they come into alignment with a different subpart of the next key drawing. Embeddings may thus be said to be *transient* in the sense that they do not last past the next keyframe, where new embeddings take over the animation process, possibly with a different topology (**G2**). This novel animation structure is well adapted to existing artistic workflows (**G3**) and grants new non-linear controls (**G4**), as described in Sections 4 and 5 respectively.

3.1. Transient embeddings

In its simplest form, as shown in Figure 2(a), an embedding is defined by a pair of lattices with the same topology at two keyframes. The lattice at the start keyframe holds strokes (a subset from the corresponding key drawing) that are propagated forward in time. A second set of strokes is stored in a transformed lattice at the end keyframe, which is lined up in time with the next keyframe. However, the end keyframe itself is never displayed. The transformation between the two lattices must be *invertible* so that strokes from the end keyframe can be propagated backward in time. The two sets of forward and backward strokes are cross-faded using stroke thickness instead of opacity. Such a representation opens up a number of possibilities. For instance, backward strokes may be obtained by copying a subset of the strokes in the next key drawing, hence avoiding popping artifacts when transitioning from one embedding to the next. But they may also differ significantly from the strokes found in the next keyframe, hence allowing topological changes to occur, as demonstrated in Section 3.3. However, in our system, topological changes never occur between keyframes.

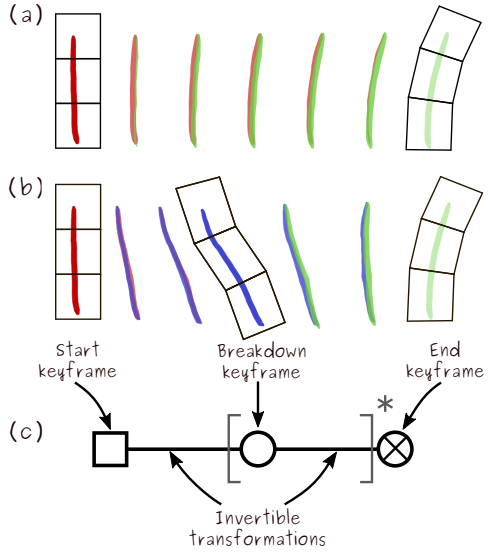


Figure 2: *A transient embedding is a sequence of transformed lattices carrying strokes between keyframes. (a) A first set of strokes (in red), stored in the start keyframe, is propagated forward via a transformation, while another set of strokes (in green), stored in the end keyframe, is propagated backward via the inverse transformation. Forward and backward strokes are cross-faded in between. (b) An additional inbetween lattice transformation may be added through a breakdown keyframe, which holds a single set of strokes (in blue) that are propagated both backward and forward in time. (c) Symbolic representation of a transient embedding: each type of keyframe is represented by a different symbol, which are connected by segments that represent lattice transformations. The regular expression notation $[\cdots]^*$ indicates that breakdowns are optional.*

A same embedding may be used over multiple keyframes, as shown in Figure 2(b). To make this possible, we introduce *breakdown keyframes*, inspired by the traditional animation technique of the same name [Wil01]. In our system, they amount to storing a new transformed lattice in the embedding, along with an additional set of strokes that is propagated both backward and forward in time.

Figure 2(c) abstracts the structure of a transient embedding with a simple sequence of symbols: a square for the start keyframe, circles for (optional) breakdown keyframes, and a crossed circle for the end keyframe. In effect, the end keyframe is a special case of breakdown that is only propagated backward in time, and is not itself displayed. Segments between symbols represent both forward and backward embedding transformations, which may be evaluated at any time step to yield cross-faded strokes.

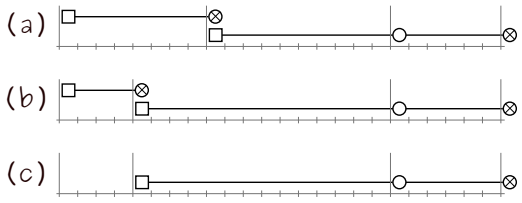


Figure 3: *In our animation system, the timeline is segmented into intervals whose boundaries are shown with tall ticks, while short ticks delimit frames. An animation is created by (a) adding transient embeddings that span one or more intervals, with their keyframes lined up at the beginning of each interval. Timing is readily modified by (b) automatically updating embeddings when interval boundaries are moved. Edits are local since (c) the removal of an embedding does not affect other embeddings.*

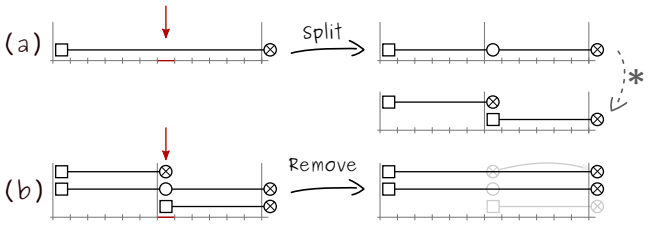


Figure 4: *Non-linear editing of the animation structure is made possible by special updates of the transient embeddings. Splitting an interval in two is done by (a) inserting a breakdown keyframe in all involved embeddings, which may optionally be converted into an end keyframe followed by a new, automatically generated embedding. Removing an interval yields (b) three different types of results depending on the type of keyframe: extension of the embedding, removal of a breakdown, or removal of the embedding.*

3.2. Animation structure

With this formal definition of a single transient embedding at hand, we must now formalize the global animation structure used in our system. This is required for a non-linear animation system as we need to ensure that a valid animation structure is maintained irrespective of the order in which operations are performed on it.

The first structural constraint is that the timeline must be segmented into contiguous intervals, which act as containers for embeddings. Intervals may hold zero, one or more embeddings. The keyframes of all embeddings in the same layer must be lined up in time with the first frame of an existing interval, as shown in Figure 3(a). Interval boundaries may then be modified, carrying with them all the embeddings they store, as shown in Figure 3(b). This amounts to modifying the timing of an animation, in which case embeddings must be evaluated at different time steps. An additional advantage of this structure is that removing an embedding does not affect the rest of the animation as illustrated in Figure 3(c), hence making the animation process globally non-destructive.

Intervals that do not contain any embedding may be safely deleted or split into two intervals. New intervals may similarly be added before or after existing ones. A more complex situation arises when the same operations are applied to an interval that contains one or more embeddings. When splitting an interval in two, each of its stored embeddings must also be split at the same frame to preserve the aforementioned keyframe alignment constraint. As shown in Figure 4(a), we do so by inserting a breakdown keyframe for each involved embedding. These breakdown keyframes may be converted into end keyframes if needed, which results in the automatic creation of a new embedding as shown in the second row of the figure. When removing a key interval, all the involved embeddings must be updated according to three different rules, depending on the type of keyframe involved. This is illustrated in Figure 4(b): an end keyframe is not removed but merely extended to the start frame of the next interval; removing a breakdown keyframe requires reconnecting the start or previous breakdown keyframe to the next breakdown or end keyframe; when a start keyframe is removed, there is no other choice than deleting the entire embedding since there is no key drawing to rely on anymore.

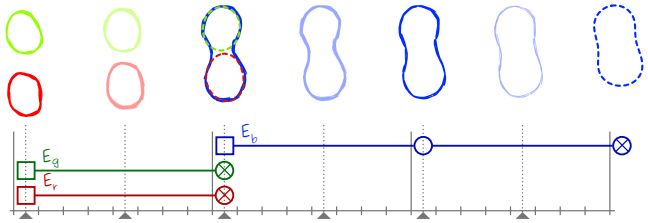


Figure 5: *We illustrate our animation system on a simple example made of three intervals. A pair of green E_g and red E_r embeddings meet a blue embedding E_b at the second keyframe, where the topology of the drawing changes. This is achieved by using backward strokes for E_g and E_r (dashed lines) that are different from forward strokes for E_b. The latter spans the two remaining intervals via a breakdown keyframe, which enables additional deformation. The end keyframe of E_b is empty and strokes are faded out. The gray triangles indicate the frames at which the above embeddings are evaluated, both at and in between keyframes.*

3.3. A simple example

We conclude this section with a simple example animation, shown in Figure 5. It demonstrates several structural properties of our animation system: a topological change with different forward/backward strokes, a breakdown keyframe, and a fade out. End keyframes are shown as dashed lines since they are not displayed, but their strokes are propagated backward to be used for interpolation. We also rely on this example for illustration purposes in Sections 4 and 5, and in the supplemental demo video.

Such a simple example leaves open a number of questions that we address in the next two sections. They precisely correspond to the matching and interpolation problems recast on transient embeddings: How are embeddings deformed *at* keyframes so that backward and forward strokes are put in alignment, yielding visually continuous animations (Section 4)? How are embeddings interpolated *between* keyframes, while providing control over motion dynamics, trajectories and smoothness (Section 5)?

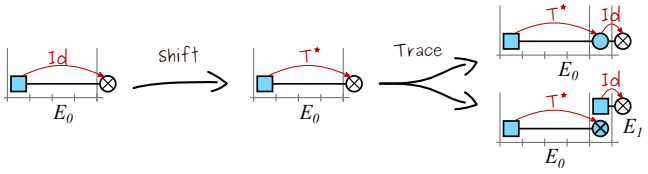
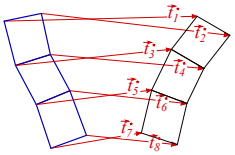


Figure 6: *With the shift-and-trace workflow, an initial embedding E_0 is first manually modified through a target transformation $T^\star$ – the shift step. The embedded strokes (symbolized by the light blue color) are then copied either to a breakdown keyframe, or to a new embedding E_1 – the trace step.*

4. The matching problem

Recall that an embedding in our system corresponds to a unit of motion. By default, all strokes in a key drawing are part of a single embedding, but this may be refined at any time by the user to assign different subsets of strokes to different embeddings. This choice must be made based on which strokes of the current keyframe are expected to move together to the next keyframe, i.e., based on motion complexity rather than the complexity of the drawing itself.

As shown in Figure 2, each embedding stores its strokes in a square lattice, which is built axis-aligned at the start keyframe. Recall that the lattice does not change topology throughout a transient embedding, but may undergo transformations at breakdown and end keyframes. The main goal of the matching problem is then to establish lattice transformations that put embedded strokes in alignment with strokes at the next keyframe. Such a *target* transformation $T^\star$ consists in a discrete vector field $\{\vec{t}_i^\star\}$ defined on lattice corners (see inset figure). We present two basic matching approaches, inspired by traditional animation workflows: shift-and-trace (Section 4.1), and pose-to-pose (Section 4.2).



4.1. Shift-and-trace

The main idea of the shift-and-trace workflow is to deform a lattice to the next keyframe (the "shift" step), and then to redraw a similar set of strokes over the deformed, embedded ones (the "trace" step). Any combination of tools may be used to yield the transformation $T^\star$. We have implemented classic linear transformations (translation, rotation and scaling), as well as an ARAP deformation tool similar to the "Warp tool" in TVPaint Animation [TVP].

The shift-and-trace workflow is illustrated in Figure 6. An embedding E_0 is subject by default to an identity transformation, and acts on a set of strokes, symbolized by the light blue color. The initial configuration is thus a static drawing exposed over the whole interval, similarly to traditional animation systems. During the *shift* step, a target transformation $T^\star$ is applied to E_0, but no stroke is stored yet in its end keyframe. The *trace* step creates a new interval and populates it in one of the following two ways: the embedded strokes may either be duplicated to a new breakdown keyframe, thus extending E_0 in time; or a new embedding E_1 may be created, with embedded strokes separately copied to the end keyframe of E_0 and the start keyframe of E_1.

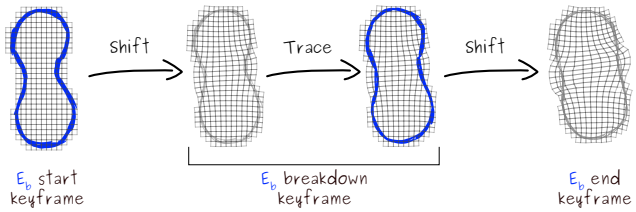


Figure 7: *The lattice of E_b is first transformed using ARAP deformation from the start to the breakdown keyframe. New, slightly different strokes are then traced in the breakdown. Finally, ARAP deformation is once again used to yield the lattice transformation in the end keyframe, which is tagged as fading out.*

Figure 8: *In the pose-to-pose workflow, the initial embeddings E_0 and E_1 are provided as input, each storing its own set of embedded strokes (symbolized by light green and light blue colors). A single registration step is used to automatically find a target transformation $T^\star$ that aligns the embedded strokes of E_0 with a subset of strokes in E_1, and to copy the latter to the end keyframe of E_0.*

At the end of the shift-and-trace process, we end up with a new interval in a configuration similar to the one we started with; the process may thus be repeated to produce a longer animation sequence. The embedded strokes of any keyframe may be redrawn at any time during that process, allowing artists to add visual complexity. Redrawing strokes in a breakdown or end keyframe is constrained to lie within the deformed lattice, since the new strokes must be propagated forward and/or backward using the same lattice topology. Redrawing strokes in the start keyframe of a new embedding does not impose any constraint since the newly created lattice can be trivially extended to cover the new strokes, which are only propagated forward.

The shift-and-trace workflow is illustrated in Figure 7 on the blue embedding E_b of Figure 5, which spans two intervals.

4.2. Pose-to-pose

The pose-to-pose workflow instead starts from an existing sequence of drawings (i.e., "poses"), each stored in its respective keyframe, and registers subsets of strokes from one keyframe to the next. Consequently, the target transformation $T^\star$ is not manually specified by the artist but automatically inferred from a pair of drawings.

The pose-to-pose workflow is detailed in Figure 8. A sequence of two embeddings E_0 and E_1 is shown, each with an identity transformation by default, but acting on different sets of strokes symbolized by different colors. As before, the initial configuration is identical to the one found in traditional animation systems: a sequence of static drawings exposed over their whole interval. Next, a *vector registration* algorithm – adapting the method of

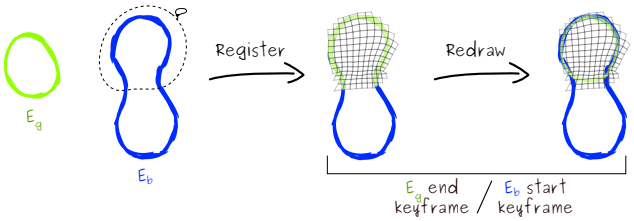


Figure 9: *A subset of strokes is first selected in E_b to be matched by E_g. Then registration automatically finds a lattice transformation for E_g that aligns strokes in both embeddings, and matching strokes in the start keyframe of E_b are copied to the end keyframe of E_g. We then partially redraw these strokes.*

Sỳkora et al. [SDC09] to vector drawings to yield real-time performance (see Section 6) – is used to find the target transformation $T^\star$ that best aligns the strokes embedded in E_0 with any user-selected subset of strokes in E_1, which are immediately copied inside the end keyframe of E_0. The transformed forward strokes of E_0 may alternatively be copied to its end keyframe.

Once again, this process ends up with an interval in a configuration similar to the one it started with, such that registrations may be applied over and over again to match pairs of drawings over several keyframes. Shift-and-trace and pose-to-pose workflows may even be interleaved since they both end up in the same configuration. Moreover, the transformation $T^\star$ found through registration may optionally be adjusted manually or semi-automatically – thanks to the real-time performance of our vector registration algorithm – at any time during that process, while the strokes in any keyframe might be redrawn as before.

The pose-to-pose workflow is illustrated in Figure 9 on the embeddings E_g and E_b of Figure 5 (a similar process is used to match E_r with E_b), with an additional redrawing step.

5. The interpolation problem

In our system, interpolation is available at all times. As shown in the demo video, after little interaction (sometimes a single click) our system provides instant interpolation results. The process is trivial for newly created embeddings since their lattice is static (e.g., *Id* arrows in Figures 6 and 8). As soon as a target transformation $T^\star$ is introduced, we must find an invertible family of transformations $T(t)$ that satisfies $T(0) = \{\vec{0}\}$ and $T(1) = T^\star$. We choose ARAP interpolation [ACOL00] for $T(t)$ since it provides a natural default behavior in general, while enabling artistic control through additional constraints. This should not be mistaken with the ARAP deformation or registration tools of Section 4. Indeed, *matching and interpolation are decoupled* in our approach.

In practice, we use the symmetric formulation of Baxter et al. [BBA08] which is fast to evaluate, easy to implement, and offers control through linear constraints. We triangulate the axis-aligned lattice and store the initial and transformed coordinates of its corners in two matrices V_0 and V_1. By construction, we have $T^\star = V_1 - V_0$, and the time-varying transformation is then defined

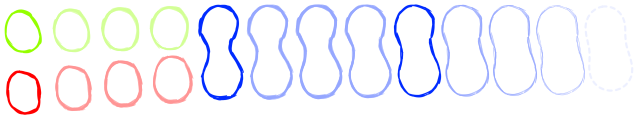


Figure 10: *Applying the default ARAP interpolation to the simple example of Figure 5 produces a reasonable initial result, but lacks control over spacing and trajectories.*

by $T(t) = V(t) - V_0$, with $V(t)$ obtained by ARAP interpolation between V_0 and V_1 (see Appendix A for details). Similarly, the inverse lattice transformation is $T(t)^{-1} = V(1-t) - V_1$.

With this formulation, we obtain results such as in Figure 10, where embedded strokes are transformed forward and backward according to T and T^{-1} respectively. A first limitation of this default interpolation behavior is that the spacing of interpolated drawings is not readily controlled. We introduce control over spacing in Section 5.1 through a formalization of the "spacing chart", which is ubiquitous in traditional 2D animation. A second limitation is that trajectories are only defined per embedding, which may lead to velocity discontinuities across keyframes, or to dissociation of embeddings between keyframes. We address these issues in Section 5.2 by sharing trajectories across embeddings and keyframes.

5.1. Spacing charts

Providing control over spacing through interpolation is crucial to convey motion dynamics. In existing animation software (e.g., [CAC]), interpolated drawings may be manually positioned on a per-frame basis, which is inspired by charts in traditional animation [Wil01]. The process may be tedious, even though presets (e.g., "ease in / ease out") can help. Furthermore, it does not grant non-linear control over the animation since it must be redone after each retiming. Alternatively, animation curves may be directly edited (e.g., [Ado, Too]), but this looses the intuitive appeal of spacing charts. We keep the best of both worlds by formalizing the spacing chart in a way that preserves editing through ticks, while allowing automatic updates after retiming.

A spacing chart is defined as a monotonically increasing 1D remapping function $S : [\mathrm{f}_0, \mathrm{f}_1] \to [0,1]$, where f_0 and f_1 are natural numbers locating two contiguous keyframes on the timeline. As illustrated in Figure 11(a-b), we let users manually adjust each tick value $S(\mathrm{f})$ for any discrete interpolated frame $\mathrm{f} \in \{\mathrm{f}_0, ..., \mathrm{f}_1\} \subset \mathbb{N}$, while enforcing $S(\mathrm{f}-1) < S(\mathrm{f}) < S(\mathrm{f}+1)$. In contrast, the spacing function S is defined over the real range $[\mathrm{f}_0, \mathrm{f}_1]$. In practice, we use a monotone cubic piecewise function [FC80] for S. It interpolates the prescribed $S(\mathrm{f})$ at discrete frame values, with tangents automatically computed to ensure smoothness and monotonicity.

The main advantage of this smooth continuous representation is non-linear editing. When an interval is extended for retiming, we linearly stretch S over the new $[\mathrm{f}'_0, \mathrm{f}'_1]$ range to get S', which is then evaluated at discrete frame positions $\mathrm{f} \in \{\mathrm{f}'_0, ..., \mathrm{f}'_1\} \subset \mathbb{N}$ to yield the new chart ticks $S'(\mathrm{f})$, as demonstrated in Figure 11(b-c). Other timeline operations, such as those depicted in Figure 4, induce straightforward operations on spacing functions, since these

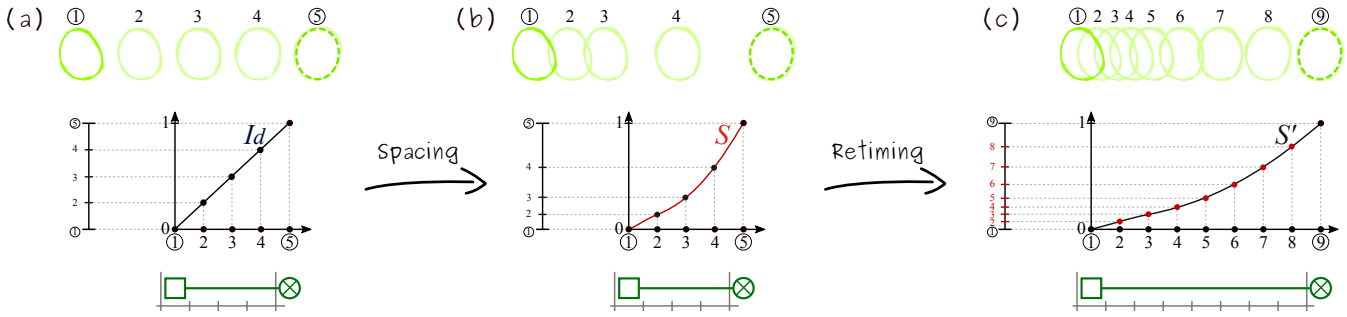


Figure 11: *We illustrate our spacing function on the green embedding E_g. By default, we use (a) the identity spacing function, resulting in a linear time parameter t. It may be modified through (b) user-provided ticks in the vertical spacing chart at left, which results in a new spacing function S (in red). After retiming (c), S is stretched to S' and new ticks on the spacing chart (in red) are automatically recomputed.*

are defined on frame ranges. Splitting an interval merely results in splitting a spacing function in two functions. Removal of an end keyframe results in an extension of the embedding which is equivalent to a retiming, whereas removal of a breakdown keyframe results in the merging of the two neighboring spacing functions.

The spacing function outputs a normalized time value that is used to compute $V(S(\mathrm{f}))$, the interpolated lattice at frame f. By construction, $V(S(\mathrm{f}_0)) = V_0$ and $V(S(\mathrm{f}_1)) = V_1$. Note that even a linear spacing function may not lead to a transformation that behaves exactly linearly between V_0 and V_1, even though they are close in practice. Indeed, lattice corners will not move at constant speed owing to the global optimization involved in ARAP interpolation. However, constrained trajectories may be added if a precise local control over motion speed is required, as explained next.

5.2. Constrained interpolation

Besides spacing, the default ARAP interpolation may be controlled by linear constraints, $CV = D(t)$ (see Appendix A). The C matrix identifies which point of the embedding should be subject to a constraint through time. Any point inside the lattice may be used as a constraint (even away from drawn strokes) as it can be expressed with barycentric coordinates in a cell. The $D(t)$ matrix provides trajectories for the identified constraints, hence locally driving lattice motion. In the formulation of Baxter et al. [BBA08], these constraints are handled by Lagrange multipliers. ARAP factorization only needs to be recomputed when constraints are added or removed (through C); whereas trajectory editing (through $D(t)$) works in real time.

In practice, we choose to represent a constrained trajectory with a cubic Bézier curve, which permits the reproduction of arc motions as prescribed in traditional animation guidelines (e.g., [JT95, Wil01]). We have favored this representation over logarithmic spirals [WNS*10] for two reasons: it better captures the motion generated by ARAP interpolation, and it provides intuitive direct controls through tangents. A Bézier curve is initially fit to the discrete trajectory (position and parameterization) of the selected constrained point using Schneider's algorithm [Sch90]. In our system, we always re-parameterize the Bézier curve uniformly along arc-length after any edit. This gives a more accurate local temporal control

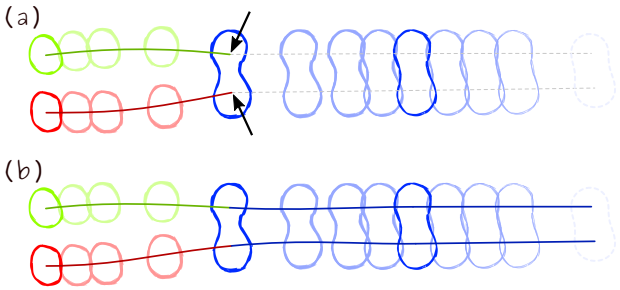


Figure 12: *Constrained trajectories help control inbetweening. In (a), we use one constraint for E_g and one constraint for E_r. Trajectories for the same points in E_b are also visualized with dashed lines. Since each embedding is interpolated separately, we can observe that trajectories are not G^1-continuous (pointed out by arrows). In (b), we activate the same constraint points in E_b and align the tangent of their trajectory starting points with the ones in E_g and E_r, which restores continuity.*

through $D(t) = D(S(\mathrm{f}))$, where f is the current frame and S the spacing function of the embedding. The geometry of the fitted trajectory may then be edited to drive lattice motion explicitly, as illustrated in Figure 12(a) on the red and green embeddings of Figure 5. Since the curve endpoints are fixed, only the tangents of the trajectory may be modified.

Constrained trajectories may be trivially chained across breakdown keyframes since the lattice topology is maintained throughout a same embedding. More interestingly, constrained trajectories may be chained *across different embeddings*. Consider for instance a pair of embeddings E_0 and E_1, and a constrained trajectory $D_0(t)$ that drives the lattice of E_0 from its start to its end keyframe. In the likely case where $D_0(1) \in E_1$ (assuming E_0 and E_1 have been properly matched), a new constrained trajectory $D_1(t)$ may be imposed on E_1, with $D_1(0) = D_0(1)$ by construction. Note that the trajectories remain independent, hence removing E_0 will not affect the chained trajectories in E_1. Any chain of trajectory constraints constructed this way can be made G^1-continuous at keyframes by aligning the tangents of adjacent constrained trajectories. This is a

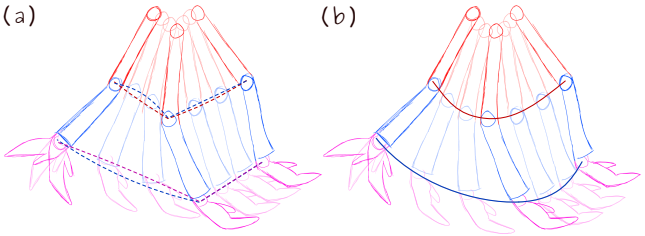


Figure 13: *Hierarchical constraints maintain spatial relationships between adjacent embeddings throughout interpolation. In (a), the arm, forearm and hand embeddings interpenetrate at intermediate frames, as shown by the dashed trajectories. In (b), by placing two hierarchical constraints – one (in red) on the arm driving the forearm, and the other (in blue) on the forearm driving the elbow – the full articulated chain remains properly connected at constrained points, which follow smooth arc trajectories.*

notable feature of our approach, since it provides *persistent* control over trajectories while relying on transient embeddings. We demonstrate that feature in Figure 12(b), where the trajectories of E_g and E_r are chained with G^1 continuity to trajectories of E_b. This produces a visually continuous motion even in the presence of a topological change, as is best seen in the supplemental demo video where we use more inbetween frames.

Trajectories might not only be used to "knit" embeddings across time but also in space. This is a useful feature when two embeddings that are visually connected at contiguous keyframes get disconnected or interpenetrate during interpolation, as shown in Figure 13(a). For a shared constraint to be added, lattices in each embeddings must overlap. One of the embeddings is identified as the leader E_l and the other one as the follower E_f. Only the constrained trajectory of E_l is edited, while E_f follows that same trajectory. This raises an issue when the spacing functions S_l and S_f of the respective embeddings are different, as E_l and E_f are then driven by the same trajectory but at different speeds. The issue is trivially solved by using the spacing function of the leader S_l on the follower E_f, *only* at the constrained embedding point; S_f is retained otherwise. Such hierarchical constraints bear a resemblance to skeletons used in cut-out animation systems, as they open up to the control of articulated structures. In particular, they may be persistent as with any other constrained trajectory, as shown in Figure 13(b). However, they may also be deactivated at any keyframe, which is useful for imposing temporary constraints, such as an object temporarily attached before being thrown away (see the supplemental results video for an example).

Constrained trajectories are easily updated after each of the timeline operations of Figure 4. When an embedding is split at a frame $\mathrm{f}_{\mathrm{split}}$, the lattice positions $V_{\mathrm{split}} = V(S(\mathrm{f}_{\mathrm{split}}))$ are used for the breakdown keyframe and a new ARAP interpolation is recomputed on each new interval, i.e., from V_0 to V_{split} and from V_{split} to V_1. Bézier trajectories are split as well, and the tangents at the split points recomputed, for instance using De Casteljau's algorithm. In all our tests, we have observed that motion before and after splitting are visually identical, even though we could not find a proof of the transitivity of ARAP interpolation. When removing a breakdown keyframe, the associated transformation and trajectories must be removed as well, which potentially leads to a simpler motion. In practice, we keep all active constraints, and fit new trajectories from the default ARAP interpolation on the new interval to preserve artistic inputs as much as possible.

6. Implementation details

Strokes & lattices. In our system, strokes are represented by polylines with stylus pressure recorded at vertices when available. As mentioned in previous sections, embeddings may apply to subsets of strokes, which consist of stroke segments. Moreover, since embeddings are transient, different subsets of strokes may be manipulated at different keyframes, as demonstrated in the last interval of Figure 17. Each new lattice is enforced to be connected and automatically initialized in an axis-aligned manner. We make only one exception: when converting a breakdown keyframe into a new embedding (see Figure 4(a)), we keep the previously transformed lattice to retain its target transformation $T^\star$. This is a small limitation of the current implementation that could be solved by transferring $T^\star$ to a new axis-aligned lattice. We provide an additional tool to add contiguous empty cells inside an embedding, which is particularly useful to fill in lattices (enforcing their rigidity through interpolation), or to provide support for constraints away from strokes. The size of lattice cells may also be adjusted to capture fine details.

Vector registration. The registration problem consists in finding a transformation $T^\star$ that aligns the strokes in an embedding (the source) with a given subset of stroke segments in the next keyframe (the destination). Our solution relies on the image registration technique of Sýkora et al. [SDC09], which works in two phases that are iterated until convergence: a "push" phase that moves lattice corners towards locations where the source and destination are similar; and a "regularize" phase that reintroduces local rigidity.

In our approach, we adapt the push phase to work on vector strokes, while the regularize phase is left unchanged, as illustrated in Figure 14. In our push phase, we compute the optimal rigid transformations that minimizes the sum of squared distances between the stroke points in the source and their nearest-neighbors in the destination, using the closed-form solution of Schaefer et al. [SMW06]. This process results in a set of disconnected transformed cells (step 1 in Figure 14). The lattice connectivity is then restored by averaging lattice corner positions (step 2). The regularize phase is similar, except we compute optimal rigid transformations that aligns the source cells with the lattice obtained at the end of the push phase (step 3), before restoring connectivity (step 4). The regularize phase is repeated N_r times, with greater values of N_r increasing rigidity of the lattice (we use $N_r = 10$ by default). Repetitions are crucial to prevent the lattice from collapsing due to the naive initial nearest-neighbor correspondences.

An alternative solution would have been to use the block matching approach of Sýkora et al. [SDC09] applied to rasterized strokes, or their distance transform, similarly to Noris et al. [NSC*11]. We have found in practice that a vectorial solution is much more efficient since it provides direct initial correspondences. As a result,

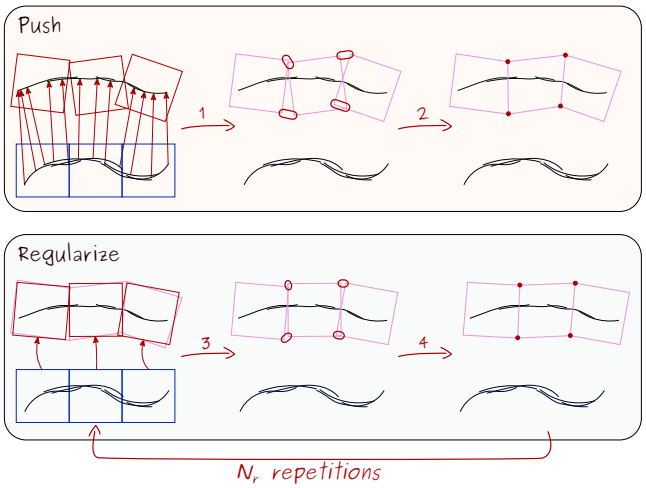


Figure 14: *Our vector registration algorithm matches a source drawing (bottom strokes) with a destination drawing (top strokes) in two phases. During the push phase, lattice cells are (1) independently rigidly transformed to match the closest destination stroke points (red arrows), after which (2) the lattice connectivity is restored through averaging. During the regularize phase (repeated N_r times), the shape of the source lattice is partly restored by (3) finding independent rigid cell transformations that match the current target lattice before (4) restoring connectivity.*

there is no need to iterate over the sequence of push and regularize phases as in the raster version: both phases are only applied once, and our vectorial registration achieves real-time performance. Thanks to its efficiency, vectorial registration may be used interactively in a semi-automatic fashion. For instance, our system allows the user to deform the lattice using the tools mentioned in Section 4.1 and to start the registration from this deformed configuration, hence allowing *plastic deformations* of the lattice. Registration might even be run continuously during the deformation so that the source strokes glide over the destination. Please see the supplemental demo video for a live illustration.

Spacing & trajectories. The analytic spacing function S may be controlled by adjusting the position of each individual tick. For intervals holding many frames, this may be tedious; hence we provide several interface tools to control multiple ticks at once. As shown in the supplemental demo video, we provide typical "ease-in/ease-out" controls, as well as options to place ticks on "halves", as routinely done by 2D artists when creating spacing charts [Wil01]. To facilitate constrained trajectory editing, besides tangents manipulation, we have implemented a sketching technique that linearly transforms a curve drawn by the user such that its endpoints match the constrained positions at keyframes, and then fit a cubic Bézier curve with uniform arc-length parameterization to the result. This is also demonstrated in the supplemental demo video.

Cross-fading. Each interpolated frame is the result of cross-fading forward and backward transformed strokes. To limit ghosting artifacts, we apply cross-fading to control the thickness of forward strokes, using a function $c(t) : [0,1] \to [0,1]$:

$$c(t) = \begin{cases} (1-(2(t-\frac{1}{2})^2)^2 & \text{if } t \in [0,\frac{1}{2}], \\ 1 & \text{otherwise.} \end{cases}$$

For backward strokes we simply use $c(1-t)$. Hence at any time t, there is always one set of strokes (forward or backward) that is displayed at full thickness. Note that cross-fading is also subjected to spacing since $t = S(\mathrm{f})$ for a frame f. By default, we deactivate cross-fading for embeddings whose end keyframe does not store any stroke, so as to keep forward propagated strokes displayed over the full interval. Cross-fading is reactivated whenever the end keyframe is populated with strokes, or the embedding is tagged as fading in or out. Yet, we use linear cross-fading in the latter case to make strokes appear or disappear at the same rate as interpolation.

Matching	**10.22 ms**
↳ Push phase	8.41ms
↳ Regularize phase	1.81ms
Interpolation	**6.67 ms**
↳ Factorization	5.55ms
↳ Solve	0.90ms
Stroke rendering	0.22ms
Total	**16.89 ms**

Table 1: *Performance of our single-threaded implementation in a worst-case scenario, recorded on a i7-4790K 4GHz CPU and a Nvidia GeForce GTX 980Ti GPU. We report timings for the second keyframe of the fish animation (Figure 19), holding* 284 *strokes in a lattice of* 753 *cells. We use $N_r = 10$ for regularization, while solve and stroke rendering timings correspond to* 10 *rendered frames.*

Performance. Our system is implemented in C++, using the *Qt* library for the GUI and an OpenGL Geometry Shader for the final stroke rendering. We use the sparse LU solver of Eigen [GJ*10] to efficiently factorize and solve the linear system involved in ARAP interpolation (Appendix A). Table 1 reports the performance of our implementation recorded for the second keyframe of Figure 19 that uses the largest lattice from all our examples, and relies on the more demanding pose-to-pose workflow with semi-automatic (and thus interactive) registration. Interpolation times are also reported since we display the interpolated results during matching via an "onion skin" visualization. We obtain real-time performance in such a worst-case scenario, as well as in all our experiments.

7. Results

In this section and the supplemental video we present complex results obtained using our animation system. Note that the video does not merely show final inbetweened results, but first and foremost the ability of our system to enable a fast and flexible creative process to get to those results.

Figure 1 shows the particularly complex example of a special effect animation traced from Gilland's book [Gil09], featuring multiple topological changes made possible by the use of several transient embeddings. Figure 13 demonstrates a first **Articulated arm**

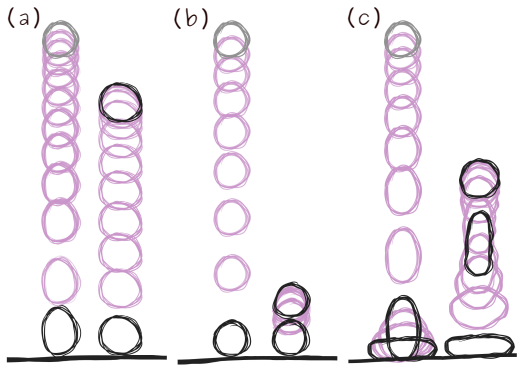


Figure 15: *Starting from the same first extreme key drawing (in gray), three ball drop animations are produced with a different sequence of breakdowns (in black) using a shift-and-trace workflow. In (a), the ball is slightly deformed in the direction of motion and acceleration is conveyed through an ease-in/ease-out spacing, with a high rebound conveying a light object. In (b), the ball is accelerated, with no deformation on contact and a low rebound, all effects conveying a heavier object. In (c), we apply a large "squash-and-stretch" deformation to the ball before and after it hits the ground, giving it an elastic and cartoony look-and-feel.*

animation exercise taken from Williams' guide [Wil01], showcasing arc motions controlled through hierarchical trajectories. In Figure 15 through 18, we reproduce three other classic animation exercises. Details are provided in figure captions; we summarize the main demonstrated features below:

Ball drop (Figure 15): different impressions of weight are conveyed through variations in breakdowns and spacing;

Walk cycle (Figure 16): breakdown keyframes are used to produce drastically different animations, starting from the same pair of extreme key drawings;

Flour sack jump (Figure 17): plausible motion dynamics are obtained by adding anticipation and follow-through via a combination of breakdowns, deformations and spacing;

Head turn (Figure 18): 3D-like rotation is conveyed through several embeddings, some being tagged as fading in or out.

Note that even a simple ball drop animation (Figure 15) would require many trials-and-errors for novice 2D animators to produce a first result. Exploring alternate results afterwards would require a significant amount of work (tens of minutes, perhaps hours) since all inbetween frames must be redrawn. In contrast, our system allows the exploration of different alternatives instantly while retaining a plausible rough hand-drawn animation style.

The Head turn example (Figure 18) reveals the main limitation of our approach (see Section 8.3): even with a special treatment of the occluded embeddings (such as the ears), our system is not yet designed to let *parts of* embeddings appear or disappear behind other embeddings, as it would require to interpolate topological changes. The method of Zhu et al. [ZPBK17] is able to produce such inbetween frames, but the user must specify correspondences for cuts, openings and boundaries on every key drawing, and compatible embeddings must then be computed throughout the animation with a prohibitively expensive optimization.

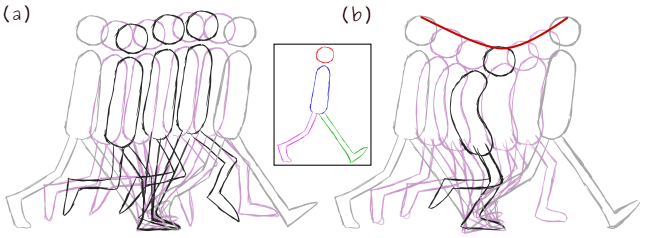


Figure 16: *Starting from the same two extreme key drawings (in gray) traced over an animation exercise by Williams [Wil01] and using the same decomposition into four embeddings throughout (see inset image), we produce (a) a "normal" walk cycle by adding three breakdowns (in black) at, and around, the passing position, and (b) a very different animation with a single breakdown keyframe and a constrained trajectory (red curve) to make the character bow down inbetween steps.*

8. Discussion

Our animation system relies on the concept of transient embeddings. In Section 8.1, we justify this choice, comparing it to an alternative stroke-level animation system. We then discuss practical limitations and future work in Sections 8.2 and 8.3.

8.1. Comparison with stroke-level inbetweening

Xing et al. [XWSY15] present an interactive 2D animation system that aims at assisting in both the drawing of new keyframes and the matching of drawings across keyframes. In their approach, the user provides guiding strokes that are used by the system to predict a new drawing based on past spatial and temporal repetitions. The user may then either directly reuse the suggested drawing or instead rely on guide strokes, either case yielding strokes matched between the current and next keyframes. In effect, this amounts to a different kind of workflow that works at the stroke level and couples drawing with matching, alternating between shift and trace steps.

Since strokes are only matched from one keyframe to the next, the representation may be considered transient, like ours. However, unlike our method, the embedded deformation model [SSP07] is carried by the strokes themselves, at a coarse sampling rate. To generate inbetween frames, the parameters of this model – an affine transformation per sample – are interpolated in time, and the full-resolution strokes are reconstructed by spatial diffusion.

We reproduce one of their animations with our system in Figure 19(a), achieving a very similar result using a pose-to-pose workflow. The comparison is not intended to show the superiority of one workflow over another. Indeed, we believe that their auto-completion algorithm – which is the core contribution of their work – could be adapted to compute lattice transformations in our system as well, providing an additional matching solution to artists. Instead, we want to stress the implications of choosing to work at the stroke level. First, coupling matching with drawing of strokes has the undesired property that when strokes are erased, matching is lost in the process. This is obviously not the case with our system, since matching is done on embedding lattices. Second, it is

Figure 17: *For this flour sack jump animation, we start from three initial keyframes (in black and red/blue), which are registered in a pose-to-pose approach using a single embedding for the whole drawing. We then add a breakdown (in dark gray) to produce an anticipation effect, then two additional breakdowns (in light gray) to refine animation. For follow-through, the last key drawing is divided into two embeddings: the red one is held fixed on screen while the blue one is slightly deformed through a shift step.*

unclear how the diffusion-based interpolation could be adapted to provide control over trajectories and motion smoothness, whereas such control is direct in our system as demonstrated in Figure 19(b) and the supplemental results video.

8.2. Practical limitations

Our implementation is intended as a proof-of-concept prototype demonstrating the potential of transient embeddings. Yet our current animation system could be improved in several ways.

On the matching side, we could improve the registration tool with an automatic global non-rigid registration prior to our more local ARAP registration; or we could give artists the option to pin some correspondences to guide that global alignment process. Such pinned points should be related to interpolation constraints (any constraint should be a pin), but they should not be equivalent since artists may not need to control the trajectory of all pinned points. As mentioned in Section 8.1, we would also like to adapt stroke auto-completion [XWSY15] to guide the matching of embeddings. Working with lattices may allow us to alleviate the influence of drawing order on prediction, which may prove especially problematic for rough drawings.

On the interpolation side, we have identified three directions of improvements. First, we would like to explicitly model the rigid component of a transform T through a *pivot transform*, such that the non-rigid component may be handled by ARAP interpolation relative to that pivot. This would simplify the control of a whole embedding motion through the trajectory of its pivot (e.g., a bouncing ball moving along an arc), which could exhibit interesting default behaviors (e.g., logarithmic spirals [WNS*10]). Second, our trajectory constraints only allow to enforce G^1 continuity across keyframes due to potential discontinuities of the spacing functions S. We would like to investigate ways to ensure full C^1 continuity by giving the option to adjust S on both sides of abutting constraints. Finally, we have focused on hard constraints for the control of trajectories; but other solutions could be devised. Soft constraints could be introduced [BBA08] to mimic exterior forces (such as wind); or constraints may be allowed to slide inside an embedding to precisely control motion at articulations. This latter improvement should be more computationally-demanding as it would a priori require a new ARAP factorization at each interpolated frame.

8.3. Future work

We have chosen to rely on ARAP interpolation to implement transformations between embeddings, as it produces natural results by default, and may be further controlled efficiently through constrained trajectories. However, it does not appear to be the most suitable solution in some cases. For instance, ARAP interpolation is not well adapted to animations that are expected to follow an underlying path, such as roots growing in a soil as shown in the supplemental results video. More generally, 3D-like motions (e.g., rotations out of the canvas plane) remain hard to reproduce with our approach. A solution might be again to rely on an underlying guide during interpolation, this time a 3D proxy surface.

3D-like animations raise another, more difficult problem: the handling of occlusions. As shown in Figure 18, occluded embeddings simply fade in or out during interpolation in our current system. Even though cross-fading is enough in most cases, it is inadequate in the case of occlusions: parts of strokes should be revealed or hidden during interpolation, which requires the handling of depth order relationships between embeddings and the interpolation of topological changes at interactive rates with minimal user intervention. Extending our system to handle occlusions while retaining non-linear editing abilities is a challenging and exciting direction of future work.

Acknowledgments

We are grateful to Antoine Antin, Clément Berthaud, Eric Scholl and Fabrice Debarge from Praxinos for their participation to the observational study and for very helpful discussions. This work is supported by the ANR MoStyle project (ANR-20-CE33-0002).

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), ACM, p. 157–164. doi:10.1145/344779.344859. 3, 7, 15

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

[Ado] ADOBE INC.: Adobe animate. URL: https://www.adobe.com/products/animate.html. 2, 4, 7

[Bae69] BAECKER R. M.: Picture-driven animation. In *Proceedings of the May 14-16, 1969, Spring Joint Computer Conference* (1969), ACM, pp. 273–288. doi:10.1145/1476793.1476838. 2

[BBA08] BAXTER W., BARLA P., ANJYO K.-I.: Rigid shape interpolation using normal equations. In *Proceedings of the International Symposium on Non-photorealistic Animation and Rendering* (2008), ACM, pp. 59–64. doi:10.1145/1377980.1377993. 3, 7, 8, 12, 15

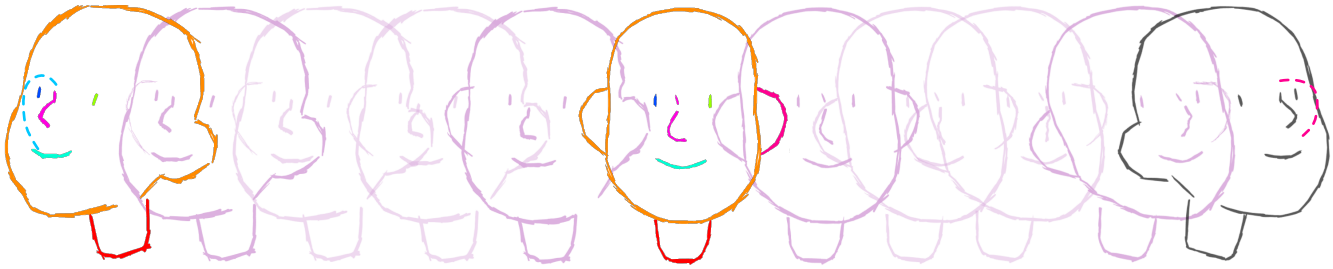


Figure 18: *This head turn animation is generated from only three key drawings whose embeddings are color-coded – the last keyframe is made of a single embedding (in black). To make the ears appear or disappear, we must draw and/or deform them occluded (blue and magenta dashed lines) and tag them for fading in or out.*

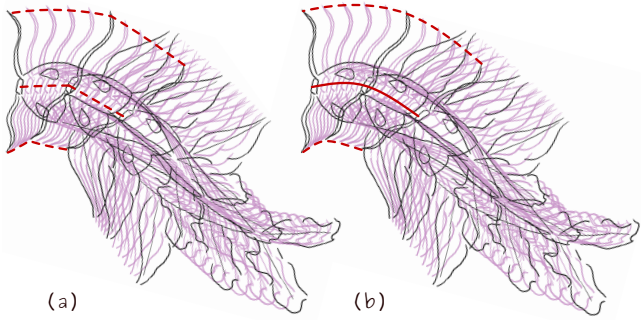


Figure 19: *We traced three keyframes (in black) from the fish example in the supplemental video of [XWSY15], and matched them using a pose-to-pose workflow. In (a), we show the interpolated drawings produced by default by our method: as highlighted by the dashed line, the trajectories are discontinuous at the middle keyframe. In (b), we constrained the trajectory depicted in solid red to follow a smooth arc. Notice how this constraint is propagated to neighbor trajectories (e.g., the whiskers, see the dashed red curves).*

[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. 2, 3

[BKLP16] BAI Y., KAUFMAN D. M., LIU C. K., POPOVIĆ J.: Artist-directed dynamics for 2d animation. *ACM Trans. Graph. 35*, 4 (2016). doi:10.1145/2897824.2925884. 3

[BW71] BURTNYK N., WEIN M.: Computer-generated key-frame animation. *Journal of the SMPTE 80*, 3 (1971), 149–153. 2

[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, 3

[CAC] CACANI PTE LTD.: Cacani. URL: https://cacani.sg/. 2, 4, 7

[Cat78] CATMULL E.: The problems of computer-assisted animation. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques* (1978), vol. 12, ACM, pp. 348–353. doi:10.1145/800248.807414. 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

[Com] COMMUNITY B. O.: Blender - a 3d modelling and rendering package. URL: http://www.blender.org. 2

[CPL21] CASEY E., PÉREZ V., LI Z.: The animation transformer: Visual correspondence via segment matching. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (2021), pp. 11303–11312. doi:10.1109/ICCV48922.2021.01113. 3

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

[DFSEVR01] DI FIORE F., SCHAEKEN P., ELENS K., VAN REETH F.: Automatic in-betweening in computer assisted animation by exploiting 2.5d modelling techniques. In *Proceedings Computer Animation 2001. Fourteenth Conference on Computer Animation* (2001), pp. 192–200. doi:10.1109/CA.2001.982393. 2

[dJB06] DE JUAN C. N., BODENHEIMER B.: Re-using traditional animation: Methods for semi-automatic segmentation and inbetweening. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2006), Eurographics Association, pp. 223–232. 2

[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). 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

[Dur91] DURAND C. X.: The "toon" project: Requirements for a computerized 2d animation system. *Computers & Graphics 15*, 2 (1991), 285–293. doi:https://doi.org/10.1016/0097-8493(91)90081-R. 2

[FBC*95] FEKETE J.-D., BIZOUARN É., COURNARIE É., GALAS T., TAILLEFER F.: Tictactoon: A paperless system for professional 2d animation. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), pp. 79–90. doi:10.1145/218380.218417. 2

[FC80] FRITSCH F. N., CARLSON R. E.: Monotone piecewise cubic interpolation. *SIAM Journal on Numerical Analysis 17*, 2 (1980), 238–246. doi:10.1137/0717021. 7

[FTA05] FU H., TAI C.-L., AU O. K.-C.: Morphing with laplacian coordinates and spatial-temporal texture. In *Proceedings of Pacific Graphics* (2005), pp. 100–102. 3

[FZP*20] FISH N., ZHANG R., PERRY L., COHEN-OR D., SHECHTMAN E., BARNES C.: Image morphing with perceptual constraints and stn alignment. *Computer Graphics Forum 39*, 6 (2020), 303–313. doi:10.1111/cgf.14027. 3

[Gil09] GILLAND J.: *Elemental Magic, Volume I: The Art of Special Effects Animation*. Focal Press, 2009. 1, 10

[GJ*10] GUENNEBAUD G., JACOB B., ET AL.: Eigen v3. http://eigen.tuxfamily.org, 2010. 10

[IMH05] IGARASHI T., MOSCOVICH T., HUGHES J. F.: As-rigid-as-possible shape manipulation. *ACM Trans. Graph. 24*, 3 (2005), 1134–1141. doi:10.1145/1073204.1073323. 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

[JT95] JOHNSTON O., THOMAS F.: *The illusion of life : Disney animation*. Disney Press, 1995. 1, 2, 3, 8

[KHS*12] KAJI S., HIROSE S., SAKATA S., MIZOGUCHI Y., ANJYO K.: Mathematical Analysis on Affine Maps for 2D Shape Interpolation. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation* (2012). doi:10.2312/SCA/SCA12/071-076. 3, 15

[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, 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

[LMY*13] LIU X., MAO X., YANG X., ZHANG L., WONG T.-T.: Stereoscopizing cel animations. *ACM Trans. Graph. 32*, 6 (2013). doi:10.1145/2508363.2508396. 2

[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

[MIT67] MIURA T., IWATA J., TSUDA J.: An application of hybrid curve generation: cartoon animation by electronic computers. In *AFIPS '67 (Spring)* (1967). 2

[MSG96] MADEIRA J. S., STORK A., GROSS M. H.: An approach to computer-supported cartooning. *The Visual Computer 12*, 1 (1996), 1–17. doi:10.1007/BF01782215. 2

[NHA19] NARITA R., HIRAKAWA K., AIZAWA K.: Optical flow based line drawing frame interpolation using distance transform to support inbetweenings. In *2019 IEEE International Conference on Image Processing* (2019), pp. 4200–4204. 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. 3, 9

[PSN20] PARK S., SEO K., NOH J.: Neural crossbreed: Neural based image metamorphosis. *ACM Trans. Graph. 39*, 6 (2020). doi:10.1145/3414685.3417797. 3

[Qui10] QUILEZ I.: Inverse bilinear interpolation, 2010. URL: https://iquilezles.org/articles/ibilinear/. 15

[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, 3

[RID10] RIVERS A., IGARASHI T., DURAND F.: 2.5d cartoon models. *ACM Trans. Graph. 29*, 4 (2010). doi:10.1145/1778765.1778796. 2

[SBv05] SÝKORA D., BURIÁNEK J., ŽÁRA J.: Colorization of black-and-white cartoons. *Image and Vision Computing 23*, 9 (2005), 767–782. doi:10.1016/j.imavis.2005.05.010. 2

[Sch90] SCHNEIDER P. J.: *An Algorithm for Automatically Fitting Digitized Curves*. Academic Press Professional, Inc., 1990, p. 612–626. 8

[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. 3, 7, 9

[SGWM93] SEDERBERG T. W., GAO P., WANG G., MU H.: 2-d shape blending: An intrinsic solution to the vertex path problem. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (1993), ACM, p. 15–18. doi:10.1145/166117.166118. 3

[SMW06] SCHAEFER S., MCPHAIL T., WARREN J.: Image deformation using moving least squares. *ACM Trans. Graph. 25* (2006), 533–540. doi:10.1145/1179352.1141920. 9

[SSP07] SUMNER R. W., SCHMID J., PAULY M.: Embedded deformation for shape manipulation. *ACM Trans. Graph. 26*, 3 (2007). doi:10.1145/1276377.1276478. 3, 11

[SZGP05] SUMNER R. W., ZWICKER M., GOTSMAN C., POPOVIĆ J.: Mesh-based inverse kinematics. *ACM Trans. Graph. 24*, 3 (2005), 488–495. doi:10.1145/1073204.1073218. 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

[Too] TOON BOOM ANIMATION INC.: Toon boom harmony. URL: https://cacani.sg/. 2, 4, 7

[TVP] TVPAINT DEVELOPPEMENT: Tvpaint. URL: https://www.tvpaint.com. 4, 6

[Wil01] WILLIAMS R.: *The animator's survival kit*. Faber and Faber, 2001. 1, 2, 3, 4, 7, 8, 10, 11

[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, 3, 8, 12

[WXXC08] WANG Y., XU K., XIONG Y., CHENG Z.-Q.: 2d shape deformation based on rigid square matching. *Computer Animation and Virtual Worlds 19*, 3-4 (2008), 411–420. doi:10.1002/cav.251. 3

[Xie95] XIE M.: Feature matching and affine transformation for 2d cell animation. *The Visual Computer 11*, 8 (1995), 419–428. doi:10.1007/BF02464332. 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. 3, 11, 12, 13

[XZWB05] XU D., ZHANG H., WANG Q., BAO H.: Poisson shape interpolation. In *Proceedings of the 2005 ACM symposium on Solid and physical modeling* (2005), ACM, pp. 267–274. doi:10.1145/1060244.1060274. 3, 15

[Yag17] YAGI Y.: A filter based approach for inbetweening. *CoRR abs/1706.03497* (2017). 3

[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, 3

[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

[YHY19] YANG W.-W., HUA J., YAO K.-Y.: Cr-morph: Controllable rigid morphing for 2d animation. *Journal of Computer Science and Technology 34*, 5 (2019), 1109–1122. doi:10.1007/s11390-019-1963-3. 3

[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. doi:10.1111/cgf.13518. 2

[YVG20] YAN C., VANDERHAEGHE D., GINGOLD Y.: A benchmark for rough sketch cleanup. *ACM Trans. Graph. 39*, 6 (2020). doi:10.1145/3414685.3417784. 2

[ZHF12] ZHANG L., HUANG H., FU H.: Excol: An extract-and-complete layering approach to cartoon animation reusing. *IEEE Transactions on Visualization and Computer Graphics 18*, 7 (2012), 1156–1169. doi:10.1109/TVCG.2011.111. 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, 11

Appendix A: Controllable ARAP interpolation

The original ARAP formulations of Alexa et al. [ACOL00] and Xu et al. [XZWB05] do not offer any control over motion trajectories. Baxter et al. [BBA08] introduce such controls through linear constraints thanks to their reformulation of the problem in terms of normal equations. Katji et al. [KHS*12] present an even more generic mathematical framework, but since we do not need such a generalization, we choose the method of Baxter et al. [BBA08] whose implementation is simpler and very efficient.

More precisely, to compute the interpolated positions of the lattice corners at time $t \in [0,1]$, we use Equation 5 in their paper:

$$V(t) = \begin{bmatrix} P^\top W P & C \\ C & 0 \end{bmatrix}^{-1} \begin{bmatrix} P^\top W A(t) \\ D(t) \end{bmatrix},$$

where the sparse matrix P encodes the triangulated lattice connectivity, the matrix $A(t)$ stores the target affine triangle deformations, the diagonal matrix W allows to specify a weight per triangle (we use its area), the matrix C expresses hard linear constraints defined on lattice vertices, and the matrix $D(t)$ stores the constrained driven positions. When no constraint is provided by the user, we compel the mean position of the lattice to follow a linear and uniform trajectory in order to have a unique solution. This implies setting $C = [1/N \ \dots \ 1/N]$, with N the number of lattice triangles, and set $D(t)$ to the linearly interpolated position of the lattice barycenter. Otherwise, each constrained point p maps to a row C_p in C with four non-zero values, one for each barycentric coordinate relative to its cell corners. Denoting $\{i,j,k,l\}$ the indices of these corners, enumerated in clockwise order starting from top-left, yields:

$$\begin{array}{cccccccc} & & i & & j & & k & & l & \\ C_p = & \big[\dots & (1-u)(1-v) & \dots & u(1-v) & \dots & uv & \dots & (1-u)v & \dots\big], \end{array}$$

with (u,v) the coordinates of p in the quad cell obtained by inverse bilinear interpolation [Qui10]. The corresponding row in $D(t)$ stores the position along the trajectory curve evaluated at t.

Note that matrix inversion, which is the most computationally expensive part of the method, needs to be performed in only two cases: when a lattice is created or its topology changed, since P is modified; or when a linear constraint is added to C. The matrix $A(t)$ is updated whenever matching is modified, whereas $D(t)$ is updated whenever a constrained trajectory is edited.

With this formulation, the inverse lattice transformation $T(t)^{-1} = V(1-t) - V_1$ might result in a non-symmetric behavior, which is obviously problematic for strokes cross-fading. We thus use the symmetric formulation of Baxter et al. [BBA08] which is slightly more complex but equally fast to compute. We refer the interested reader to their paper for details.