Computers & Graphics 65 (2017) 31–44

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

DiLight: Digital light table – Inbetweening for 2D animations using guidelines[☆]

Leonardo Carvalho[a,*], Ricardo Marroquim[a], Emilio Vital Brazil[b]

[a] Computer Graphics Lab, Systems Engineering and Computer Science Department, Federal University of Rio de Janeiro, Brazil
[b] IBM Research, Rio de Janeiro, Brazil

ARTICLE INFO

Article history:
Received 26 October 2016
Revised 31 March 2017
Accepted 5 April 2017
Available online 28 April 2017

Keywords:
2D animation
Inbetweening

ABSTRACT

In traditional 2D animation several intermediate frames are drawn between two consecutive key frames. This process can be very time-consuming and tedious for the animator. In this paper we present a semi-automatic inbetweening method that is tightly coupled with the animation production pipeline. We exploit the fact that artists typically start with an outline of the drawing to help preview the illustration, and use these guidelines to improve the curve correspondence inference. The method is based on a two-level matching algorithm, where the first one finds correspondences between the guidelines, and the second one between the final art. This separation further aids the artist by using transformed guidelines and drawings from preceding frames to guide the creation of new ones, acting as a digital light table. We show the robustness of our method with a variety of animation examples, including sequences from animation books and professional animators.

1. Introduction

When an artist is drawing, he usually starts with an outline that approximately describes the shapes with guidelines, with as few details as possible. These guidelines help the artist in previewing the final art, and in avoiding asymmetries in the result. Moreover in cartoon animations, a great amount of drawings (frames) are necessary to create the illusion of movement when presented in sequence. Since the difference between two successive frames is usually small, it is common to start with a larger separation that gives a hint about the overall movements being depicted, these frames are called *key frames*. Posteriorly, the intermediate frames are drawn, in a process known as *inbetweening*, which is usually accomplished manually, requiring a great effort [1]. There are some methods used to preview the animation during its creation. A common practice is to draw the frames on onionskin papers, and increase its translucence using a light table, such that previous frames are visible during the drawing of the current frame. In computer aided animation systems this mechanism is usually simulated by displaying some frames in a sequence with varying opacity. Nevertheless, this does not reduce the burden of drawing all inbetween frames.

To aggravate the issue, with the development of different media devices, the frame rate is increasing, and some expect that it will achieve up to 120 frames per second in future standards [2]. This shift leads to smoother animations at the cost of requiring an even greater amount of frames. Even though this effort may be dramatically reduced with the aid of computational methods, it is not a trivial task.

Almost three decades ago Catmull [3] already discussed the problems of computer-assisted animation, stating that one of the greatest challenges in this area is automatic inbetweening. This is due to the fact that cartoons represent three-dimensional characters projected into a two-dimensional space, implying in loss of information. To achieve an automatic solution, a model describing the characters as imagined by the animators must be known. Catmull listed some approaches that may be used to solve this issue: infer missing information from the drawings; manual specification of the missing data; break the characters into layers; use of skeletons; 3D outlines; and restricting the animation. The challenges and approaches discussed by Catmull are still valid today. We follow more than one of these guidelines in our work. We use layer division, and an approach similar to the use of skeletons, but with more flexibility, allowing the animation of non-rigid elements.

Most computer-aided inbetweening solutions require the specification of correspondence between curves from pairs of key frames. Ideally the correspondence is one-to-one, but in practice there are hidden lines, or cases where one line breaks into two or more from one key frame to the next. Finding them automatically

[☆] This article was recommended for publication by Prof P Poulin.
[*] Corresponding author.
E-mail address: leocarvalho84@gmail.com (L. Carvalho).

http://dx.doi.org/10.1016/j.cag.2017.04.001
0097-8493/© 2017 Elsevier Ltd. All rights reserved.

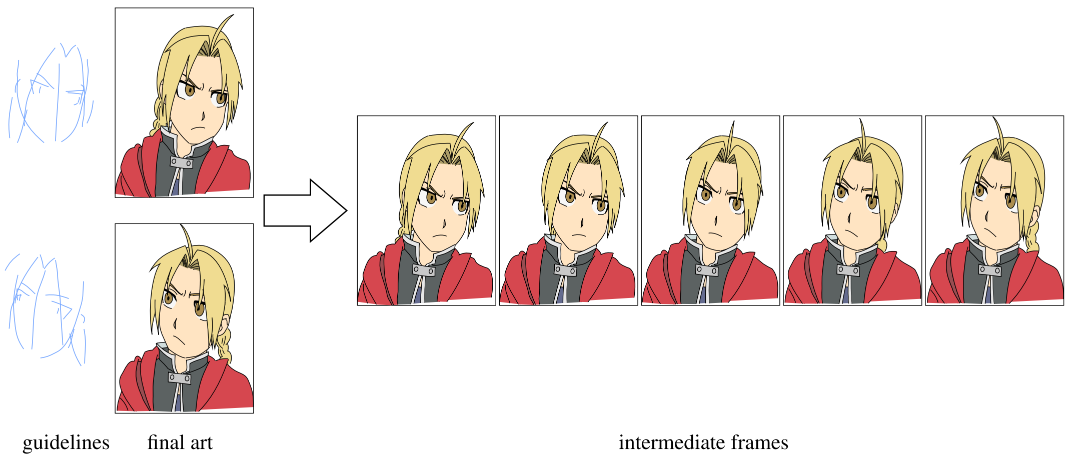


Fig. 1. Overview of the proposed method, where 5 new frames are generated between the two key frames. The input is given by guidelines and final art for the key frames. These guidelines are commonly used by the animators to aid the drawing. Our method also use them to guide a transformation from one key frame to the next. This transformation helps the matching and interpolation of the shapes in the final arts.

and precisely is a crucial step for inbetweening methods. This is the main problem we tackle in this work. Once established, each pair of corresponding curves may be interpolated to generate the intermediate frames.

We present a system to assist in the process of creating a classical animation. Fig. 1 illustrates an example of the input and output of our method. Once the artist finishes the initial guidelines of a new frame, the system finds correspondences with the guidelines of the preceding frame. The advantage of this early matching approach is two-fold: first, it allows warping the preceding final frame to guide the drawing of the new frame's final art, serving as an improved digital light table; second, it greatly improves the matching algorithm of the final art strokes, allowing for larger movements and distortions between frames. After computing the final correspondences, any number of intermediate frames can be generated by interpolating the strokes. The artist can also create layers to better organize the drawings and further aid the inference of intermediate frames. The goal of our method is not to entirely substitute a human animator, but to reduce repetitive tasks. The artist can still make adjustments and corrections to fine-tune the results.

As aforementioned animators usually draw guidelines prior to the final drawings that compose animations. Our main contribution is a method that exploits this fact to:

- give the animator visual hints of how to draw the next key-frame;
- automatically generate intermediate frames;
- allow for large displacements;
- provide control over occlusion.

2. Related work

Many works target automatic (or semi-automatic) inbetweening of 2D cartoons. The first appeared more than 40 years ago [4,5]. In this section we discuss those most closely related to our approach.

Some authors studied the problems involved in 2D animation systems. For example Durand [6] discussed some requirements for a computer aided 2D animation system, where the preferred approach among those cited by Catmull was the restriction of the class of animations that may be drawn. Correspondences are manually defined, and then interpolated using splines in variable time increments. Patterson and Willis [7] made a new analysis of computer aided animation, where the inbetweening problem is divided into two sub-problems: how the silhouettes change, and how parts of the drawing occlude one another. Their approach uses a hierarchy display model, we use a similar hierarchical approach.

Some work uses auxiliary structures to aid inbetweening, similar to what we do with guidelines. Burtnyk and Wein [8] use skeletons to guide the movement of final art shapes during inbetweening. The strokes are described according to coordinate systems defined by the skeletons, which consist of line segments. The results are limited to simple animations, however. Reeves [9] presents an inbetweening method improving the control over interpolation. The animator specifies the key frames and a set of moving points constraints, consisting of curves in space and time, and controlling the trajectory and dynamics of certain points in the key frames. Although it provides a good animation control, the method requires a lot of manual work. Fiore et al. [10] developed a semi automatic inbetweening method based on modelling and 2.5D animation techniques, handling problems such as self-occlusion and silhouette changes. They used 3D skeletons and a hierarchical structure to guide the animation of characters with many parts. However, the user needs to handle control points used in the curve representation, manually define corresponding curves, and specify the skeleton's influence regions. We also use a hierarchical structure to work with complex objects and characters, but in our case the skeleton is replaced by the guideline information and we avoid almost completely manual correspondences.

Another popular direction for 2D animation, or deformation, is as-rigid-as-possible (ARAP) methods. Igarashi et al. [11] introduced a method to deform 2D shapes using some pre-defined vertices as hard constraints. They are, however, limited to shapes that have a simple closed boundary. Later, Sýkora et al. [12] proposed a method without hard constraints to deform 2D shapes. They followed a two step approach, where the first pushes the vertices to new positions, and the second regularizes the locations to better preserve the shape. Even though these methods allow for large deformations and try to preserve the shape as best as possible, they cannot handle well non planar deformations, and non-rigid transformations are only barely handled, thus limiting the types of possible animations.

The inbetweening problem requires the solution of sub-problems, like finding correspondences and morphing shapes. There are many works that focus on a specific sub-problem. Sederberg and Greenwood [13] study how to smoothly transform two polygonal two-dimensional shapes, using a physically based method, where figures are bended as if they are made of wires, with the minimum amount of work. An automatic matching algorithm is described by Song et al. [14], using the relation between feature points in the input curves as a parameter for the matching. In this algorithm cartoon matching is formulated as a many-to-many labeling problem, and the correspondence is constructed using a local shape descriptor. In the work of Xie [15] affine transformations are used to find corresponding shapes between two-dimensional figures, and generate intermediate frames. This method, however, is not capable of handling cases where there is a large distortion between frames, and it does not handle occlusions. Bregler et al. [16] present a technique where the movement of figures in cartoons is tracked and transfered to three-dimensional models, two-dimensional drawings and photographies. Non-rigid movements are tracked using a combination of affine and inbetweening of key-shapes. Melikhov et al. [17] present a system to interpolate key frames, preserving the artistic style of hand-made drawings, with smooth and natural movements. Skeletons are used to morph the texture around strokes. Each frame is represented by a graph, which is used to find matchings between figures. de Juan and Bodenheimer [18] describe two semi-automatic techniques that allow the re-use of traditional animation. Cartoon images are segmented from their backgrounds, and inbetween contours are calculated using implicit surfaces and a non-rigid elastic registration algorithm.

Noris et al. [19] present a method to generate temporal noise-free inbetweens from sketchy animations by combining motion extraction and inbetweening techniques, while preserving the sketchy style of the input frames. Their input is a complete animation, from which some key frames are selected. Motion is then extracted and applied to frames, which are matched and interpolated. This is similar to our work, but they focus on sketchy animations, while aim at creating animations with clean drawings assisted by guidelines (more on this discussion in Section 7.2). In a similar fashion, Ben-Zvi et al. [20] proposed a method to create line-drawn video stylization. Their method predicts stroke matching, merging and splitting. In a video, the frames are usually very close in time, but suffer abrupt topological changes. Their method tries to deal with this noisy behavior of extracted curves, and track them along many frames. In our case, our input does not contain this noisy behavior, but frames are further apart in time and curves can undergo large deformations from one keyframe to the next. Yu et al. [21] focused on finding corresponding objects in two-dimensional animations using local shape descriptors to help matching similar objects in two frames.

A recent work focus on topological aspects of vector graphics. Dalstein et al. [22] created a data structure to control vector graphics animation with time-varying topology, which allows changes like merging, splitting and appearing/disappearing using topological information from the frames. These features may be edited in both space and time. Drawings are represented using a representation from their previous work, Vector Graphics Complexes [23]. We use a similar structure in this work to create curves and regions.

Other authors developed complete systems for the creation of cartoon animations. TicTacToon [24] is a system for professional 2D animations. Drawings are made directly within the system, and may be automatically inbetweened. Unfortunately, the authors themselves state that their automatic process is as slow as drawing the intermediate frames by hand. Reeth [25] developed a 2 1/2D animation system called RUFIAS, with several methods to create animations. This system includes a drawing method using skeletons, that may be used in animations. The animations are controlled by several parameters defined by the animators using this system. Kort [26] presented an algorithm for computer aided inbetweening and its integration in a pen-based graphical user interface. The correspondences are inferred from a cost function based on the relation between the curves. Their algorithm is embedded in a user interface, so that artists may control the results. More recently, Whited et al. [27] presented an interactive tool for tight inbetweening, where a set of user-guided semi-automatic techniques was developed. The authors also present a novel technique for stroke interpolation, where stroke motion is constructed from logarithmic spiral vertex trajectories to achieve more natural trajectories. As stated by the authors, however, in tight inbetweening the keyframes are very similar, usually having the same topology and strokes that are very close in shape, and when this is not the case their method requires some user intervention to guide the matching. In our case, we are not limited to tight inbetweening, thus the keyframes are usually much less similar and farther apart.

Recently Xing et al. [28] developed a system to auto-complete hand-drawn animations. It is related to our work, as it uses information from previous frames to help the artist on the drawing of current frame. Our system is complementary since we are more interested in smoothly interpolating the key frames, while Xing et al.'s method focus on aiding the artist on drawing the key frames. We further position our work in regards to this related work in Section 7.2.

3. 2D curve representation

Even though there is more than one possible representation for 2d curves, we opted to use vector graphics, a popular choice among many other animation methods. Frisken [29] describes an algorithm where a parametric curve is generated on-the-fly from a sequence of digitized points. We use this algorithm to transform the user input into a set of curves, where each curve c is represented as a list of n_c segments, described by $3n_c + 1$ control points $c[0], c[1], \dots, c[3n_c]$. The ith segment is a cubic Bézier curve denoted by B_i defined by the control points $c[3i], c[3i+1], c[3i+2], c[3i+3]$. Several graphics creation tools and file formats use this kind of curve, such that it is straightforward to import/export curves from our system to other tools.

Each segment is parametrized from the domain [0, 1] using the definition of cubic Bézier curves. We can parametrize the complete curve by dividing the domain [0, 1] in n_c sub-intervals, which are mapped to each segment. One simple way of doing this division is by selecting a sequence $0 = t_0 < t_1 < \cdots < t_{n_c} = 1$, and mapping the interval $I_i = [t_i, t_{i+1}]$ to segment B_i, such that $c(t) = B_i((t - t_i)/(t_{i+1} - t_i))$, where $t \in I_i$. One convenient way to define t_i is by making I_i proportional to the arc length.

We are interested in the calculation of the distance between two curves c_i and c_j. The method used in this work is described next. First, define the functions dfw and dbw, that represent distances between curves in different orientations:

$$\mathrm{dfw}(c_i, c_j) = \sum_{k\in I} d(p_i[k] - \bar{p}_i, p_j[k] - \bar{p}_j),$$

$$\mathrm{dbw}(c_i, c_j) = \sum_{k\in I} d(p_i[k] - \bar{p}_i, p_j[n-1-k] - \bar{p}_j),$$

where each curve is uniformly sampled using n points, with $n = 5 \cdot \max(n_{c_i}, n_{c_j})$ (which gives a reasonable approximation of the shape of the curves, as 5 points usually describe the shape of a cubic segment fairly well), $I = [0, \dots, n-1]$, $p_i[k] = c_i(k/(n-1))$, $\bar{p}_i = \frac{1}{n}\sum_{k\in[0,\dots,n-1]} p_i[k]$ (similarly for curve c_j), and $d(u, v)$ is the Euclidean distance between points u and v. These functions are used to check which orientation is better suited when interpolating c_i and c_j. After calculating these functions, we define the distance as:

$$d(c_i, c_j) = \begin{cases} \frac{1}{n}\sum_{k\in I} d(p_i[k], p_j[k]) & \text{if } fw < bw, \\ \frac{1}{n}\sum_{k\in I} d(p_i[k], p_j[n-1-k]) & \text{otherwise,} \end{cases} \quad (1)$$

where $fw = \mathrm{dfw}(c_i, c_j)$ and $bw = \mathrm{dbw}(c_i, c_j)$.

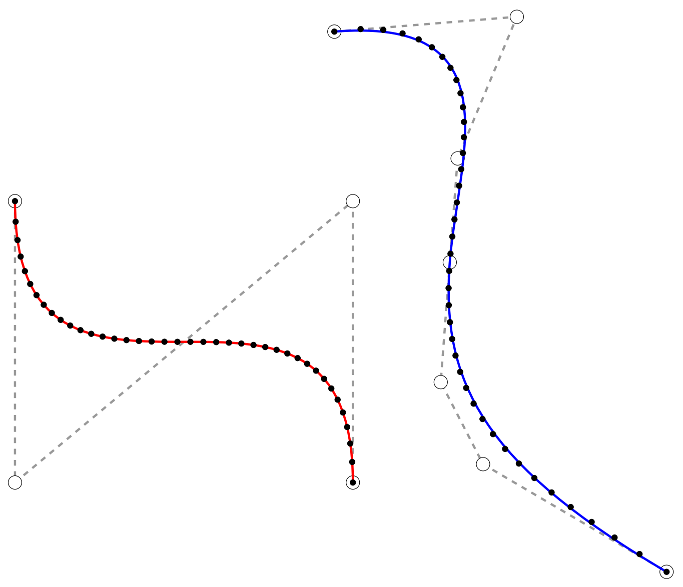


Fig. 2. Applying non-linear transformation T to the curve on the left (red) resulting on the curve on the right (blue). White dots and dashed lines show the control points and control polygon, respectively, for each curve. We apply T on samples of the red curve (black dots), the resulting points are close to the blue curve. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

We also tried to use other distance functions, such as Hausdorff distance, but we achieved better results using Eq. (1), mismatching of curves was more frequent when using Hausdorff distance. In addition the information provided by functions dfw and dbw are useful to determine the curves orientation when curves are interpolated, as will be described in Section 4.4.

Given a transformation $T: \mathbb{R}^2 \rightarrow \mathbb{R}^2$. we also need to compute a transformed curve Tc. If T is a linear transformation, the curve Tc can be obtained simply by applying T to the control points of c. But it usually will not work for non-linear transformations. So, we use a recursive fitting algorithm to find a curve formed by the cubic segments whose image is close to Tc. The algorithm is the following: given a transformation T and a cubic Bézier segment c, calculate the control points of a cubic Bézier curve Tc, where $Tc[0] := T(c[0])$, $Tc[3] := T(c[3])$, and $Tc[1]$, $Tc[2]$ are calculated minimizing the sum in a least square sense:

$$S = \sum_{i=0}^{n} \|Tc(i/n) - T(c(i/n))\|^2,$$

for a fixed number n, we used $n = 9$, which usually captures well the distortion caused by T. We compare the normalized error $\sqrt{S}/\|b\|$, where b is the residual vector of the linear system, against a fixed threshold ϵ. If it is above ϵ then c is split into two cubic Bézier curves, one equal to $c(t)$ for $t \in [0, 0.5]$ and the other equal to $c(t)$ for $t \in [0.5, 1]$, and the process is repeated for these smaller curves. This process is applied to all cubic segments of a curve. For this work we used $\epsilon = 1$. Fig. 2 illustrates a result of this operation. For a more detailed explanation of a similar adaptive fitting method, please refer to method described by Schneider [30].

4. Method

Each key frame is separated into two parts. The first is the set of guidelines. The second is the detailed drawing, which we call the final art. Fig. 3 depicts the complete pipeline of the proposed method.

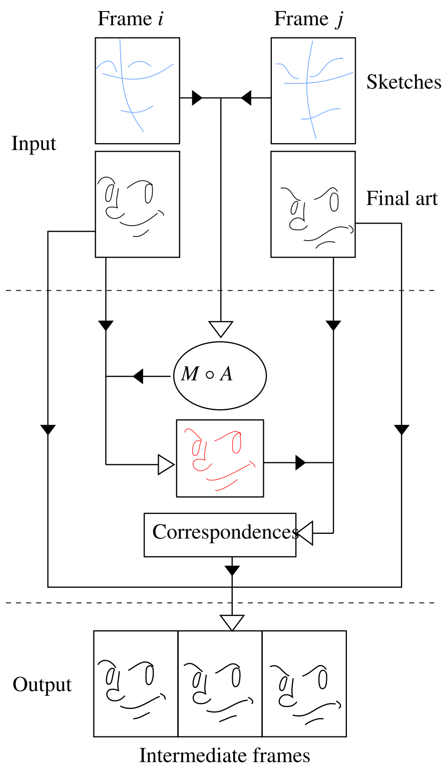


Fig. 3. Pipeline of the proposed method. The guidelines from two frames are processed to calculate a transformation $M \circ A$. which is applied to key frame i to generate a warped drawing (red). Correspondences are then found between the transformed frame i and key frame j, and are used to generate the intermediate frames. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The central problem is to find correspondences between one key frame and the next. For this purpose we could use directly the final art, but it usually includes a set of small details that are hard to handle. By contrast, the guidelines are simpler while still conveying the basis shape of the drawings, making it easier to extract the movement information between two key frames, and use it to simplify the inference of curve correspondences between consecutive keyframes.

Our system first receives as input two consecutive key frames i and j, and automatically creates pairs (g_i, g_j) of corresponding guidelines (Section 4.1). Then, a similarity transformation A is computed and applied to the guideline strokes of key frame i (Section 4.2). We further refine this transformation by calculating a non-linear morph function M that approximates both sets of guideline strokes (j and transformed i). If the key frame i already contains its final art, it is transformed by the morph function $M \circ A$ enabling a visual feedback to help the artist draw key frame j. It is important to observe that our final goal is to create inbetween frames that interpolate the final art, not the guidelines. Therefore, we proceed by computing the correspondences between the final art of frame j and transformed frame i (Section 4.3). Finally, we define a function for each pair of correlated strokes (c_i, c_j) to interpolate intermediate frames as needed (Section 4.4).

4.1. Processing guidelines

The first step is to create the guidelines for a pair of consecutive key frames. These guidelines are expected to be much simpler than the final art, simple enough to describe only an average change between key frames. Each guideline from one key frame may correspond to a guideline of the next key frame. This correspondence does not have to be one-to-one, i.e., there may be guidelines without correspondence from one key frame to the next, but the algorithm works best if the number of guidelines is the same (or at least approximately) in all key frames.

The more information we have about the curves movements, the more reliable is the correspondence inference result. However, during this first step we still do not have such information. So, to compute the matching, we first align the guidelines of each key frame and look for a rotation and scale that best describe the movement, as detailed next.

The guidelines of each key frame are translated, such that the barycenter of all guidelines in one frame is moved to the origin of $\mathbb{R}^2$. This barycenter is calculated as the average of all control points defining the guidelines of one key frame. This operation helps the search for corresponding guidelines, avoiding miscalculations due to a possible translation.

Next, the control points of all guidelines in key frame i are scaled, such that the key frames have similar dimensions. The scale factor is defined by $\max(w_j, h_j)/\max(w_i, h_i)$, where w_i, h_i are the dimensions of the minimum bounding box containing all control points of the guidelines in key frame i, and similarly for w_j, h_j.

A bipartite graph K_{N_i,N_j} is defined, where each vertex of the graph represents a guideline, so N_i and N_j represent the number of guidelines in i and j, respectively, and each edge connects one guideline from i to one in j. The edge connecting guideline g_i with g_j is attributed the weight $-d(g_i, g_j)$, where d is the distance function defined in Eq. (1) (see Fig. 6 top for an example).

We then use Kuhn's [31] augmenting path algorithm to find the match P with maximum weight and maximum cardinality, resulting in a possible correspondence between guidelines from i and j. However, j could be rotated creating mismatches. Thus, we rotate i by a set of angles $\alpha \cdot k$, $k \in \{1, 2, \dots, 360°/\alpha\}$, calculate P for each angle, and keep the one with the maximum weight. In this work we use $\alpha = 45°$.

4.2. Calculating transformations

Once we have the correspondences between guidelines from two sequential key frames, it is necessary to find a transformation that morphs the guidelines from key frame i to their corresponding guidelines in key frame j. This process is described next.

Let g_i, g_j be a pair of corresponding guidelines, and p_i, p_j be the sample of g_i and g_j that are used in the calculation of the distance between these curves, with points in p_j reversed if $\mathrm{dfw}(g_i, g_j) > \mathrm{dbw}(g_i, g_j)$. A similarity transformation A is computed (i.e., a combination of rotation, translation and scale) minimizing the sum:

$$\sum_{(p_i,p_j)\in G} \sum_{k=0}^{n} \|Ap_i[k] - p_j[k]\|^2,$$

where n is the number of samples used for each pair (p_i, p_j), and G is the union of all pairs (p_i, p_j). We solve for the elements of A using a least-squared approach [32].

This operation is denoted by

$$A = \mathrm{similarity}(G^i, G^j), \quad (2)$$

where G^i is the union of all samples of guidelines in key frame i, similarly for G^j.

This transformation roughly places one set over the other, but their shapes are still different, since we are restricted to a similarity.

To further approximate the two sets, a second transformation M is computed using an MFFD (multilevel free-form deformation) [33]. This method takes a set of point pairs in the plane and generates a C^2-continuous one-to-one function moving each point

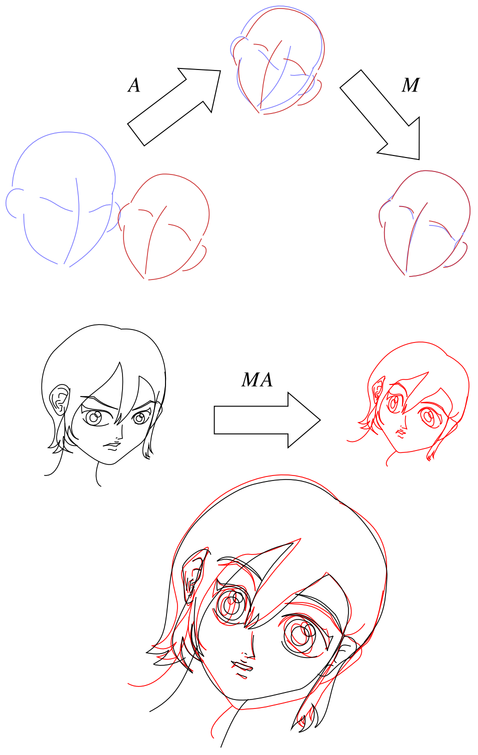


Fig. 4. The similarity transformation A and MFFD Transformation M are calculated from the guidelines (top) and they are composed and applied to the first key frame (middle). On the bottom, the transformed key frame (red) is compared with the second key frame (black). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

as close as possible to its pair. A 2D B-spline approximation is applied to a hierarchy of control grids, minimizing the distance between points from the two input sets. The guidelines are resampled such that two consecutive sample points from Ag_i and from g_j fit inside a cell of the finest control grid used for the calculation of the morph, and the sets G^i and G^j are update accordingly. The transformation M is calculated to morph the points AG^i to G^j, and can be applied to any point in the domain of the drawing (a virtual canvas).

By applying the composition $M \circ A$ to the curves in the final art of the first frame, we get a distorted drawing similar to the drawing in the second key frame. Fig. 4 shows an example of this operation.

It is important to compute the similarity transformation A prior to M because in some cases the animation is predominantly a simple rotation and/or scale between the key frames, so AG^i will be much closer to G^j, reducing the distortion caused by M. This is illustrated on Fig. 5.

4.3. Computing drawing correspondences

At this point, the transformed drawing of the key frame i is very similar to the drawing of the next key frame. This is crucial to help the next step of our method, i.e., the computation of correspondences between the final art curves.

To find this correspondences another bipartite graph matching problem is solved, where each vertex of the graph is a curve from the final art drawing, and each edge connects curves from the two key frames. We define a bipartite graph K_{N_i,N_j} where N_i and N_j are now the number of final art curves of i and j, respectively. Each edge has a weight defined as minus the distance between the curves, as the one used when matching guidelines. Applying the

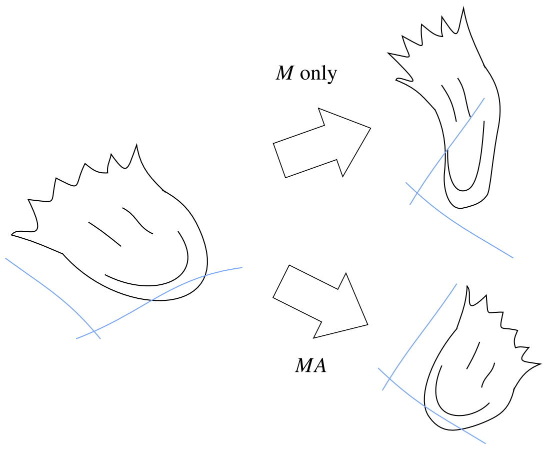


Fig. 5. When the animation is predominantly a rotation and scale, the morphed drawing will be much less distorted when A is calculated and applied before M (bottom right) compared to the result of applying M only (top right).

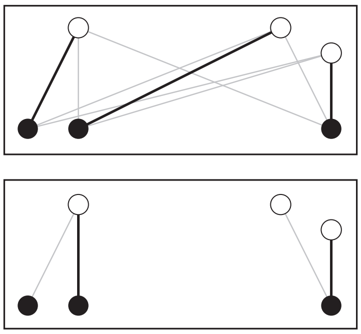


Fig. 6. Bipartite graph matching example. Vertices are represented with circles, and the edge weight is proportional to its length. We want to match white vertices with black vertices. Edges are marked with lines. Thick black lines are the matching result. On top, the matching algorithm using all edges. On bottom, distant vertices were removed from the graph and are not considered by the matching algorithm.

maximum cardinality and maximum weight matching algorithm, we find the correspondences between strokes in the final art.

All the same, sometimes it may match curves that are too far apart, which usually is undesirable. Fig. 6 illustrates this case. This situation usually occurs when curves from one key frame are not present in the other key frame. To avoid these unwanted matches it is enough to not include edges that correspond to distances above a given threshold. The threshold is defined as a percentage value of the maximum distance between curves. The user can dynamically adjust this threshold until the result includes all the correct correspondences and avoids incorrect ones. This is the only parameter in the system that must be set by the user.

Since N_i and N_j can be different and some edges are not included, it may happen that some curves are not matched. This is common, for example, when curves represent parts of the object that are occluded in one of the key frames. These curves will simply be ignored by our interpolation, so the animator must handle them manually, by editing the intermediate frames to add the missing strokes. However, in most cases the user can avoid this situation by using layers, see Section 5.3.

4.4. Interpolating key frames

Finally, intermediate frames are calculated by interpolating corresponding curves from the key frames. There are several approaches that can be used for this purpose, like the methods developed by Sederberg and Greenwood [34], Cong and Parvin [35], Fu

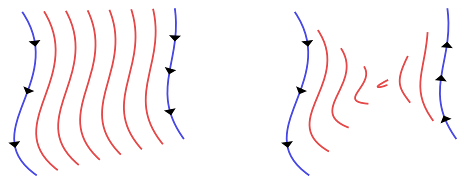
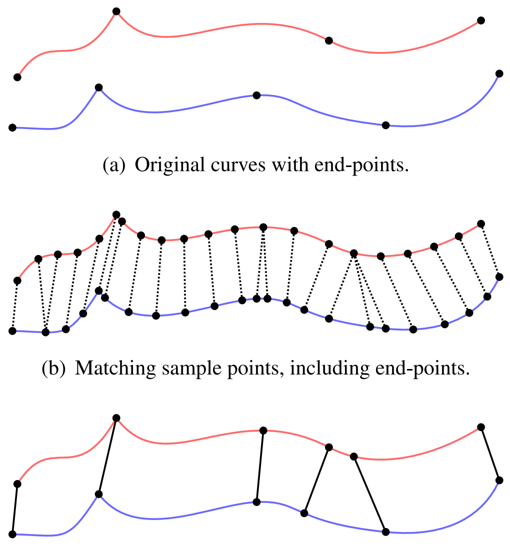


Fig. 7. Curves (in blue) may be interpolated in two ways according to the order of the control points (interpolated curves are shown in red). Arrows indicate the orientation used for each curve. The preferred way is found by comparing the values of dfw and dbw of these curves. In this case it is the one on the left. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)


(a) Original curves with end-points.
(b) Matching sample points, including end-points.

(c) Matchings containing end-points.

Fig. 8. Equalizing the number of segments of two curves.

et al. [36], and Whited and Rossignac [37]. We use a method that works by matching similar shapes based on the method by Sederberg and Greenwood [13], it is simple and yields satisfactory results. Nonetheless, the overall algorithm does not depend on it, so it could be replaced by other methods if some specific features are desired. The method we employ is described next.

Suppose we want to interpolate curves c_i and c_j. First of all, there are two options for matching the two curves: using the same orientation for both of them, or inverting the orientation of one of them, see Fig. 7. To check which one produces the best match we compare the values of $\mathrm{dfw}(c_i, c_j)$ and $\mathrm{dbw}(c_i, c_j)$. If $\mathrm{dfw}(c_i, c_j) > \mathrm{dbw}(c_i, c_j)$, then the control points of one of the curves are reversed.

A straightforward way to interpolate the curves is to match each cubic segment from one curve to the other. Nevertheless, usually they do not match naturally, i.e., the curves contain different numbers of segments, or even when they are equal, one segment may be mapped into a much smaller (or larger) one. To solve this issue, it is necessary to equalize the number of segments and keep each pair with similar shapes. This operation is illustrated in Fig. 8, and is detailed next.

Fig. 8(a) shows an example of two curves, and the end-points of their cubic Bézier segments. Let $0 = t_0^a < t_1^a < \cdots < t_{n_a}^a = 1$, be the sequence used in the parametrization of curve c_i (see Section 3), where n_a is the number of cubic Bézier segments in c_i, and likewise $0 = t_0^b < t_1^b < \cdots < t_{n_b}^b = 1$ for curve c_j. Curves c_i and c_j are uniformly sampled, using $n = 5 \cdot \max(n_a, n_b)$. Not all end-points correspond to one of the samples positions, in fact most of the points are very unlikely to be included in the uniform sampling,

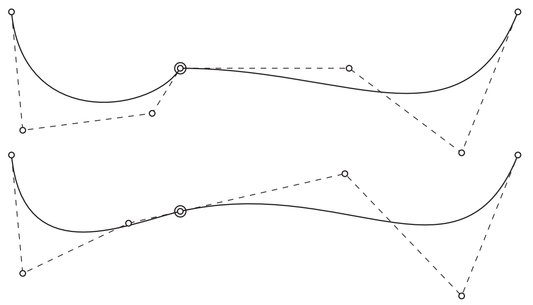


Fig. 9. G1 continuity enforced at a control point. On top: a curve with G1 discontinuity at the point marked with a large circle. On bottom: resulting curve after forcing G1 continuity.

so we add these end-points. To avoid very small components, if an end-point is too close (given a certain tolerance) to another sample, it replaces this sample. This is done when $|t - t_k| < 0.2/(n-1)$, where t is the parameter used for the sample, and t_k the parameter of the end-point, for each curve c_i and c_j. We then apply the algorithm proposed by Sederberg and Greenwood [13], which gives us a matching between pairs of sample points. Fig. 8(b) shows the result of the matching of sampling points (including the end-points). From the resulting matching, we use only the pairs that contain at least one end-point, creating two sequences of parameters $0 < \tilde{t}^a_0 < \cdots < \tilde{t}^a_{\tilde{n}} = 1$ and $0 < \tilde{t}^b_0 < \cdots < \tilde{t}^b_{\tilde{n}} = 1$, such that the sample $c_i(\tilde{t}^a_k)$ matches $c_j(\tilde{t}^b_k)$, for $k \in [0, \ldots, \tilde{n}]$, where $\tilde{n}+1$ samples were considered. Then we create curve $\tilde{c}_i$ that is equal to c_i, but with $\tilde{n}$ cubic Bézier segments, created by cutting c_i at every value t^a_k that is not an end-point, likewise for curve c_j, creating curve $\tilde{c}_j$. Fig. 8(c) shows this operation. Now we have a pair of curves with the same number of control points and each pair of segments have similar shape.

Now define the function $\text{interp}(c^1, c^2, \alpha)$ that computes a new curve that is an interpolation of curves c^1 and c^2, both with the same number of control points. The parameter $\alpha \in [0,\ 1]$ controls the interpolation such that $\text{interp}(c^1, c^2, 0) = c^1$ and $\text{interp}(c^1, c^2, 1) = c^2$. First, the control points of curves c^1 and c^2 are interpolated with parameter α, and then G1 continuity is enforced for each control point if curves c^1 and c^2 are both G1 continuous at the corresponding control points (it is not a problem if at least one of the curves is not G1). To guarantee G1 continuity at a control point q, it should be forced to be collinear with the previous point p and with the next point r. We accomplish this by moving p to $q - \frac{\|u\|}{\|d\|}d$, and r to $q + \frac{\|v\|}{\|d\|}d$, where $u = p - q$, $v = r - q$ and $d = r - p$. Fig. 9 shows an example of this operation. Some results of function interp can be seen in Fig. 10.

For frame k between i and j, we want to find curve c_k from an interpolation of corresponding curves c_i and c_j. Let $\tilde{c}_i$ and $\tilde{c}_j$ be the result of the method described above to equalize the number of control points of curves c_i and c_j. Let A be the similarity transformation mapping guidelines computed in the beginning of the method (Section 4.2). The resulting curve c_k is defined then by

$$c_k = A_k(\text{interp}(A(\tilde{c}_i), \tilde{c}_j, \alpha)), \qquad (3)$$

where $A(\tilde{c}_i)$ is the curve obtained by applying transformation A to the control points of $\tilde{c}_i$, and A_k is a similarity transformation that moves the interpolated curve to an intermediate position between the original curves c_i and c_j. The parameter $\alpha \in (0,\ 1)$ is further described in Section 5.1. To calculate A_k, let s, θ and t be the parameters of A^{-1} (the inverse of A), representing its scale, angle of rotation and translation, respectively. Define $A_{i\alpha}$ to be the similarity transformation with scale parameter $(1-\alpha)s + \alpha$, rotation angle $(1-\alpha)\theta$, and translation vector $(1-\alpha)t$. This is equivalent to the interpolation of A^{-1} with the identity transformation.

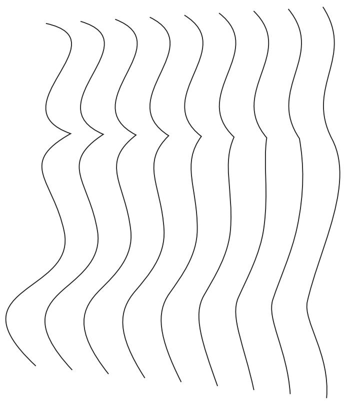


Fig. 10. Result of the function interp: the leftmost and the rightmost curves are interpolated using seven different values for $\alpha \in [0,\ 1]$.

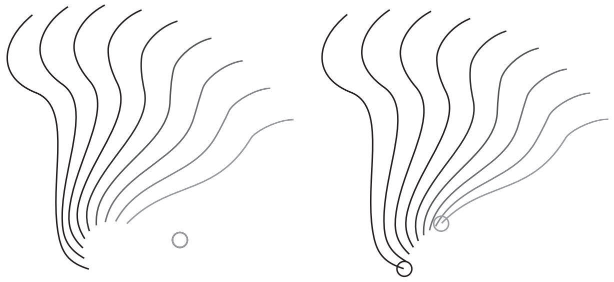


Fig. 11. Example of curve interpolation using different rotation centers (marked as circles). On the left, the center points for both frames are the same. On the right, the center points are located in extreme points of the curves, giving a different result.

As these transformations are centered at the point (0, 0), if α is varied between 0 and 1 while applying $A_{i\alpha}$ to a point, an undesirable trajectory is usually created, because the shape rotates around (0, 0) instead of its center of rotation. To correct this the user may specify centers of rotations, which are used to translate the resulting points to a desirable position. Let o_i and o_j be the centers of rotations of the two key frames being processed, $o_\alpha = (1-\alpha)o_i + \alpha o_j$, and T be a translation with direction $o_\alpha - A_{i\alpha}(o_j)$. Finally, define $A_k = A_{i\alpha} \circ T$. Fig. 11 shows the result of a complete interpolation between two curves, using rotation centers in different positions.

5. Extensions

In this section we present three extensions to render or system more useful and flexible: control of animation speed; definition of regions; and layer hierarchy.

5.1. Timing

It is essential for the animator to be able to control the timing, i.e., speed up or down the current animation. Even more, the animation may have varying speed, for example, a bouncing ball must present an accelerated movement when falling, and a decelerated movement when rising, otherwise it will not look natural.

In this work the user can control the timing by adding/removing frames between key frames (causing a linear change of speed between key frames), and by manipulating curves that guide the spacing of frames, allowing for non linear acceleration. For every pair of key frames there is a curve, which is initially an identity function, such that the speed is constant

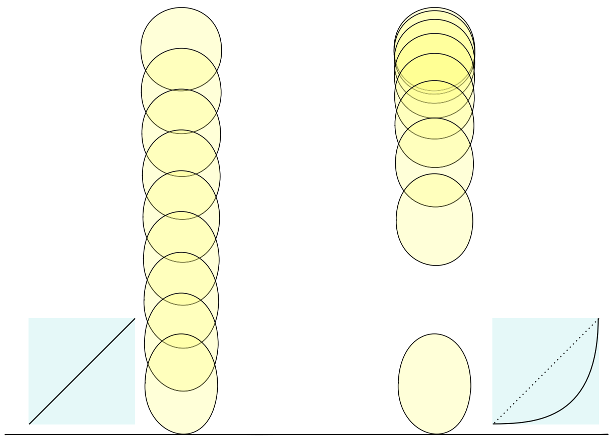


Fig. 12. A falling ball with constant speed (left) and accelerated (right). The balls were painted translucent to aid visualization.

from one key frame to the next. This curve is defined by a cubic Bézier segment, whose first and last control points are fixed to (0, 0) and (1, 1), respectively, and the other control points can be manipulated (but restricted to the unit square). It is possible to make an accelerated/decelerated movement by bending this curve below/above the identity line.

The curve is used to define the parameter α used when interpolating the key frames (see Section 4.4), such that $\alpha = C((k-i)/(j-i))$, where C is the curve parametrized by x-coordinate.

Fig. 12 shows an example of a falling ball with different speed.

5.2. Regions

To allow for colored objects, the artist may select a sequence of curves to define the contour of a region, which may be colored as desired. The curves may be removed from the drawing, but kept in the representation of the regions. In addition, straight segments are added between consecutive curves to fill the gap between them, such that it is possible to create regions with open contours, even though they are represented as closed ones.

After the curves are matched it is necessary also to infer the correspondence of regions. Once more, the bipartite matching algorithm is used. Each node of the graph now represents a region, and the edges connect each pair of regions that have at least one pair of matched curves in common. The edge weights are given by minus the distance between regions, defined as the maximum between the distances of the corresponding curves (as before, considering the first frame being morphed by $M \circ A$).

To interpolate regions, we need to interpolate all the curves used in their contours. Some curves from the regions may not be matched by the previous algorithm (Section 4.3), so new matches are inferred for them. Let R_i be a region defined by m_i contour curves $c_i^1, c_i^2, \ldots, c_i^{m_i}$, and R_j a region defined by m_j curves $c_j^1, c_j^2, \ldots, c_j^{m_j}$. Suppose, without loss of generality, that the curves follow the same orientation. Let (c_i^p, c_j^q) be a pair of matched curves, and (c_i^r, c_j^s) be the next pair of matched curves following the orientation of the regions. Note that they might be disjoint pairs, in this case a new correspondence is created by matching the curve formed by the sequence of curves between c_i^p and c_i^r with the curve formed by the sequence between c_j^q and c_j^s. If there is no curve between c_i^p and c_i^r, then a new curve is created formed by a single point (the intersection point between these curves), i.e., during the animation these curves will shrink to the size of a point. Likewise for curves c_j^q and c_j^s. Finally, the new matched curves are interpolated using the method described in Section 4.4. Fig. 13 illustrates this process.

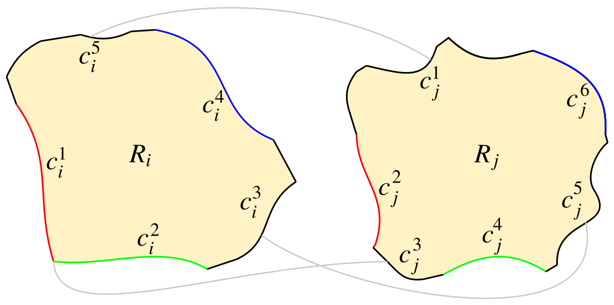


Fig. 13. Correspondences between regions. Matched curves are marked with the same colors. The gray lines indicate new correspondences for unmatched intermediate curves (in black). Note how a new curve c_j^3 appears on the right region, thus it is matched to a single vertex on the left. When animating the vertex will expand to become a curve.

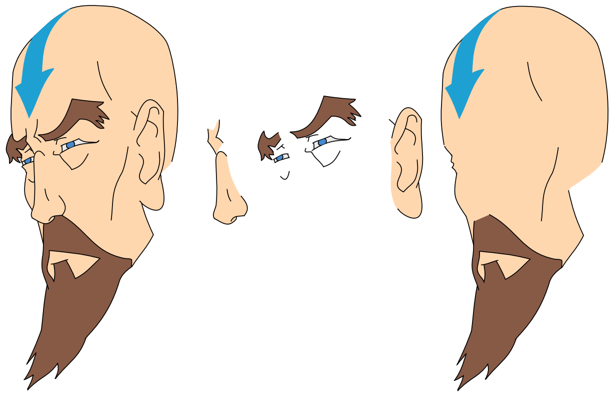


Fig. 14. Drawing (on the left) with its four layers (on the right).

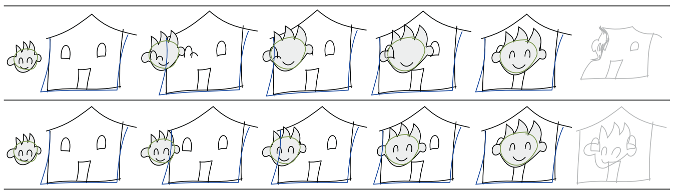


Fig. 15. Example of an animation with two keyframes where elements cross each other, without the use of layers (top) and with the use of layers (bottom). Note how without layers the curves are wrongly matched when generating the inbetweens. The last drawing to right in each row is the first keyframe warped to the space of the second keyframe. There are only two guidelines in this animation, one in green for the face and one in blue for the house. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5.3. Layer hierarchy

Drawings often contain overlapping curves from different parts of what is being depicted. This may mislead the search of corresponding curves, where undesired pairs may be formed if their match weight is smaller than the weight of the expected pairs. To improve the animator's control, our system allows splitting parts of the drawing into layers, and the inbetweening method is applied separately for each layer. Fig. 14 shows an example of how a drawing may be separated into layers.

The layer composition not only avoids the formation of unwanted pairs, but it also allows elements to cross each other (see Fig. 15), which is more convenient than restricting the animations to non-crossing elements. Furthermore, separating drawings into layers is a process typically used by artists [1].

Sometimes it is useful to specify a correlation between the layers, such that the movement from one “parent” layer is applied to “children” layers. This leads to a hierarchy of layers, where the transformations applied to a layer is also applied to its sublayers. If the current layer is not a sub-layer, then the previous Eqs. (2) and (3) are used. Otherwise the equations have to be reformulated. Suppose the layer currently being processed is a sub-layer of layer p. First, the similarity transformation (Eq. (2)) is redefined as

$$A = \text{similarity}(A_p(G^i), G^j), \tag{4}$$

where A_p is the similarity transformation from layer p. Eq. (3) is accordingly changed to

$$c_k = A_{kp} \circ A_k(\text{interp}(A \circ A_p(\tilde{c}_i), \tilde{c}_j, \alpha)), \tag{5}$$

where A_{kp} corresponds to the intermediate transformation A_k calculated for layer p. When computing A_k, the center of rotation changes to $o_\alpha = (1-\alpha)A_p(o_i) + \alpha o_j$.

It is also important to control the ordering between layers, such that one can specify if one layer should appear in front or behind another layer. This ordering may change during the animation, for example, an arm may appear in front of the body at some frames but it may move behind the body at a later time. To control the ordering, each layer has an index for each frame, working as a third dimension, such that layers with lower indices are placed behind layers with higher indices. It is only necessary to specify the index for the key frames, as intermediate indices are interpolated from the key frames indices using the α parameter.

6. Results

The proposed method was implemented in C++ and embedded in an application, where the artist can draw curves (using the approach by Frisken [29]), create regions by selecting their boundary curves, define the colors that are used, erase curves or regions, adjust control points, adjust the points of rotation, and manage layers (add, remove, hide, change position, create sub layers). There are two types of virtual pens that can be used, one for the guidelines, and another for the final art. Regions can only be created from final art curves. After finishing one key frame, and the guidelines of the next, the system automatically calculates and displays the warped version of the first frame, working as a virtual light table. For all examples we achieved interactive timings (less than one second to obtain all inbetween frames). All the results were done with an Intel Core i7, with 16 GB of RAM, but we achieved similar performance even with a simple netbook. The actual animations have more inbetween frames than shown here, they were simplified to save space. Nevertheless the key frames are completely shown, always depicted inside a rectangle. Guidelines are colored coded to indicate the correspondences.

Figs. 14and 16 are an artist reproduction inspired by the animation series The Legend of Korra. Fig. 16 illustrates the result for two frames of a character face. The set of guidelines is simple (19 strokes) but it is sufficient to define the morph between the frames. Some details are partially occluded in the first frame, such as the wrinkle under the right eye, and the part behind the ear. Thanks to the layer separation that was not a problem for the method.

Fig. 17 shows some frames from a walking horse sequence. Five inbetween frames were generated for each pair of key frame. The arrows indicate the sequence of the frames. Even though there are large occlusions when the legs cross each other, they are correctly handled as they are separated into layers.

We asked some professional animators to test the system, creating some animations. Fig. 18 shows a sample animation created by a professional animator, using 7 layers. In Fig. 19 there is a sample animation created by another professional animator. It has only two key frames, but the artist decided to divide them into 28 layers to create different shadings. In Fig. 20, there is a reproduction from the animation Fullmetal Alchemist: Brotherhood.

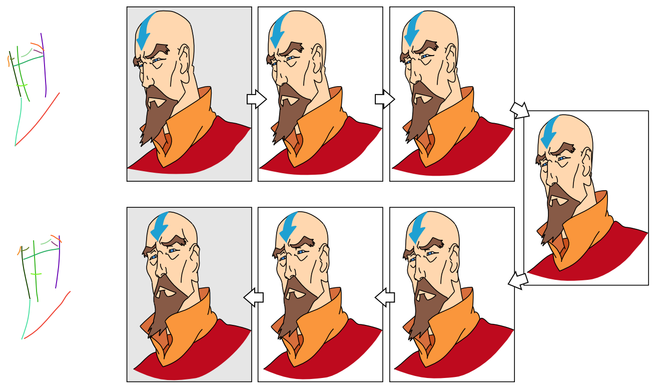


Fig. 16. Character turning his head.

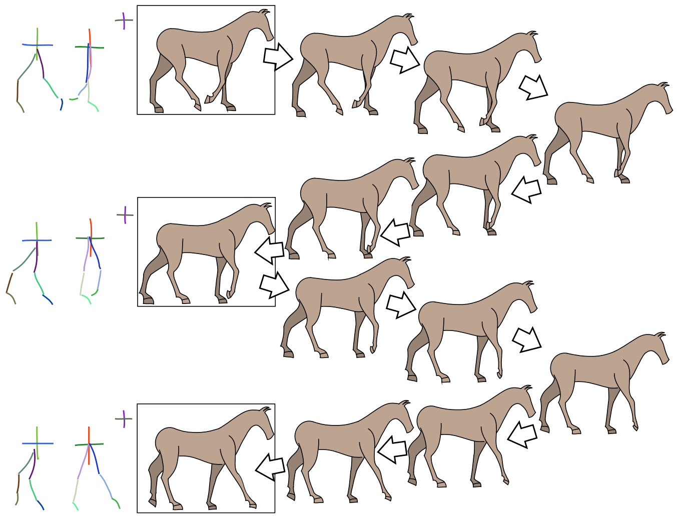


Fig. 17. Walking horse.

The next results were created based in examples from The Animator’s Survival Kit [1]. Fig. 21 shows a sneaking man, with 3 key frames, and 5 intermediate frames between each pair of key frames. This example presents a large distortion between the drawings. Nevertheless, it is appropriately handled by our method. In Fig. 22, a woman is moving a hand and hips, where 8 key frames were used, with 3 intermediate frames between each pair of key frames. The arm movement is properly inferred by separating the arm into sub-layers, and putting the center of rotation of each sublayer at a joint. In Fig. 23, there is a man screaming, where 8 key frames were used, with 3 intermediate frames between each pair of key frames.

Table 1 provides more information about each example. Note how all inbetweens for all examples were processed in under one second.

7. Discussion

7.1. Limitations

Our method is able to handle quite large changes, such as in Fig. 21. However, as usual in computer aided inbetweening methods, the technique is not able to deal with exaggerated changes in the drawing. As the input is two-dimensional, it does not hold

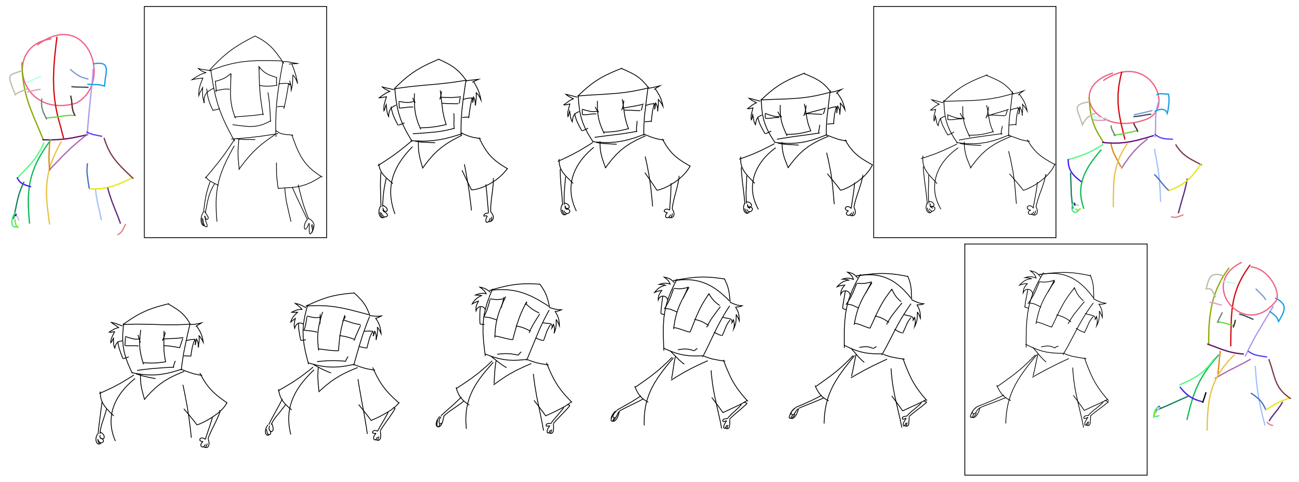


Fig. 18. Sample animation. ©Diogo Viegas.

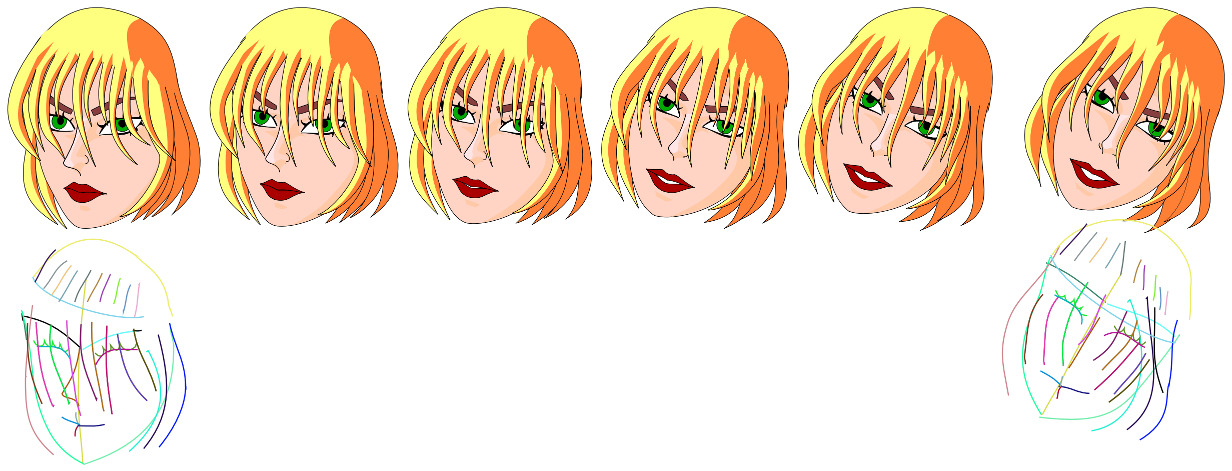


Fig. 19. Sample animation. ©Leandro Araujo.

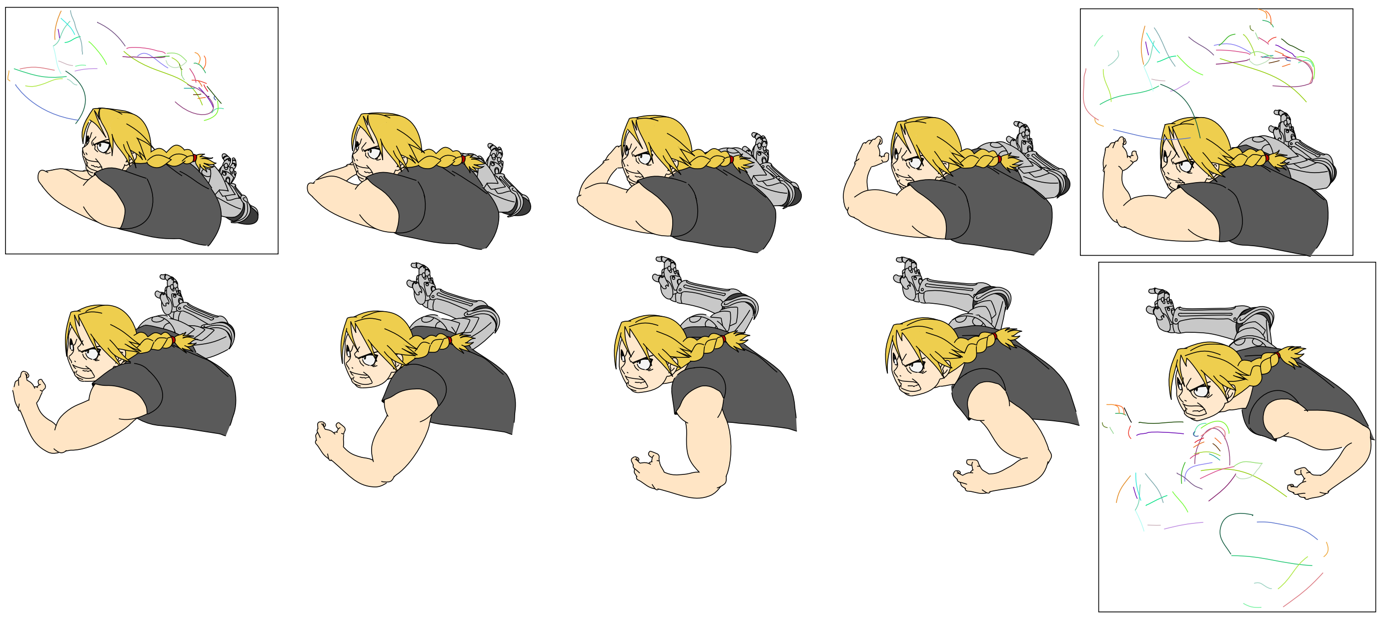


Fig. 20. Reproduction from the animation Fullmetal Alchemist: Brotherhood. Guidelines are displaced for clarity purposes. Animation follows left to right, top to bottom.

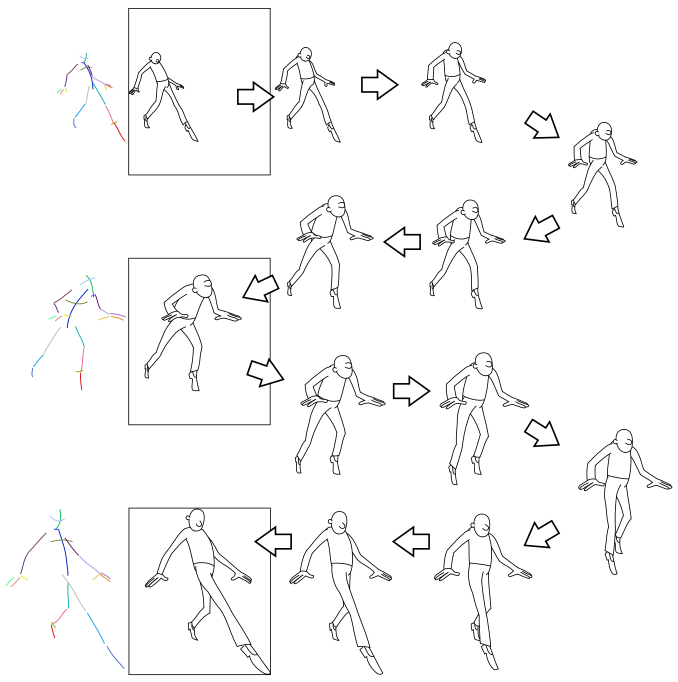


Fig. 21. Sneaking man.

all the necessary information to properly handle the shape of the elements in the intermediate frames.

The method described works fine when the movement in a layer from one key frame to the next can be described approximately by a similarity transformation, otherwise the algorithms are prone to provide undesirable matchings. The drawing must be organized into layers in order to work for more complex movements. This is illustrated in Fig. 24.

Another common limitation for all 2D animation systems is when large parts of the subject are completely occluded from one frame to the other. Our system can handle that, but it may generate a not reasonable result. It can work if the occluding and occluded parts are located in different layers and their shapes are not related, as for example, the horse legs in Fig. 17. Similarly, if an eye is hidden by its own face, it should be located in another layer, as occurred between the second and third key frames in the last example (Fig. 22), when the woman turned her head revealing her right eye. Also, between the last pair of key frames in Fig. 22, the right hand changes orientation, such that the palm of the hand is visible in one key frame, but the back of the hand is visible in the other. This part of the animation can be seen more detailed in Fig. 25. In this case, the interpolated frames do not reflect the proper hand movement.

These artefacts usually happen in only a few specific places, and they can be manually corrected by the artist. Although this might seem to be a laborious task, it is much simpler than creating all in-between frames manually. Actually, once the artist gets used to the system, he can even predict where this might happen and handle it promptly.

Finally, our method to control the animation timing does not take into account G1 continuity, so the artist has to be careful to not create acceleration jumps when passing through a keyframe. This could be better addressed, but we have actually noted no such issues without having to actually put any additional manual effort to create smooth passages.

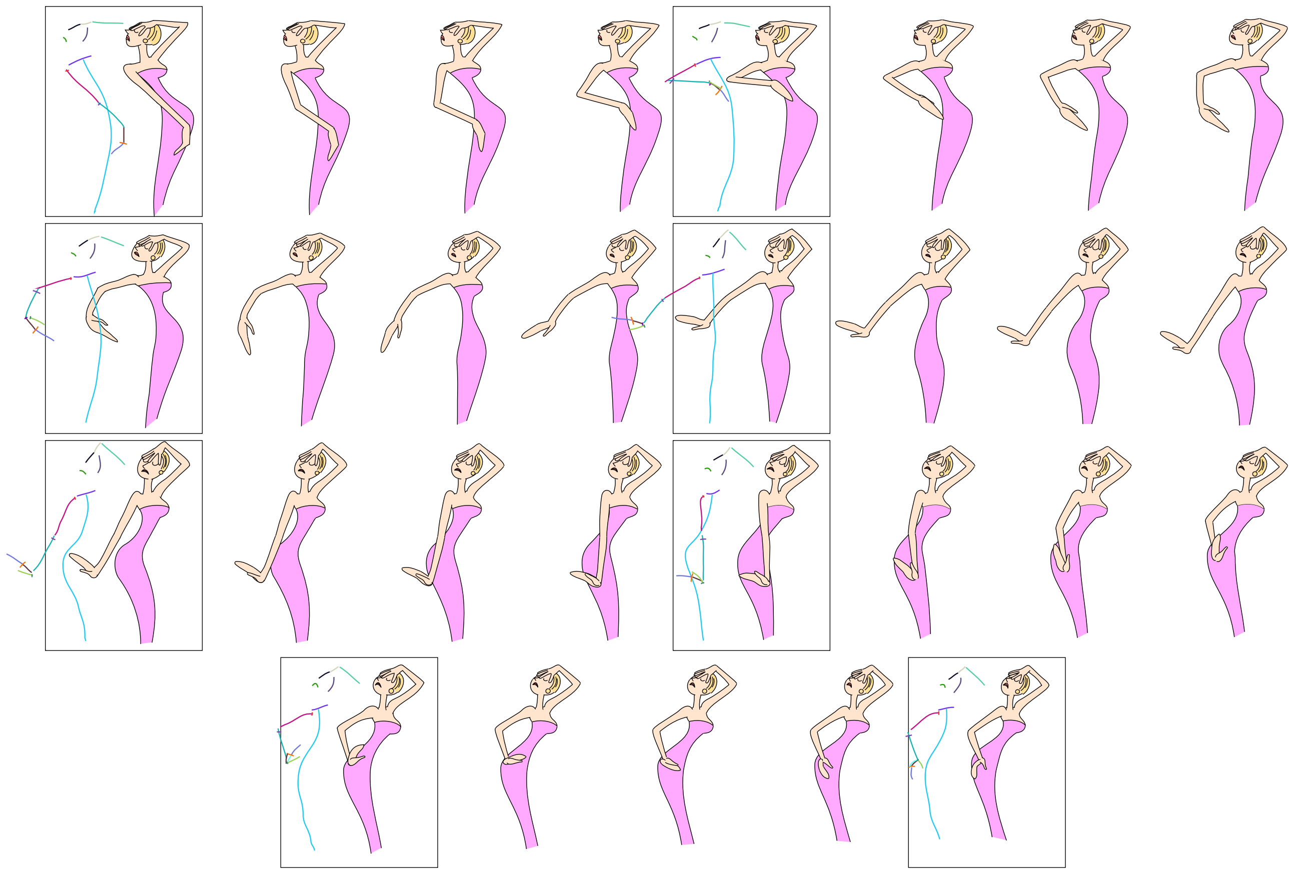


Fig. 22. Woman dancing and moving hand. Animation follows left to right, top to bottom.

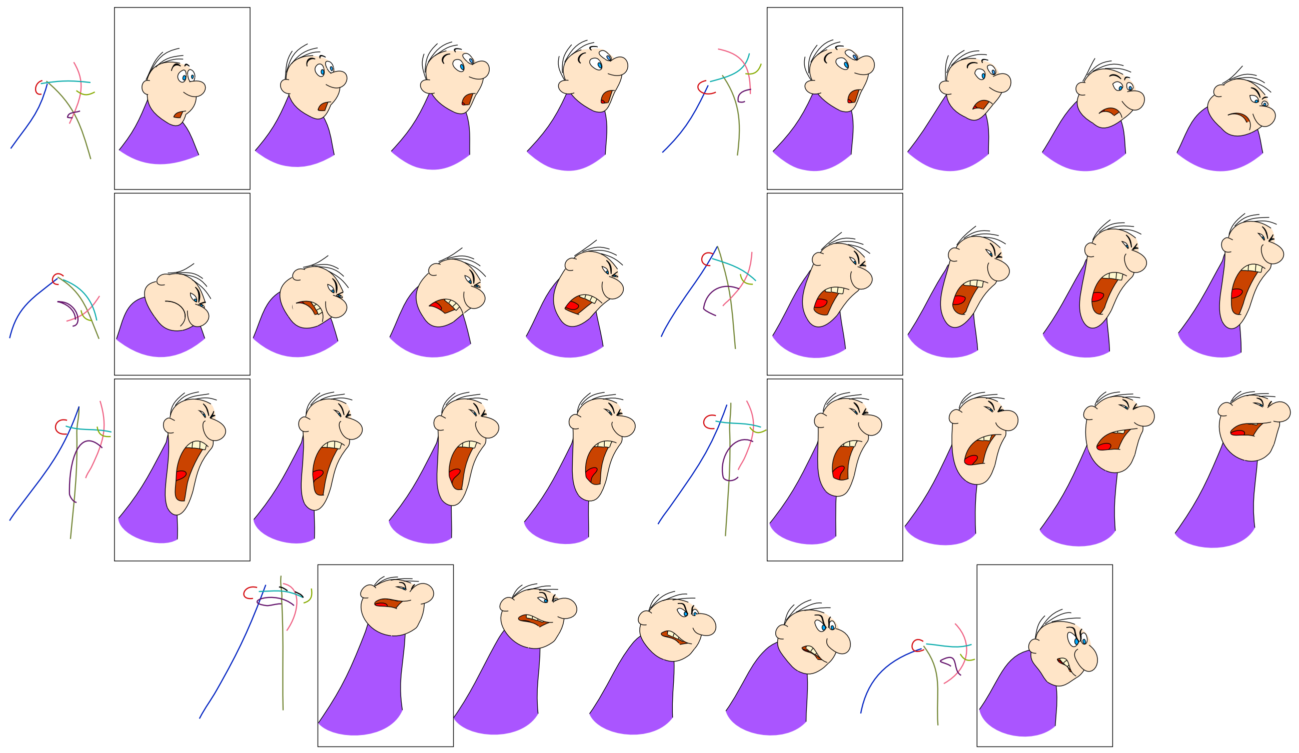


Fig. 23. Screaming man. Animation follows left to right, top to bottom.

7.2. Comparison with previous works

In this section we briefly compare our work with two closely related methods. The first is Xing et al.'s [28]. Their main goal is to accelerate the production of key frames. Even though we also provide some aid in drawing subsequent key frames with the digital light table idea, our main concern is to generate a smooth sequence of inbetweenings. Observing Xing et al.'s results, the jumps between keyframes are quite evident (see the walking example), while our final animations are continuously smooth, that is, once the animation is produced it should not be possible to point out which are the original key frames. This lack of temporal smoothing might also be perceived in more complex scenes as a stretching/shrinking effect (Jerry's example). In all, we do not argue that our method should replace Xing et al.'s, but rather complement it, as our main contributions touch different animation issues.

Another popular approach is as-rigid-as-possible (ARAP) transformations [12]. Even though it works well for some applications, there are a few drawbacks for generating inbetweens: the registration is seldom exact (it oscillates near the target but never converges perfectly), thus it might lead to jumps when passing from one keyframe to the next; it usually takes a few seconds to converge; it might get trapped in local minima. We have also experimented using the ARAP technique as a first approximation to help the matching algorithm (instead of the similarity transformation for example), but the method also contains a number of parameters that do not work well for all cases, such as: image resolution; grid size; block matching parameters; and number of iterations. Fig. 26 illustrates the use of the ARAP method and how changing a single parameter might influence in the resulting transformation. Even though we could probably make the approach work for all our examples, it would often entail adjusting some parameters. One of our goals is to exactly avoid parameter tweaking for each new animation, and as previously discussed, the only one that needs intervention is the matching distance threshold.

As in our method, Noris et al. [19] also employ a two step approach: a global registration followed by a stroke matching algorithm. Their first step uses a modification of the ARAP [12] method discussed above. Their stroke matching algorithm, however, works in a greedy-like manner for pairs of frames F_1 and F_2, by finding for each stroke in F_1 the best match in F_2 using a criterion based on the Hausdorff distance. Moreover, some strokes from the second frame might be left unmatched. Differently, our bipartite graph

Table 1
Statistics for the results. Columns are: animation name; corresponding Figure; number of layers; number of guidelines for each keyframe; number of curves and regions; number of keyframes; total number of intermediate generated frames; distance threshold; and total time to generate all inbetweens.

	Fig	Layers	Guidelines	C&R	KF	Inbetweens	ϵ (%)	Time (ms)
Tenzin	16	7	12	127	3	48	20	320
Horse	17	3	18	50	9	40	20	189
Diogo Viegas	18	7	32	56	4	30	40	125
Leandro Araujo	19	28	51	291	2	37	50	404s
Edward	20	17	46	281	3	65	55	858
Sneaking man	21	5	22	56	3	78	55	193
Woman	22	6	14	58	8	35	40	282
Screaming	23	5	7	38	8	70	58	224

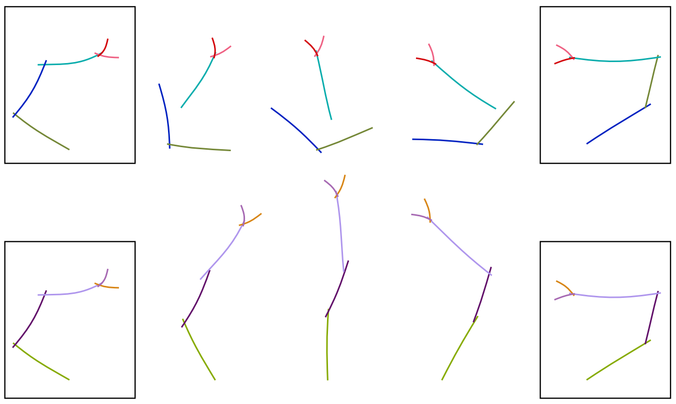


Fig. 24. Red lines on top: failure case, the matching algorithm fails when the motion in one single layer is more complex than a similarity transformation. Blue lines on bottom: the same example, separated into layers to correct the matching. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

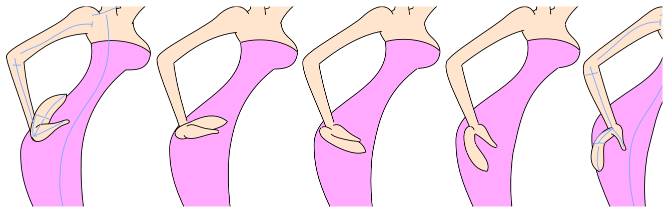


Fig. 25. Wrong prediction on the hand. As the movement is more complex and we do not have volume information the resulting animation is not what the animator expects.

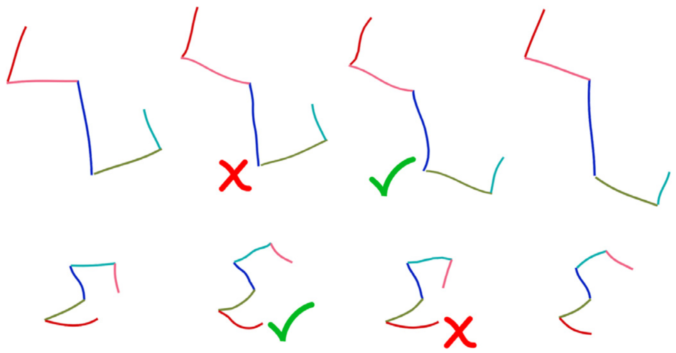


Fig. 26. Using the ARAP approach as an alternative to compute the rough alignment for the guidelines between a source and target frame. Each row represents a different example. From left to right: guidelines from the first frame (source); transformation using cell size 16 × 16; transformation using cell size 30 × 30; and guidelines from the second frame (target). The smaller cell size works for the second example (bottom row) but does not work for the first one (top row). For the larger cell size, it is the other way around. Cell size was the only parameter that was modified for this example. For more explanations on the parameters, we refer the reader to the original paper [12]. .

strategy finds the best mutual correspondences between pairs of frames, and not only from one frame to the next, tackling the issue of lack of smoothness across multiple frames as reported by Noris et al. [19]. Nevertheless, this issue is much less perceptible in a sketchy sequence than in clean drawings, thus our need for extra care in the matching algorithm.

8. Conclusion and future works

We presented a system to create 2D animations using a new method to automatically generate inbetween frames by processing the guidelines of the drawings. Drawing the guidelines impose very little extra work for the animator. Even though the guidelines are usually drawn before the actual drawing, the artist could also decide to draw them on top of the final art (but the initial preview would not be available in this case). Furthermore, we opt for the use of layers, that also follows the traditional animation pipeline, instead of restricting the classes of possible animations. The system generates a preview of a frame, which works as a virtual light table, and at the same time is used to infer the movement information between the frames. We were able to produce several compelling animations using our system even in difficult situations, such as large movements and distortions, and even correctly handling occlusion cases for many situations. Only a few minor issues are left for the artist to handle, considerable reducing the production time.

The procedure is not meant to replace other methods, but to be used in conjunction with them to improve the results. Most limitations can be avoided or reduced by using other methods in each part of the pipeline. The method worked well even using simple algorithms that would not be suitable to be used directly with the final art (without using the information from the guidelines). Therefore we believe our method can greatly reduce the artists effort while, at the same time, staying within the traditional animation pipeline.

As future work we will investigate how to handle more topological events appropriately, such as the case where curves break into two or more parts, to give more flexibility to the artist. We allow for some topological changes when dealing with regions, but more events could be incorporated using ideas from the work of Dalstein et al. [22], for example. In fact, using a more powerful vector representation such as Vector Graphics Complexes [23] may be a good alternative for the use of layers in some cases, as it allows for features such as sharing edges, and 3-way vertex split. Vector Graphics Complexes can also better incorporate the semantics of the drawing as opposed to plain vector graphics representations, such as SVGs. As a consequence, it could also be a great direction to improve the correspondence algorithm.

The method described by Xing et al. [28] is also an interesting direction to draw the key frames more efficiently, and possibly enhance the correspondence algorithms. Another direction to improve the correspondence solution is by employing an iterative method, alternating between the matching algorithm and the transformation computation. Another possibility is using the work of Noris et al. [38], where a drawing is segmented using scribbles. This idea could be incorporated in our work to define the layers in an alternative way, for example.

Finally, a user study would be very useful to further validate the contributions of the method and the usability of the system, and to guide new future research directions.

Acknowledgements

This research was partially supported by CNPq-Brazil. We thank Diogo Viegas and Leandro Araujo for testing the system and making animation samples.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.compfluid.2016.08.006

References

[1] Williams R. The Animator's Survival Kit. second. Faber & Faber; 2009.
[2] Parameter values for ultra-high definition television systems for production and international programme exchange. International Telecomunication Union; 2012. Recommendation ITU-R BT.2020.
[3] Catmull E. The problems of computer-assisted animation. SIGGRAPH Comput Graph 1978;12(3):348–53.

[4] Miura T, Iwata J, Tsuda J. An application of hybrid curve generation: cartoon animation by electronic computers. In: Proceedings of the AFIPS spring joint computing conference. In: AFIPS Conference proceedings, vol. 30. Washington D.C.: AFIPS / ACM / Thomson Book Company; 1967. p. 141–8.
[5] Burtnyk N, Wein M. Computer-Generated Key Frame Animation. J Soc Motion Pict Telev Eng 1971;80(3):149–53.
[6] Durand CX. The "toon" project: requirements for a computerized 2D animation system. Comput Graph 1991;15(2):285–93.
[7] Patterson J, Willis P. Computer-assisted animation: 2D or not 2D. Comput J 1995;37(10):829–39.
[8] Burtnyk N, Wein M. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. Commun ACM 1976;19(10):564–9.
[9] Reeves WT. Inbetweening for computer animation utilizing moving point constraints. SIGGRAPH Comput Graph 1981;15(3):263–9.
[10] Fiore FD, Schaeken P, Elens K, Reeth FV. Automatic inbetweening in computer assisted animation by exploiting 2.5D modelling techniques. In: Proceedings of the fourteenth conference on computer animation (CA2001); 2001. p. 192–200.
[11] Igarashi T, Moscovich T, Hughes JF. As-rigid-as-possible shape manipulation. ACM Trans Graph 2005;24(3):1134–41.
[12] Sýkora D, Dingliana J, Collins S. As-rigid-as-possible image registration for hand-drawn cartoon animations. In: Proceedings of the international symposium on non-photorealistic animation and rendering; 2009. p. 25–33.
[13] Sederberg TW, Greenwood E. A physically based approach to 2&ndash;d shape blending, vol. 26. New York, NY, USA: ACM; 1992. p. 25–34.
[14] Song Z, Yu J, Zhou C, Wang M. Automatic cartoon matching in computer-assisted animation production. Neurocomputing 2013;120(0):397–403. Image Feature Detection and Description.
[15] Xie M. Feature matching and affine transformation for 2D cell animation. Vis Comput 1995;11(8):419–28.
[16] Bregler C, Loeb L, Chuang E, Deshpande H. Turning to the masters: motion capturing cartoons. In: Proceedings of the 29th annual conference on computer graphics and interactive techniques. SIGGRAPH '02;. New York, NY, USA: ACM; 2002. p. 399–407.
[17] Melikhov K, Tian F, Soon SH, Chen Q, 0002 JQ. Frame skeleton based auto-inbetweening in computer assisted CEL animation. In: Proceedings of the IEEE international conference on , cyberworlds computer society; 2004. p. 216–23.
[18] de Juan CN, 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. SCA '06;. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association; 2006. p. 223–32.
[19] Noris G, Sýkora D, Coros S, Whited B, Simmons M, Hornung A, et al. Temporal noise control for sketchy animation. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on non-photorealistic animation and rendering. NPAR '11. New York, NY, USA: ACM; 2011. p. 93–8.
[20] Ben-Zvi N, Bento J, Mahler M, Hodgins J, Shamir A. Line-drawing video stylization. Comput Graph Forum 2016;35(6):18–32. doi:10.1111/cgf.12729.
[21] Yu J, Liu D, Tao D, Seah HS. Complex object correspondence construction in two-dimensional animation. IEEE Trans Image Process 2011;20(11):3257–69.
[22] Dalstein B, Ronfard R, van de Panne M. Vector graphics animation with time–varying topology. ACM Trans Graph 2015;34(4).
[23] Dalstein B, Ronfard R, van de Panne M. Vector graphics complexes. ACM Trans Graph 2014;33(4).
[24] Fekete J-D, Bizouarn E, Cournarie E, 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 SIGGRAPH '95. New York, NY, USA: ACM; 1995. p. 79–90.
[25] Reeth FV. Integrating 2 1/2-D computer animation techniques for supporting traditional animation. In: Proceedings of the computer animation CA '96. Washington, DC, USA: IEEE Computer Society; 1996. p. 118–25.
[26] Kort A. Computer aided inbetweening. In: Proceedings of the 2nd international symposium on non-photorealistic animation and rendering NPAR '02. New York, NY, USA: ACM; 2002. p. 125–32.
[27] Whited B, Noris G, Simmons M, Sumner R, Gross M, Rossignac J. Betweenit: an interactive tool for tight inbetweening. Comput Graph Forum (Proc Eurographics) 2010;29(2):605–14.
[28] Xing J, Wei L-Y, Shiratori T, Yatani K. Autocomplete hand-drawn animations. ACM Trans Graph 2015;34(6) 169:1–169:11.
[29] Frisken SF. Efficient curve fitting. J Graph Tools 2008;13(2):37–54.
[30] Schneider PJ. chap. an algorithm for automatically fitting digitized curves. In: Graphics Gems. San Diego, CA, USA: Academic Press Professional, Inc.; 1990. p. 612–26. ISBN 0-12-286169-5.
[31] Kuhn HW. The hungarian method for the assignment problem. Nav Res Logist Q 1955;2(1–2):83–97. doi:10.1002/nav.3800020109.
[32] Cohen-Or D, Greif C, Ju T, Mitra NJ, Shamir A, Sorkine-Hornung O, et al. A sampler of useful computational tools for applied geometry, computer graphics, and image processing. A K Peters/CRC Press; 2015.
[33] Lee S-Y, Chwa K-Y, Shin SY. Image metamorphosis using snakes and free-form deformations. In: Proceedings of the 22nd annual conference on computer graphics and interactive techniques. SIGGRAPH '95. New York, NY, USA: ACM; 1995. p. 439–48.
[34] Sederberg TW, Greenwood E. Shape blending of 2-D piecewise curves. In: Proceedings of the mathematical methods for curves and surfaces; 1995. p. 497–506.
[35] Cong G, Parvin B. A new regularized approach for contour morphing. In: Proceedings of the 2000 conference on computer vision and pattern recognition (CVPR 2000), 13–15 June 2000, Hilton Head, SC, USA. IEEE Computer Society; 2000. p. 1458–63.
[36] Fu H, Tai C, Au OK. Morphing with laplacian coordinates and spatial-temporal texture. In: Proceedings of the pacific graphics posters; 2005. p. 100–2.
[37] Whited B, Rossignac J. B-morphs between b-compatible curves in the plane. In: Proceedings of the SIAM/ACM joint conference on geometric and physical modeling. SPM '09. New York, NY, USA: ACM; 2009. p. 187–98.
[38] Noris G, Skora D, Shamir A, Coros S, Whited B, Simmons M, et al. Smart scribbles for sketch segmentation. Comput Graph Forum 2012;31(8):2516–27.