Computer Aided Inbetweening

Alexander Kort

Fraunhofer Institute for Applied Information Technology

Abstract

The production of inbetweens is a tedious task for animators and a complicated one for algorithms. In this paper, an algorithm for computer aided inbetweening and its integration in a pen-based graphical user interface are presented.
The algorithm is layer-based, assuming an invariant layering order. It is applicable to animations in a style similar to paper cut out, in which the drawings on the cut-out pieces are inbetweened as well.
The content of each key drawing is analysed and classified into strokes, chains of strokes and relations that hold among them. Rules decide what parts of different drawings may be matched. These rules specify allowed changes between relations in key drawings. A cost function based approach determines the correct matching of strokes. Generated animation paths between corresponding strokes determine the resulting inbetweens.
To hold for possible mismatchings and to allow for artistic control over the results, the inbetweening algorithm is embedded in a free form graphic user interface. Thus artists are enabled to focus on the part of the inbetweening task computers are not able to solve.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques; I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Object Recognition; I.2.10 [Artificial Intelligence]: Vision and Scene Understanding—Shape;

Keywords: keyframe animation, inbetweening

1 Introduction and Overview

Like in other fields of animation, the use of computers has entered the world of 2D cartoons animation production, but still mostly unchanged since the days of Snow White [Thomas and Johnston 1981] is the division between key drawings and inbetweens.

Drawing the inbetween frames is time-consuming and tedious. Automation of these steps would allow the artist to concentrate on the more creative work of drawing the key images. So each artist could become a key drawing animator and leave the boring work of drawing the inbetweens to the machine.

But inbetweens are not just interpolations between key drawings. When drawing the inbetweens, the inbetweener utilises

- her background knowledge of the physical rules of the world,
- her expert knowledge when to bend or ignore these rules and
- her idea what emotions should be evoked by the animation.

For these reasons, automatic inbetweening will probably remain an unreachable goal, at least for the near future.

The approach shown here does not aim at automatic, but at computer assisted inbetweening. Embedding an inbetweening algorithm into a graphical user interface enables fluent corrections of the results.

The proposed algorithm is not applicable to all kind of cel animations. It's restricted to those animations in which the cel content's layering order is invariant. This includes animations in a "cut-out"-style. It transcedes cut-out because contents of cut out pieces can change between different key cels, thus changing in the inbetweens as well. These changes are not limited to affine transformations. The "cutting out" of pieces and the correspondence detection between them is solved by the algorithm without user interaction required.

The chosen approach is based upon the following assumptions :

1. each drawing is made of *stroke chains*, structures consisting of one or more connected strokes. A stroke "is a single path specified by the movement of a pen" [Igarashi 1999].
   Stroke chains model the merging of connected strokes as well as the closing of unintended and recognition of occlusion-induced gaps. Occluded lines may appear in key drawings, but not necessary in both. Modelling this allows interrupted stroke chains to be recognized as such and processed accordingly in further matching steps.

2. a stroke chain in one key drawing may have a corresponding stroke chain in another key drawing. It does not need to have one.

3. the transition between stroke chains is modeled by animation paths. These animation paths indicate both the correspondence between stroke chains in key drawings and the spatial interpolations between them.

The proposed algorithm works upon vectorized strokes. It will address the tasks of

- *Identifying stroke chains* in given key drawings. The vectorized strokes in the key drawings are analysed and grouped to stroke chains based upon adjunctions and occlusions.

- *Detecting the correspondences* between stroke chains in key drawings, including point correspondence. The stroke chains in each drawing are analysed and classified into specific relations like inclusion, adjacency and layering. A rule based algorithm operates on these relations, detecting the allowed matchings between stroke chains in different drawings. These rules model changes between the different drawings.
  For allowed matchings, a cost function rates the animation paths. The cost function is based upon translation, curvature and bending. Animation paths are optimized with regard to this cost function.

- *Generating the inbetweens*, based on the resulting animation paths. The layering relations obtained in previous steps help to handle cases of occlusions.

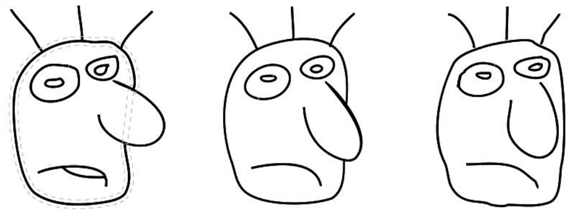


Figure 1: On the left and right are the key images. The stroke chain (highlighted on the left) and the detection of the layering order between nose and face contour allowed generating the inbetween in the middle.

This algorithm is embedded in an interactive free form drawing environment. The artist can draw strokes, which are vectorized immediately. Key drawings can be selected in various ways. After calculating inbetweens, the artist can correct animation paths manually and specify the desired timing. In one or more iterative steps of inbetweening and corrections the artist is assisted in producing images fulfilling her aesthetic criterias.

2 Related Work

Burtnyk and Wein [1976] introduced interactive skeleton techniques for enhancing motion dynamics in key frame animations. Stick figure representations of key frame components are animated manually, and the inbetweens are interpolated. Correspondence is established based upon the drawing order.
Catmull [1978] presented an analysis of computer-supported animation and the expected problems. Six different solution categories were presented, all related to the problem of information loss when going from 3D to 2D. According to this analysis, the method presented here is a combination of infering missing information, using the help of animators and restricting the class of animations that may be drawn.

An overview about the principles of animation in computer graphics is given by Lasseter [1987]. Some of the principles covered there like "squash-and-stretch" prevent an easy transfer of successful similarity detection methods from other application fields.

In a cartoon world, the geometry of a character depicted may depend on the camera's position. Artistic criterias are important, not geometric identity. This complicates further any use of estimated 3D models for 2D Inbetweening. Rademacher [1999] proposed a solution for viewer-dependent geometry for characters already modeled in three dimensions.

Durand [Durand 1991] listed the requirements for a computerized 2D animation system.

J.D. Fekete et al. [1995] presented an approach to the whole vectorized drawing process. He also covered the advantages and disadvantages of automatic inbetweening. The advantages are reduction of the number of hand-based-inbetweens and the possibility for procedural rendering, like texture mapping on inbetweened regions.

One disadvantage mentioned is that the automation changes the nature of inbetweening and limits it complexity. It would restrict the animation to fairly standard drawings. The approach presented here aims at recognizing changes that cannot be interpolated due to lack of knowledge. So the parts of drawings which are automatic inbetweenable are detected and interpolated, those which are not are left out for the artist to add.

He further states that tuning the inbetweens urges the animator to acquire "skills from [...] a computer graphist who can precisely manipulate Bezier curve handles". This is addressed by embedding the inbetween algorithm into a free from stroke environment, hiding things like Bezier control points from the animators view.

One part of the method proposed here is the blending of stroke chains, once their correspondence is established. Work on shape-blending was done by Sederberg and Greenwood [1992], who proposed a solution to the vertex correspondence problem between two-dimensional polygonal shapes. An approach to the vertex path problem was shown by Sederberg et al in [Sederberg et al. 1993].

D. Cohen-Or et al [1998] introduced a method for 3D distance-field metamorphosis, which was also applied on two-dimensional shapes. They relied on anchor points provided by an animator.

The proposed algorithm uses animation paths for spatial interpolation between the shape of the objects, animation paths are covered in [Gomes and Darsa 1999].

Madeira, Stork and Groß[1996] describe a region-based approach for automatic coloring. It is based on similarity between regions. Like in the method proposed here, they use heuristics based upon relations (e.g. inclusion) to guide the region matching. But their approach - which is extended in [Madeira 1999] - requires identity of relations, while this approach tries to model allowed changes between relations in key frames.

Xie [1995] followed a different approach, building intermediate images to generate inbetweens for animations. The inbetweens produced this way were restricted to results of affine transformations.

A field closely related to automatic inbetweening is automatic colouring. Automatic colouring algorithms can exploit that the images involved are less distinct than key drawings in inbetweening. Seah and Feng [2000] proposed a method based upon using both distance vector estimation and segmentation of images in regions.

Part of the work presented here is the user interface. I have chosen a freeform user interface, both with respect to the strokes drawn and the possible user-definition of timings and animation paths. Igarashi [1999] covered freeform user interfaces in depth. Following this approach, the user interface could be called *supportive*. In these "the user interacts with the target material directly without using GUI widgets" [Igarashi 2000], with the drawings on different sheets and the animation paths drawn between them as target material.

3 The Model

3.1 Stroke chains

A drawing consists of strokes. A stroke is an uninterrupted path of possible varying width. Strokes can form the outline of regions, can be thin elements like hair or can represent special cases of "degenerated" regions like a closed mouth in a face. Sometimes lines or borders are partially occluded by other regions and thus split into different strokes. Since these occlusions may not be repeated in the other key drawing, the inbetweening algorithm must consider this case.

A stroke chain is modeled as chain of one or more strokes. These strokes are functions

$$s : [0 \ldots 1] \rightarrow \Re^3$$

The first two dimensions are the x and y values on the drawing canvas. Following [Madeira et al. 1996], the third dimension is the width. Strokes s_i, s_{i+1} in a chain are connected ($s_i(1) = s_{i+1}(0)$).

All elements belong to one of the following two classes :

- *cubic bezier curves*, which are drawn by the artist and vectorized using [Schneider 1988] or are generated in a previous inbetweening step. They are visible.

- *line segments*, which are invisible. They connect strokes which are presumed to be part of a common shape's outline, but are interrupted by an occluding shape. (cf. figure 1).

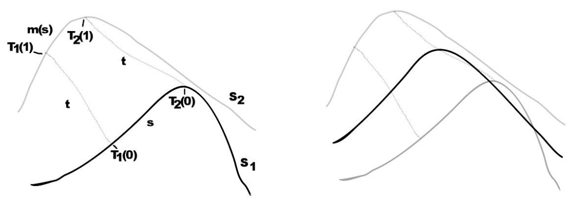


Figure 2: On the left side is an example for correspondence animation paths. Shown are two animation paths T_1 and T_2 between stroke chains S_1 and S_2.

Stroke chains of connected visible and invisible strokes are the building blocks both for the matching algorithm and the generation of inbetweens.

3.2 Animation paths

Animation paths $A_i : [0 \ldots 1] \to \Re^2$ model the correspondence of stroke chains.
A mapping function $m : [0 \ldots 1] \to [0 \ldots 1]$ specifies the correspondence of points on these chains. So the path from $S_1(s)$ on the stroke chain S_1 ends at point $S_2(m(s))$ on stroke chain S_2. Together A_i and m define the location of the morphed points on the inbetweens.

Let $t \in [0 \ldots 1]$ be the time and $s \in [0 \ldots 1]$ be the scalar value of the point on the first stroke to be morphed. Then the corresponding interpolated point is given by

$$\begin{aligned} I(s,t) &= (1-t)S_1(s) + tS_2(m(s)) + (1-s)A_i(t) + sA_{i+1}(t) \\ &\quad - (1-s)(1-t)A_i(0) - (1-s)tA_i(1) \\ &\quad - (1-t)sA_{i+1}(0) - stA_{i+1}(1) \end{aligned}$$

This is based upon the Coons transformation (cf. [Gomes and Darsa 1999]). It is applied on two stroke chains S_1 and S_2 with neighboring animation paths A_i and A_{i+1} (see figure 2).

4 The user interface

4.1 The Workflow

All functionality is available via the graphical user interface depicted in figure 3. The user can draw upon the drawing area, assisted by an "onion skinning" feature. Cel levels can be loaded and

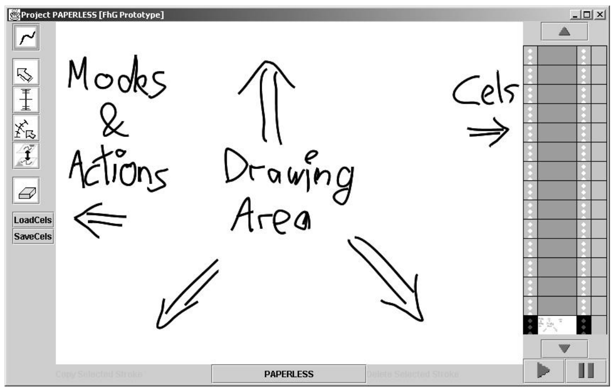


Figure 3: The user interface in which the inbetweening algorithm is embedded. On the right side are the cels thumbnails.

saved. Cels are displayed on a thumbnail display. The GUI offers further functionality for moving, copying, deleting and grouping of stroke chains.

The user can launch the inbetweening algorithm by selecting two key drawings on the thumbnail display. The inbetweens are then generated and displayed. The artist may not be satisfied with the results for the following reasons :

- The wrong stroke chains are matched to each other
- The point correspondence is wrong, though the stroke chain correspondence is right.
- Point correspondence is good, but the proposed animation path is not the one the artist wants
- The timing on the otherwise good animation path is wrong

In all these cases, the results can be removed and/or overridden. This is done via animation paths.

The user can remove animation paths. He can also draw new animation paths between stroke chains. These new animation paths may contradict existing point correspondence (e.g. by crossing with existing paths between the same involved stroke chains). In this case the older animation path is removed. The new path may even contradict existing stroke correspondences. Then all other paths which start or end at one of the involved stroke chains are removed.

After removing or rendering an animation path, new inbetweens are generated. If the last animation path between two stroke chains was removed, the interpolation of these stroke chains disappears from the inbetweens.

After each change, it is possible to restart the matching algorithm. Then the algorithm keeps the existing assignments and tries to match only the strokes still unassigned in the key frames. It is possible to start with drawing a correspondence animation path before the first inbetweening. So matching and interactive correction can alternate quickly in the workflow.

4.2 Timing

Timing defines at what time a part of the drawing is at a certain place on the cel. Since according to [Whitaker and Halas 1981] timing is "the part of the animation which gives *meaning* to the movement", the artist can change the timing interactive. The proposed default timing is a "slow-in-slow-out".
In the graphical user interface, handles orthogonal to the animation path (cf. figure 4) are used to display the timing aspects. They are centered at the corresponding point of the inbetweened stroke. By grabbing a handle, the user can move the corresponding stroke across the animation path. It is also possible to choose between affecting all the handles or just the one selected. All changes in the inbetweens are immediately rerendered and displayed.

5 The matching algorithm

The matching approach is based on

- analysing the key drawings with respect to their constituents (the stroke chains) and the relations they are involved in
- a set of rules, regulating which stroke in one drawing can be matched to which stroke(s) in the other drawing
- a cost function measuring the quality of a pairwise matching of stroke chains.

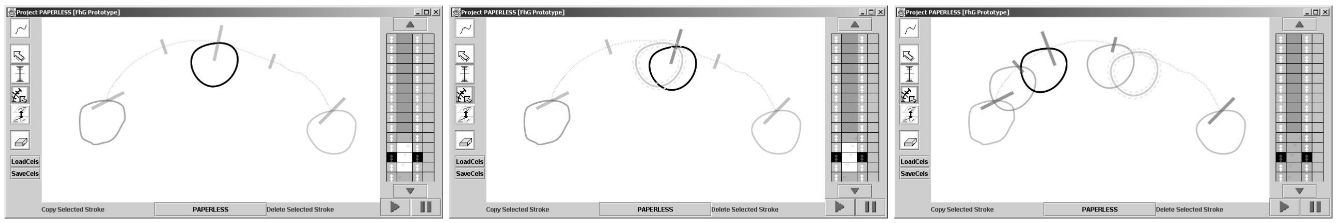


Figure 4: Changing the timing. On the left is the initial configuration with slow-in-slow-out movement. In the middle one handle was used to change the inbetweened ball. At the right is a sequence where all inbetweens are dragged together with the grabbed handle.

- a scheme to find the best allowed assignments between stroke chains in different drawings. The goal is to minimize the total sum of the cost of the stroke chain matchings. Therefore an optimization algorithm is used

The inbetweened stroke chains are generated using the best assignments.

5.1 Grouping the stroke chains

In both key drawings, stroke chains are built automatically. Selecting a drawing as key drawing starts a grouping algorithm on it. It follows the following heuristics

- don't join strokes at ends which have a longer distance than given maximum distance
- if three or more strokes join at an end within the maximum distance, merge those who have the least change in angle direction
- if two strokes end at the same stroke once, and each does so at a t-junction, join them with a line segment.
- if one stroke has with both his start and end t-junctions adjacencies to the same other stroke, close it with a line segment.

An efficient problem method for resolving ambiguities in grouping is not within the scope of this paper. In experiments a set of different possible groupings was generated by taking both a joined and a not-joined version into account and further extend both of them. The matching algorithm was applied to each element of this set of groupings. The best result was chosen.

5.2 Building a stroke chain graph

Generating a matching algorithm based on curve similarity alone is a difficult approach in the domain of cartoon animation. Curves change between frames, and these changes have a meaning. But while curve similarity alone is not sufficient, assuming equal topology of the stroke chains for matching is too restrictive. Changes in relations between stroke chains may happen, so relying on equivalence of relationships in different key frames would prohibit a lot of the dynamics appreciated in animation.

This algorithm follows the approach of choosing between acceptable and unacceptable changes in relations. So relationships may change between different drawings. This enlarges the class of possible inbetweenings compared to the affine [Xie 1995] or the topological identical [Madeira 1999]. By defining the allowed changes between relations, it is possible to describe the possible transitions between key drawings. So the first step is to identify what relations hold, the second step is classifying the allowed changes.

Therefore, a *stroke chain graph*

$$G\left(S=\{s_1,...,s_n\},U_1,\ldots,U_l,B_1\ldots B_m\right)$$

is built for each key drawing. The vertices of the graph are the stroke chains $\{s_1,...,s_n\}$, the edges are unary $U_i \subseteq S$ and binary $B_i \subseteq S^2$ relations between them.

Some relations ($endsAt(a,b)$, $closed(a)$, $startEndInside(a,b)$, $included(a,b)$, $isolatedEnd(a)$ and $isolatedStart(a)$) are based upon properties of the stroke chains.

They are found by testing for inclusion of stroke chains and adjacency tests at stroke chains' start and end. Inclusion tests are done with fast intersection tests of cubic splines with horizontal lines.

Other relations are defined by user interactions, in this case grouping stroke chains together $shareGroup(a,b)$.

Finally relations are also derived from other relations :

$$\begin{aligned}
contour(a) &\Leftrightarrow \nexists b \in S \quad included(a,b)\\
isolated(a) &\Leftrightarrow isolatedEnd(a) \wedge isolatedStart(a)\\
dangling(a) &\Leftrightarrow isolatedStart(a) \oplus isolatedEnd(a)\\
above(a,b) &\Leftrightarrow endsAt(b,a) \wedge \neg dangling(b)\\
leaving(a,b) &\Leftrightarrow startEndInside(a,b) \wedge above(a,b)
\end{aligned}$$

5.3 Determining possible assignments

Each key drawing is described by a stroke chain graph. Allowed assignments between stroke chains of different stroke chain graphs are determined using a set of rules. Each rule describes a forbidden assignment between stroke chains.

The stroke chains are outlines of three-dimensional volumes. Rules determine the possible correspondences between the outlines of these volumes. Every rule introduced forbids a set of possible transitions. Some rules forbid transitions that are covering unlikely special cases (like an open contour becoming a closed contour), while the others are motivated by lack of three-dimensional information neccessary for the generation of satisfactory inbetweens.

Given two stroke chain graphs G_1 and G_2, the rules used describe forbidden assignments involving either two stroke chains $a \in G_1, x \in G_2$ or four stroke chains $a,b \in G_1$ and $x,y \in G_2$. They forbid assignments

- from a to x : $\boxed{\neg(a \rightarrow x)}$
- from a to x and b to y at the same time : $\boxed{\neg(a \rightarrow x, b \rightarrow y)}$

The following rules are applied : *Do not*

- invert the layering order [1]

$$above(a,b) \wedge above(x,y) \Rightarrow \neg(a \rightarrow y, b \rightarrow x)$$

[1]In general, without 3D-information it will be futile to generate an acceptable inbetween of this

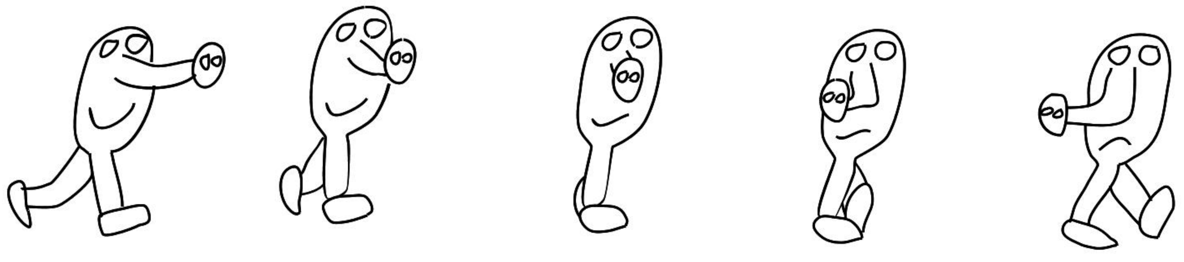


Figure 5: On the left and right are the two key drawings. In the middle are three inbetweenings.

- invert the inclusion order

$$included(a,b) \wedge included(x,y) \Rightarrow \neg(a \to y, b \to x)$$

- match open contour stroke chains to closed stroke chains

$$\neg closed(a) \wedge contour(a) \wedge closed(x) \Rightarrow \neg(a \to x)$$

- match open dangling stroke chains to closed stroke chains

$$\neg closed(a) \wedge dangling(a) \wedge closed(x) \Rightarrow \neg(a \to x)$$

- match closed contour stroke chains to open stroke chains

$$closed(a) \wedge contour(a) \wedge \neg closed(x) \Rightarrow \neg(a \to x)$$

- match isolated stroke chains to stroke chains being adjacent with both ends to another stroke

$$\begin{array}{rl} & isolated(a) \wedge \neg closed(a) \wedge \neg closed(x) \\ \wedge & \neg isolatedEnd(x) \wedge \neg isolatedStart(x) \end{array} \Rightarrow \neg(a \to x)$$

- invert leaving relationships

$$leaving(a,b) \wedge leaving(x,y) \Rightarrow \neg(a \to y, b \to x)$$

- leave a user-defined group

$$shareGroup(a,b) \wedge \neg shareGroup(x,y) \Rightarrow \neg(a \to y, b \to x)$$

These are rules forbidding specific mappings between two strokes. So if there is no mapping available for a specific stroke, it remains as an unassigned stroke in the matching.

Some experiments were done on relations based on geometric order relations like $WestOf$ or $NorthOf$. These were measured relative to the coordinate system defined by the major axis of the including stroke chain (or the canvas coordinate system) and its orthogonal. But due to squash-and-stretch [Lasseter 1987], the estimation of the coordinate system was too unreliable.

5.4 Finding the best animation paths

Configurations model the animation paths between two stroke chains (see 5.4.1). The quality of single configurations is measured with a cost function (see 5.4.2). To find the lowest-cost matching, different configurations are initialised (see 5.4.3) and optimized (see 5.4.4). The best configuration is chosen as the result describing the best possible matching between these two stroke chains.

5.4.1 The Configuration

Each stroke chain has a set of potential matching positions. These sets

$$\begin{array}{rcll} A &=& \{\alpha_1,..\alpha_n\} & \alpha_i \in [0\dots1] \\ B &=& \{\beta_1,..\beta_n\} & \beta_i \in [0\dots1] \end{array}$$

are used as position markers on the stroke chains in the two key drawings. Position α_i on one stroke corresponds to position β_i on the other stroke.

Both sets are monotone. The drawing direction of stroke chains can not assumed to be the same. The correct direction is determined using the cost function. So while A is assumed to be monotonously increasing ($\forall i : \alpha_i < \alpha_{i+1}$), B can be either de- or increasing ($\forall i : \beta_i < \beta_{i+1}$ or $\forall i : \beta_i > \beta_{i+1}$).
If closed stroke chains are involved, the starting points are not immediately evident. While $\beta_1 = 0 \wedge \beta_n = 1$ resp. $\beta_1 = 1 \wedge \beta_n = 0$ still holds, the offset $\alpha_1 \in [0\dots1]$ determines the relative shift of the stroke chains. In these case the monotonicity assumption can be violated for one pair ($\exists! j \alpha_j > \alpha_{j+1}$). As in determination of the drawing direction, an evaluation of the cost function with different offsets determines the correct one.

The mapping is modeled as

$$m(t) = \beta_i + \frac{t-\alpha_i}{\alpha_{i+1}-\alpha_i}\left(\beta_{i+1}-\beta_i\right)$$

with i chosen so that $t \in \left[\alpha_i, \alpha_{i+1}\right]$ holds.

The algorithm for finding the animation paths between stroke chains models the animation paths between the corresponding points α_i and β_i as quadratic splines. Therefore, n controlpoints $\vec{c}_i \in \Re^2$ are introduced. With stroke chains $\vec{S}_A$ and $\vec{S}_B$ the animation paths are

$$A_i(u) = (1-u)^2 \vec{S}_A(\alpha_i) + 2(1-u)u\vec{c}_i + u^2 \vec{S}_B(\beta_i)$$

A configuration for the animation paths between two stroke chains is modeled as

$$X = \left\{\left(\alpha_i, \vec{c}_i, \beta_i\right) | 1 \le i \le n\right\}$$

5.4.2 Cost function

To choose the best assignment, a *cost function* is used. It should measure the goodness of a configuration for animation paths between two stroke chains.

The chosen cost function [2] takes into account

[2]This paper focuses on the rules governing the dynamics of inbetweening. Any cost function measuring the goodness of fit between two stroke chains can be used with the method presented here. Further work will be done on more elaborated cost functions.

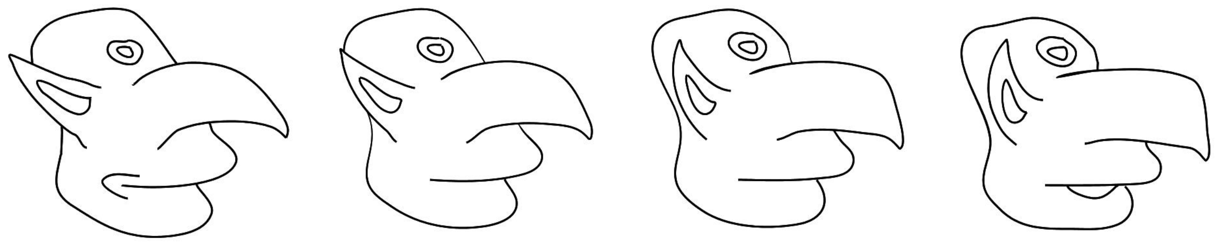


Figure 6: A successful inbetweening with the key drawings at the left and right.

- the translation, relative to the enclosing stroke chain's or canvas' centroid
- the local curvature
- the distance between neighboring points

These variables are sampled in a regular interval both on the key stroke chains and on an intermediate interpolated stroke chain. This stroke chain is generated using the given configuration. Nonmonotonicity across time is punished both for curvature and distance between neighboring sampled points. The total cost is realised as a weighted sum of these influences. The weights are chosen so that the translation is the most important contributor, with the other variables being equally weighted.

5.4.3 Initial configuration

Initial animation paths are generated. The α_i and β_i are chosen initially based upon the stroke chains curvatures extremas. For each extrema on one of the two stroke, an $(\alpha_i, \vec{c}_i, \beta_i)$ is added to the configuration. If there is an extrema on the first stroke, α_i marks its position, and β_i is the position mapped on the second stroke. The inverse holds for an extrema β_i on the second stroke. There is no matching between the extremas yet, mapping of extremas to each other occurs during the later optimization step.

Configurations are generated both for B monotonously decreasing and increasing. In case of closed stroke chains involved, several of these configuration are created with different offsets for α_i. This takes into account the possible relative shift.

5.4.4 Optimizing the configurations

All initial configurations are improved using the metropolis algorithm [Press et al. 1988]. All parameters in the configuration are changeable. Considering the relative small size of the configurations, this is fast enough for interactive behaviour. The lowest cost configuration is selected as the matching result.

The number of involved animation paths is reduced afterwards. If one animation path is a combination of its two neighbors , it is removed as redundant.

5.5 Finding the best matching between stroke chain graphs

The best overall matching is found by building up an assignment tree between the stroke chains $\{a_1, \ldots, a_m\}$ and $\{b_1, \ldots, b_n\}$ of the stroke chain graphs G_1 and G_2. In this tree, every node corresponds to

- a matching between two stroke chains $a_i \to b_j$
- an unassigned matching $a_i \to 0$ or
- a surplus matching $0 \to b_j$.

The nodes are attributed with the cost of the matching. In each path from a leaf to the root node, each stroke can appear only once. No forbidden mappings are allowed in any path from leaf to root node. Nonassignment of stroke chains is punished with high costs, but it is explored whenever possible. So even when a match would be possible, the variant with the involved stroke being unassigned is tested as well.

The complete tree can theoretically become huge. To avoid building up complete trees, pruning techniques are used. The algorithms expands the most promising node first. It also uses cost look-a-head, integrating knowledge about upcoming forbidden mappings.
Finally one or more complete paths in this tree remain that involve all stroke chains in the two key drawings. The path with the least cost is the chosen matching.

5.6 Generating the inbetweenings

Stroke chains are rendered using the definition of $I(s,t)$ (see 3.2). The value of t is determined by the timing of the inbetween. With fixed t, coordinate and width values are sampled for monotonous increasing s. The same algorithm used for vectorizing the drawn strokes [Schneider 1988] is used on these sampled points, resulting in vectorized inbetweened strokes.

Part of the stroke chain graph are the above relationships determining the layering order. If intersections exist in the inbetweens, they are used to determine the correct occlusions. So only the visible parts of the inbetweened stroke chains are rendered.

5.7 Summary of the algorithm

The matching algorithm proceeds as follows

1. analyse the key frames and group their content into stroke chain graphs
2. determine forbidden mappings for each key frame
3. build an assignment tree for the allowed mappings only, calculating the cost for each mapping
4. select the best path in the assignment tree
5. generate the inbetweens

6 Results

To test both the algorithm and its embedding into a graphical user interface, an experimental prototype in Java was developed. This software is used together with the Wacom-PL series of active matrix displays. It runs as a stand-alone Java application as well as

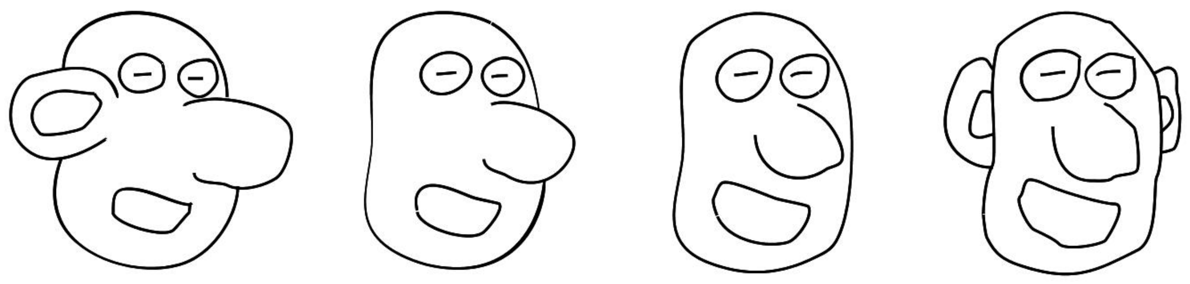


Figure 7: An example where the algorithms reaches its limits. Correspondences are detected correctly, but an interpolation of the ear is not within the strict two-dimensional scope of the algorithm and therefore omitted.

in conjunction with a drawing software realised by Digital Video, Rome.

Evaluation took place by the user partners in the PAPERLESS project, SBP (Rome), Varga (London) and Neptuno (Barcelona). An early prototype was shown to the public at the MIFA 2001. During evaluation, about 50 pairs of keyframes, most of low image complexity, were generated and inbetweened.

The graphical user interface was evaluated as useful and innovative, both from the user partners and the visitors at MIFA. Users appreciated not having to care about defining object hierarchies and changing control points. Stroke matching was also mostly accepted, though in complex images speed problems might prohibit a true interactive workflow. The quality of some resulting inbetweened strokes leaves room for improvements.

The character in figure 5 was generated with correct assignments between the stroke chains. The body contour needed some manual adjustment (one animation path added and one removed). Looking at the hidden leg and the shoe (esp. in the last inbetween) shows suboptimal shape-blending. Nonetheless, the correct layering order was established, allowing for automatic determination of the hidden lines of leg and shoe.

Part of this problems are eased by the interactive environment allowing for fast correction or rejection of the dissatisfying parts. In figure 8 all matchings were correct, but the matching between the body contours went wrong. One animation path overriding (shown in figure 9) suffices to get the results visible in figure 6. The layering order between ear and face contour allowed correct inbetweening. The application of the rule “do not match isolated stroke chain to adjacent stroke chain” prevented matching the wrinkle around the mouth in the first key frame to the open mouth in the second key frame.

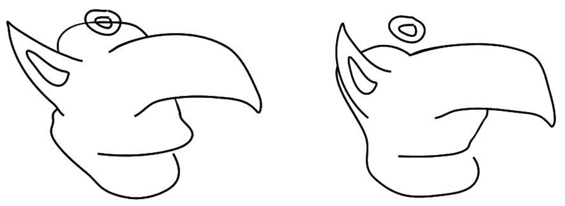
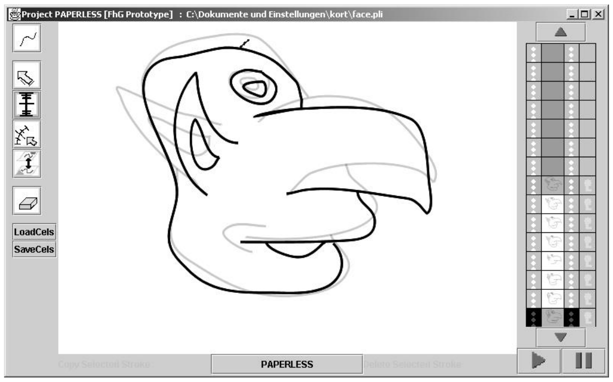


Figure 8: Two inbetweens with correct matchings of stroke chains, but wrong mapping between the stroke chains forming the face’s boundary. Key frames are the same as in figure 6


Project PAPERLESS [FhG Prototype] : C:\Dokumente und Einstellungen\kort\face.pli
LoadCels
SaveCels
PAPERLESS
Figure 9: Overriding a wrong assignment with one animation path

7 Conclusion and further work

The approach presented here is an environment enabling the artist to change seamlessly between interactive editing and inbetweening. The integration of hidden lines allows for inbetweening of more dynamic scenes. The use of flexible rules allows e.g. changes in topology and makes inbetweening possible for a larger class of images than before.

Due to the reasons sketched before, inbetweening algorithms will probably never be perfect. Embedding them into a free form graphical user interface allows the artist to call them when they are needed and to use, change or reject the results.

Still there are many ways to improve and extend the work done so far.

To improve the quality of the interpolated strokes in 2D, we may extend the cost function with additional criterias. For comparison an established algorithm like [Sederberg and Greenwood 1992], [Sederberg et al. 1993] or [Cohen-Or et al. 1998] can be integrated.

We observed that professional users draw loose or duplicate strokes. We would like to extend our approach to encompass these “loose and sketchy” [Curtis 1998] key drawings, maybe even keeping this style in the resulting inbetweens.

So far only two key drawings are involved in one matching. Extending the algorithm to more key drawings would allow the transfer of information about forbidden mappings among sets of key frames. This would further narrow the number of allowed mappings.

Probably the most interesting task is the integration of three-dimensional knowledge in the matching and inbetweening process. So far animations are restricted to a layered two-dimensional world with a strict order. Estimating a complete 3D model would allow using three-dimensional keyframe interpolation methods like

[Nebel 1999] . But this estimation would possibly even for simple characters – due to changes in pose, shape (squash-and-stretch) and viewer-dependent geometry [Rademacher 1999] – require a prohibitive high number of drawings as input.

But even if a complete three-dimensional reconstruction is not feasible, a three-dimensional shape could be approximated (like in [Williams 1991]). Combining this approximated shapes from several key frames, possibly including information from "virtual" modelsheets, could be used in rules working in a 3D-world and for animation paths modeled as projections of 3D movements. This would enlarge the class of possible inbetweenable animations, e.g. allowing a complete inbetwening of figure 7.

8 Acknowledgements

This work was done in the context of the European Union Project PAPERLESS - Tools for Paperless animation (IST-12329). I thank my colleagues and partners for valuable ideas and feedback. I'd especially like to thank the user partners for introducing me into the world of animation.

References

BURTNYK, N., AND WEIN, M. 1976. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *Communications of the ACM 19*, 10, 564–569.

CATMULL, E. 1978. The problems of computer-assisted animation. In *Computer Graphics (Proceedings of ACM SIGGRAPH 78), 12(3)*, ACM, 348–353.

COHEN-OR, D., LEVIN, D., AND SOLOMOVICI, A. 1998. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics 17*, 2, 116–141.

CURTIS, C. 1998. Loose and sketchy animation. In *ACM SIGGRAPH 98 Conference Abstracts and Applications*, ACM Press / ACM SIGGRAPH, New York, E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 317.

DURAND, C. X. 1991. The "toon" project : Requirements for a computerized 2d animation system. *Computers & graphics 15*, 2, 285–293.

FEKETE, J. D., BIZOUARN, E., COURNAIRE, E., GALAS, T., AND TAILLEFER, F. 1995. Tictactoon: A paperless system for professional 2d animation. In *Proceedings of ACM SIGGRAPH 95*, ACM Press / ACM SIGGRAPH, New York, E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 79–90.

GOMES, J., AND DARSA, L. 1999. *Warping and morphing of graphical objects*. Morgan Kaufmann Publishers, Inc., San Francisco.

IGARASHI, T. 1999. *Freeform User Interfaces for Graphical Computing*. PhD thesis, University of Tokyo.

IGARASHI, T. 2000. Supportive interfaces for creative visual thinking. In *Collective Creativity Workshop*. May 7-8, Nara (Japan).

LASSETER, J. 1987. Principles of traditional animation applied to 3d computer animation. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87), 21(4)*, ACM, 35–44.

MADEIRA, J., STORK, A., AND GROSS, M. H. 1996. An approach to computer-supported cartooning. *The Visual Computer 12*, 1–17.

MADEIRA, J. 1999. *A new region matching-based approach to computer-assisted cartooning*. PhD thesis, TU Darmstadt, Aachen.

NEBEL, J.-C. 1999. Keyframe animation of articulated figures using autocollision-free interpolation. In *17th Eurographics UK Conference'99*. April 13-15, Cambridge, UK.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 1988. *Numerical Recipes in C*, second ed. Cambridge University Press, Cambridge.

RADEMACHER, P. 1999. View-dependent geometry. In *Proceedings of ACM SIGGRAPH 99*, ACM Press / ACM SIGGRAPH, New York, E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 439–446.

SCHNEIDER, P. J. 1988. *Phoenix : An interactive curve design system based on the automatic fitting of hand-sketched curves*. Master's thesis, University of Washington.

SEAH, H. S., AND FENG, T. 2000. Computer-assisted coloring by matching line drawings. *The Visual Computer 16*, 269–304.

SEDERBERG, T. W., AND GREENWOOD, E. 1992. A physically based approach to 2d shape blending. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87), 21(4)*, ACM, 25–34.

SEDERBERG, T. W., PEISHENG, G., AND GUOJIN, W. 1993. 2d-shape blending : an intrinsic solution to the vertex path problem. In *Proceedings of ACM SIGGRAPH 93*, ACM Press / ACM SIGGRAPH, New York, E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 15–18.

THOMAS, F., AND JOHNSTON, O. 1981. *Disney animation : the illusion of life*. Abbeville Press, New York.

WHITAKER, H., AND HALAS, J. 1981. *Timing for animation*. Focal Press, Oxford.

WILLIAMS, L. 1991. Shading in two dimensions. In *Graphics Interface '91*, Morgan-Kaufman Publishers, Canadian Human-Computer Communications Society, 143–151.

XIE, M. 1995. Feature matching and affine transformation for 2d cel animation. *The Visual Computer 12*, 419–428.