EUROGRAPHICS 2010 / T. Akenine-Möller and M. Zwicker
(Guest Editors)

Volume 29 (2010), Number 2

BetweenIT: An Interactive Tool for Tight Inbetweening

Brian Whited[1,2] Gioacchino Noris[3] Maryann Simmons[1] Robert W. Sumner[4] Markus Gross[3,4] Jarek Rossignac[2]

[1]Walt Disney Animation Studios [2]Georgia Tech [3]ETH Zürich [4]Disney Research Zurich

Abstract

The generation of inbetween frames that interpolate a given set of key frames is a major component in the production of a 2D feature animation. Our objective is to considerably reduce the cost of the inbetweening phase by offering an intuitive and effective interactive environment that automates inbetweening when possible while allowing the artist to guide, complement, or override the results. Tight *inbetweens, which interpolate similar key frames, are particularly time-consuming and tedious to draw. Therefore, we focus on automating these high-precision and expensive portions of the process. We have designed a set of user-guided semi-automatic techniques that fit well with current practice and minimize the number of required artist-gestures. We present a novel technique for stroke interpolation from only two keys which combines a stroke motion constructed from logarithmic spiral vertex trajectories with a stroke deformation based on curvature averaging and twisting warps. We discuss our system in the context of a feature animation production environment and evaluate our approach with real production data.*

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Computational Geometry—Geometric algorithms Computer Graphics [I.3.6]: Methodology and Techniques—Interaction techniques

1. Introduction

Inbetweening is a cornerstone of the 2D animation pipeline. An animator produces drawings at *key* frames that capture the heart of the animated motion. The *inbetweener* then draws the frames between each pair of keys that fill in the intermediate motion. A typical feature-length animation requires over one million inbetween drawings, with ratios of 3-4 inbetweens for each key [JT95].

The manual production of inbetween frames is both difficult and time-consuming, requiring many hours of intensive labor by skilled professionals. Furthermore, its cost is a significant part of the total production budget. Consequently, researchers have long sought an automatic solution. Despite a large body of work in this area spanning over four decades, no definitive solution exists. This is in part because the problem is ill-posed—there is no concise set of rules for producing high-quality inbetweens, especially in cases of complex motion and changing occlusion. The artist relies on his or her perspective of the 3D world in which the animated characters are embedded, as well as on an artistic vision for achieving the desired aesthetics. It would not be sufficient to derive the three dimensional world from the drawings, even if it were possible, as most often the characters have their own, undefined, laws of physics and deformation.

We approach the challenging task of automatic inbetweening with two important observations derived from studying the traditional animation process. First, *tight* inbetweens, those drawn between two key frames that are very similar in shape, are among the most laborious and time consuming to produce. These inbetweens are tedious to draw because they require the greatest amount of technical precision and the least amount of artistic interpretation. By focusing on tight inbetweening, we pinpoint an expensive portion of the inbetweening process while restricting the problem to one that is more tractable from an algorithmic standpoint. Our second observation is that an effective system should automate as much as possible while effortlessly deferring to artist control whenever needed. We never take the artists out of the loop. Instead we seek to make them more productive by automating the tedious and time consuming tasks so they can focus their efforts on areas where their creative talents and expertise are required.

Based on these observations, we have developed *BetweenIT*, an interactive environment that automates tight inbetweening when possible while allowing the artist to guide,

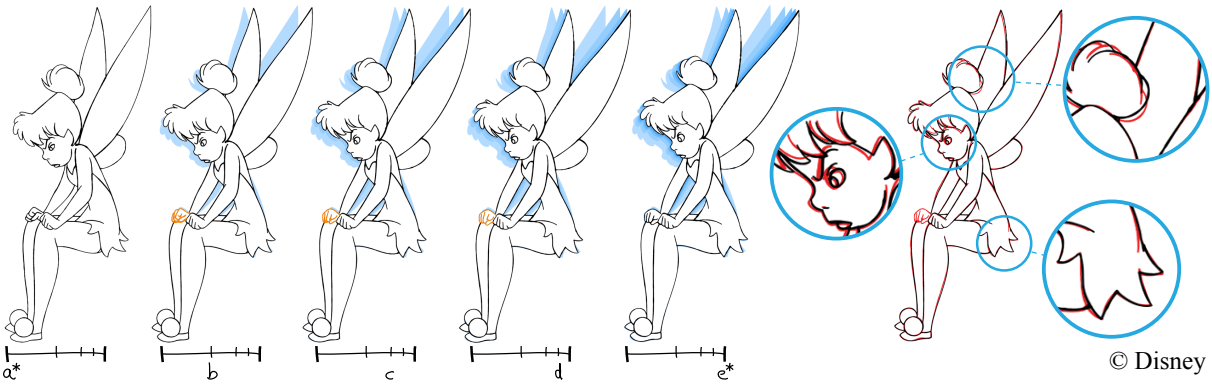

Figure 1: *Tinker Bell: Frames (a) and (e) are the given key frames and (b-d) (except for the right hand) were generated automatically. The "shadows" of previous frames are used to visualize their evolution. The topology of the right hand is not compatible across the two keys and therefore not suitable for automation: the portion indicated in red was hand-drawn. Frame (c) is overlaid on top of the original hand drawn inbetween (red) on the right.*

complement, or override the results. BetweenIT operates on vectorized pairs of consecutive key frames that were drawn in the program or obtained from scanned drawings. These key frames are segmented automatically into strokes. The strokes of one key frame are then matched with corresponding strokes of the subsequent key frame. Finally, interpolated strokes are generated for each requested inbetween frame. Supported user editing options include: adjusting vertex trajectories across frames; resolving ambiguities in the correspondence; and controlling the behavior of occluded strokes. All interactions can be specified by the artist graphically via a drawing interface.

2. Background

While some of the underlying mechanisms have benefited from the advent of digital technology, a 2D production today follows much the same basic workflow as traditional animation [JT95]. The process begins with a storyboard, which provides a visual representation of the story. In layout, the staging for each scene is designed, including establishing the setting, choosing and placing character and prop elements, and specifying camera motion and cuts.

Character and effects animation is then done in multiple stages. First, animators produce the subset of drawings that lay down the core of the action, the "keys." These extreme drawings are often "ruff" versions that capture the spirit, flow, and arcs of the animation. A "clean up" artist is responsible for taking the ruff drawings and producing clean lines that remain true to the original intent.

Each key drawing has one or more *timing charts* associated with it (e.g. see Figure 1). The animator uses these charts to specify how many drawings should be produced between keys, and at what intervals. It is the job of the inbetweening artist to draw the requested intermediate drawings to produce seamless motion.

When inbetweens are needed for dramatically different key drawings, advanced artistic skill and interpretation are required to produce satisfactory intermediate frames. In these cases, a further "breakdown" might be done, where the artist produces those frames within the range that present special drawing problems. *Tight* inbetweens require less artistic interpretation – but considerable technical skill to lay down the lines accurately.

The inbetweened frames represent a significant portion of the drawings, budget, and time for a production. A typical feature animation averages about four drawings per frame (for different characters, props, etc.). An 80 minute feature with 24 frames per second requires 460,800 drawings per production. If a quarter of these frames are done by the animators, then that leaves 345,600 inbetweened drawings. Since multiple drawings are often produced before arriving at the final version, the final tally can add up to over a million inbetweened drawings per full length production [JT95].

One critical consideration during animation and inbetweening is arcs of motion. Natural motion always follows an arc of some sort, and therefore to achieve fluid and lifelike animation, artists must capture these natural arcs. This is one of the most challenging aspects of inbetweening since making a drawing on an arc is much more difficult than one placed linearly between the keys [JT95].

Our approach focuses on automating the process of inbetweening animated frames post clean-up for characters, props, and effects. We introduce a novel interpolation scheme which automatically derives natural arcs of motion from pairs of consecutive keys. Our system fits into the current 2D pipeline with minimal change.

2.1. Related work

The problem of automatic 2D inbetweening dates back more than forty years to the inception of computer graphics as a field of research [MIT67]. To date, it has remained unsolved. The many challenges in automatic inbetweening include the

information loss and ambiguity inherent in a 2D projection of a 3D character, and the topology variations between key frames that result from changes in occlusion. An automated approach must address several challenges comprising: establishing correspondence in the presence of occlusions and topological changes, computing stroke trajectories through time, and avoiding the unnatural distortion that often occurs when there is a rotational component in an object's movement.

Many of the early methods are stroke-based. They require the user to identify a correspondence between the strokes of consecutive key frames (e.g., [MIT67, BW71, Lev77, KBB82, Dur91]) and do not handle occlusions or topological changes. Correspondences in Reeves's method [Ree81] are indicated by a collection of curves called moving points that are sketched by the user and connect the key frames to specify both trajectory and dynamics. A cubic metric space blending algorithm is used to find the positions of the remaining points, but requires heuristics to complete the patch network defined by the key frame strokes and moving point trajectories. Our method employs a semi-automatic correspondence algorithm that infers correspondence for the entire key from a small number of user gestures. Interpolation is automatic and designed to follow natural-looking motion arcs. However, our method supports an optional trajectory redrawing interface similar to Reeves's moving points that allows customized trajectories to be specified by the artist.

More recently, Baxter and Ichi [BiA06] describe a system for computing an N-way interpolation of several input keyframes at the stroke level. Correspondences are computed automatically and editable by the user when mismatches occur. This work focuses on calculating appropriate blending weights for the N-way interpolation. Stroke pairs are interpolated independently, whereas our technique utilizes a graph structure, which preserves connectivity and smoothness between adjacent strokes.

Kort [Kor02] presents a user-guided inbetweening system that identifies correspondences and computes inbetweens automatically but allows the user to correct undesired correspondences, trajectories, or timing. The method is restricted to the class of animations in which occlusions are resolved via an invariant layering. Our method is not restricted to layers and our workflow allows occlusion ambiguities to be resolved with help from the artist via a simple and efficient user interface.

Another layer-based approach by de Juan and Bodenheimer [dJB05] targets the reuse of previously created 2D animations. Unlike the previously discussed methods, their system is image-based, rather than stroke-based. Characters in completed animation frames are segmented from the background and divided into layers. Inbetweening is accomplished via radial-basis function (RBF) interpolation of the layer contours, followed by morphing of the interior texture. The RBF interpolation requires the shapes to be properly aligned, and artifacts can result from misalignments. The MeshIK system of Sumner and colleagues [SZGP05] includes a boundary-based interpolation scheme that does not require alignment, but it does not address interior texture morphing. As shown by Baxter, Barla, and Anjyo [BBiA09a], compatible embedding enables a maximally rigid interpolation [ACOL00, FTA05, BBiA09b] that naturally blends both a shape's boundary and its interior texture. Similar methods have also been applied for cartoon capture and reuse [BLCD02]. Nevertheless, methods based on texture blending are susceptible to blurring artifacts unless extreme care is taken to align internal texture features. Because we use a stroke-based algorithm, our method does not suffer from blurring problems. Sýkora and colleagues [SDC09] use a similar image-based approach to register and morph cartoon frames. Their approach works well for rigid pose changes but is not designed to handle the precise interpolation of details that we require. Recent work [MHM*09] on interpolating images shows promising results given two similar input frames of video, but is not designed to handle sparse line-drawings or allow for user editing, and is prohibitively slow.

A final class of inbetweening systems employs skeleton-based methods for key frame interpolation. In the early work of Burtnyk and Wein [BW76], key frame components are embedded in skeletal structures which are animated directly to deform the embedded shape. The inbetweening work of Melikohv et al. [MTS*04] uses a skeleton concept to deform the texture surrounding hand-drawn lines. The vectorized strokes in our system encode the skeleton and the varying thickness of the drawn lines, preserving the original hand-drawn appearance.

Fekete and colleagues [FBC*95] evaluate automatic inbetweening in the context of a paperless 2D animation system and identify several advantages and disadvantages. The advantage of reduced hand-drawn inbetweens and greater animation reuse is tempered by the need to specify correspondences, struggle with awkward timing specification, and build template models. Our BetweenIT system addresses all of these concerns: correspondences are mostly automatic and require little user guidance (Section 3), timing changes are specified in a natural fashion (Section 4), and templates are not required.

3. Core Algorithms

This section details the automatic algorithms, which include graph building, matching, and interpolation. For tight inbetweening, these algorithms often produce adequate results automatically. The handling of more difficult cases that involve artist intervention is discussed in Section 4 where we present the entire workflow, including all user interaction. Our solutions are designed with the goal of an artist-friendly workflow: the focus is on algorithms that quickly produce a plausible result which can then be guided or edited by the artist.

3.1. Representation

A single drawing in our system is stored as a graph of strokes. The graph nodes, which we refer to as *salient points*, are the junctions at which strokes meet or end, and the graph edges are the strokes themselves. Each salient point contains pointers to its incident strokes, which are ordered counter-clockwise around the salient point. A single stroke S is represented as a piecewise linear curve with n vertices. Each vertex i has an associated thickness measure T_i, for $i \in 1 \ldots n$.

3.2. Stroke matching

The input to the inbetweening process is a pair of consecutive keys which have been segmented into a stroke graph. The user initiates the automatic matching algorithm in one of two ways: by selecting a pair of corresponding strokes, one from each key frame, or by selecting a region via a selection lasso. When the lasso is used, the most similar pair of strokes within the selected region is chosen automatically. This initial pair provides the seed for our Correspondence Tracing Algorithm (CTA).

In each key, starting from the selected stroke, the CTA traverses the graph in both directions in a depth-first order, respecting the circular order of the incident strokes around nodes. The traversal is performed simultaneously on both keys. The recursion stops when a incompatibility in the connectivity is detected or when two corresponding strokes are too dissimilar.

Our similarity metric computes an error E between two strokes A and B based on two different factors, E_L and E_A. E_L is the difference in arc-length between the two strokes. E_A is the area of the region bound by the two strokes when they are brought into endpoint alignment. E is then computed as $(E_A + E_L^2)/(L(A) + L(B))^2$ where L is arc-length, and compared against a constant T_E to determine if two strokes are similar. We have found that $T_E = 1$ works well and use it for all examples presented in this paper. Other similarity measures may be used [BiA06, Vel01, SKK03] that take into account proximity, orientation, curvature measures, and other shape descriptors. We have opted for a simple approach which works well in practice and is fast. No metric is perfect in all cases and for tight inbetween situations, the choice of particular method is not a major factor.

The inbetweening operates on the subgraph of strokes matched by CTA. In the tight inbetweening problem, the two keys often have identical topology and similarly shaped strokes. In such situations, CTA establishes stroke-to-stroke correspondence for a whole connected component of the key. In more complex situations, especially where the keys have different topologies, the artist has the opportunity to provide further correspondence seeds, run CTA again, and compute inbetweens for missing portions of the keys. When the lasso selection is used, successive CTA traversals are executed, each time computing the best seed among the remaining strokes not covered by the previous traversals. This process continues until no stroke pairs with $E < T_E$ remain.

3.3. Vertex Correspondence

By default, smooth (subdivided) versions of the key strokes are re-sampled using uniform spacing in arc-length in order that matched strokes on each key have the same number of vertices. This re-sampling defines the default vertex-to-vertex correspondence. The artist alters this correspondence by identifying corresponding salient points, one on each stroke of a matching pair. These salient points are inserted as new endpoints, hence splitting the strokes. More complex correspondence algorithms could be substituted [SKK03], but we have found that our simple, fast, and predictable technique works well in practice.

3.4. Composite Interpolation

Our interpolation scheme is motivated by the basic principle of action arcs in 2D animation as described in the classic *Illusion of Life*: "most movements will describe an arc of some kind...One of the major problems for the inbetweeners is that it is much more difficult to make a drawing on an arc than one halfway inbetween...No one has ever found a way of insuring that the drawings will all be placed accurately on the arcs, even when experienced people are inbetweening the action, and it is one of the most basic requirements for the scene" [JT95]. In addition to this desire for overall natural motion arcs, inter-stroke continuity must be maintained. The approach also must be efficient enough to fit into an interactive framework. To this end, we have developed a novel stroke interpolation scheme which combines a *stroke motion* constructed from logarithmic spirals with *stroke deformation* based on curvature averaging and twisting warps.

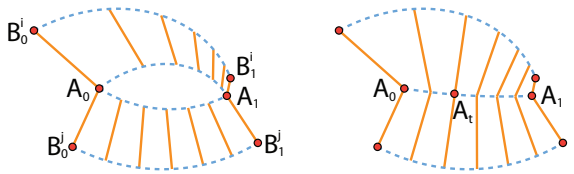

Figure 2: *Stroke motion: The left figure shows the two spiral paths resulting from calculating the spiral parameters individually for the strokes incident on A_0. On the right they have been computed as a weighted average of the parameters calculated for each incident stroke.*

3.4.1. Stroke Motion

Input strokes may be undergoing a motion that involves not only translation, but also rotation and scaling. Hence, we have developed a solution that provides natural, arched motions when the corresponding key features are congruent (related by a rigid body motion) or similar (related by an affinity that is a combination of rotation, translation and uniform scaling). The stroke motion is a map between time t (which evolves from 0.0 to 1.0), and a set of similarities. A similarity may be defined by two points and their images. Therefore, we define a stroke motion by the movement of its endpoints.

In choosing the particular type of endpoint motion, we

seek trajectories that produce pleasing arcs, preserve rigid shapes, and approximate the non-linear perspective shortening of objects that approach the viewer. Linear trajectories yield unacceptable shortening of rotating motions. Higher order polynomial curves (including cubic Bézier and rational B-splines) require endpoint derivatives or access to more than two keyframes. Furthermore, cubics and bi-arcs sometimes produce unexpected inflection points. The simplest trajectory satisfying these requirements is the logarithmic spiral (Figure 2), a pleasing curve found in nature (e.g. shells, weather systems, spiral galaxies) [Tho92].

Given an initial position A_0 and final position A_1 of an endpoint A, we compute its position A_t at time t using a logarithmic spiral motion $l(t)$ which is the combination of a rotation $r(\alpha)$ by angle α and a uniform scaling $s(\rho)$ by a factor ρ, both with respect to a fixed point (spiral center) F. We need to consider all strokes incident on A when computing the appropriate values for α and ρ. The process involves the following steps:

1. A weight is assigned to each incident stroke S^i based on its relative arc-length: We compute the sum L^i of the arc-lengths of the stroke at time $t=0$ and at $t=1$. The weight w^i is set as follows, where L_T is the total of all L^i:
$$L_T = \sum_i (L(S_0^i) + L(S_1^i)), \quad w^i = \frac{L^i}{L_T}. \tag{1}$$
2. Each incident stroke S^i, has one endpoint with initial position A_0 and final position A_1. Let B_0^i, B_1^i be the position of the remaining endpoint. We compute the angle α^i between $A_0\vec{B}_0^i$ and $A_1\vec{B}_1^i$ and the scale factor $\rho^i = \frac{||A_1\vec{B}_1^i||}{||A_0\vec{B}_0^i||}$.
3. The final angle and scale factor are a weighted average: $\alpha = \sum w^i\alpha^i$ and $\rho = \sum w^i\rho^i$.
4. The fixed point F is computed by solving the linear system
$$\vec{FA}_1 = r(\alpha)s(\rho)\vec{FA}_0 \tag{2}$$
In the case of pure translation, the determinant of the system is zero (and the fixed point is at infinity). We handle this case by performing a linear interpolation of the endpoints if the determinant is near zero.

The position A_t at time t of endpoint A from the initial key moved by this logarithmic spiral motion is
$$A_t = l(t)A_0 = F + r(\alpha t)s(\rho^t)(\vec{FA}_0) \tag{3}$$
This process is carried out for each endpoint (see Figure 2 right). For any given stroke, the stroke motion is then defined by the motions of its endpoints.

3.4.2. Stroke Deformation

The stroke motion described above produces the positions A_t and B_t of the two endpoints A, B, of a stroke at time t. We must now compute the evolving shape of the stroke that connects these endpoints. The morph should smoothly blend between the key stroke shapes while preserving tangent continuity between adjacent strokes that are smoothly connected in both keys. We achieve this with a three-step deformation involving *intrinsic shape interpolation*, *curve fitting*, and a *tangent aligning warp*.

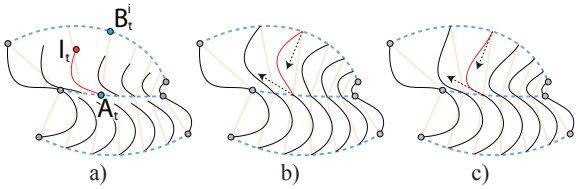

Figure 3: *Stroke deformation: (a) Intrinsic shape interpolation. (b) Curve Fitting: transforms I to align the position I_t with B_t^i. Note the discontinuity at the endpoint where the red stroke meets the black one. (c) Tangent Alignment warp: enforces continuity at endpoints.*

Intrinsic Shape Interpolation: Each stroke is represented in terms of its edge lengths, vertex angles, and vertex thicknesses. The shape of a stroke at time t is constructed from a linear interpolation of this intrinsic description as proposed in [SGWM93], which is a discrete version of curvature interpolation [SE02]. Note that if we start the construction at A_t, the resulting interpolated curve I ends at some position I_t, which does not necessarily coincide with B_t (Figure 3 a). We avoid using the optimization described in Sederberg et al. [SGWM93] to fix the discrepancy as it can produce unacceptable results, as outlined by the authors. We also interpolate the thickness parameter at each vertex linearly between keyframes.

Curve Fitting: Each stroke I is then rotated by by angle $\angle(\vec{A_tB_t})(\vec{A_tI_t})$ and scaled uniformly by $||\vec{A_tB_t}||/||\vec{A_tI_t}||$ about A_t to produce I_t' which coincides with B_t (Figure 3 b). These first two steps produce pleasing morphs that preserve the continuity of the stroke graph but fail to preserve smoothness of connections between adjacent strokes. This shortcoming is addressed by the next step.

Tangent Aligning Warp: In order to be consistent with the logarithmic spiral, we assume that the angle of the tangent at an endpoint varies exponentially over time. Therefore, we compute the desired endpoint tangent direction of an interpolated stroke by linearly interpolating the polar representation of the tangent in the two keys. Each interpolated stroke is then warped (Figure 3 c) so that it matches the desired tangents at its endpoints. To do this, we use a variation of the Twister warp [LKG*03]. First, assume the angles between the endpoint tangents of I' and the desired tangents at A_t and B_t are a and b. Assume that L is the total arc length of the stroke I'. For every vertex position P of I', L_P is the arc-length from P to A_t along I'. The Twister warp moves P to $P+(P_A-P)+(P_B-P)$, which simplifies to $P_A+(P_B-P)$, where P_A is the image of P by a rotation of angle r_A around A_t and P_B is the image of P by rotation of angle r_B around B_t, where r_A and r_B are defined as follows:
$$r_A = a\cos^2(\frac{\pi}{2}\frac{L_P}{L}), \quad r_B = b\sin^2(\frac{\pi}{2}\frac{L_P}{L}) \tag{4}$$

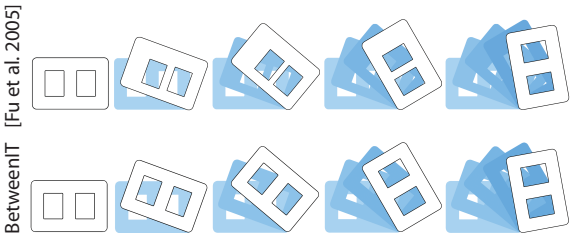

Figure 4: *Comparison with the work of Fu et al [FTA05] – Note that with our method the motion of the inner components is coordinated with the outer portion.*

The proposed interpolation scheme produces aesthetically pleasing results that ensure tangent continuity across smooth junctions between strokes and preservation of features that are present in both keys. In addition, it is an approach suitable for interactive use, since it does not require numeric iterations or other optimizations.

3.4.3. Comparison with prior art

In developing our interpolation algorithm, we have evaluated several existing techniques. Leading contemporary methods formulate the interpolation as a global optimization problem that maximizes the rigidity of the inbetween shapes to avoid scaling and shearing artifacts [FTA05, SE02, ACOL00]. Since the optimization is formulated in the differential domain, the result is invariant to global translation. Consequentially, these methods cannot be applied as-is because the input to our application is made up of multiple connected components which must move as a semantic unit when the translation invariance is resolved—an issue not addressed by existing work. Figure 4 demonstrates this problem using our implementation of [FTA05].

Our logarithmic spirals ensure that if portions of the drawing move by the same rigid body motion or affinity to new poses in another key frame, then their trajectories are consistent through inbetweening (Figure 4, 5). If the components are transformed each by a different affinity, there is no fundamental reason to believe that they need to follow a consistent path. In either scenario, our workflow allows the artist explicitly specify the path, if desired. Our stroke deformation executes interactively from start to finish in contrast to the aforementioned algorithms which require an expensive precomputation step.

4. Workflow

We have developed a natural workflow for user-guided inbetweening in which various interactive operations allow the user to optionally guide and/or override the algorithmic tools described in the previous section. As with other parts of the system, the interface design was driven by conversations with artists. For example, stroke-based interactions are generally favored over clicking, and therefore we have designed the BetweenIT interface around the concept of *guide strokes*. The semantics of a guide stroke, and hence the resulting action, depends on the current mode. The following sections describe the modes and user interactions in more detail.

Correspondence Editing: If not satisfied with the output of the automated CTA stroke matching, the user can trim or extend the set of matched stroke pairs. To remove strokes from the matched set, the user simply draws a guide stroke along them on one of the keys. To extend the matched set, the user draws a guide stroke, providing a new seed for the next round of CTA. This new CTA traversal will not override previous inbetweens and will stop once a stroke with an existing correspondence is found.

Salient Point Editing: To fine tune the correspondence, the user can specify additional salient points indirectly through guide strokes. With the strokes from the adjacent keys displayed in the same view, the user draws a guide stroke that passes through (approximately) the matching stroke pair. This action splits each stroke by inserting a new salient point. The salient point on the first stroke is the point closest to the starting point of the guide stroke and the salient point on the second stroke is computed similarly. The initial two strokes are thus broken into four. The same guide stroke serves to define the trajectory of vertex corresponding to the salient point as it moves between the keys. The user may also desire to merge adjacent strokes, thereby deleting a salient point. This operation is performed by connecting two adjacent strokes with a guide stroke.

Timing Specification: In practice, the timing for a set of inbetweens is specified prior to the inbetweening step and loaded in with the input key drawings. For total flexibility, however, the user may adjust the time parameterization interactively. As the timing is edited, any already computed inbetweens will update on the fly for immediate feedback.

Trajectory Drawing: Any of the automatically computed vertex trajectories may be edited as follows. The user draws a guide stroke to indicate the new, desired path. The ends of the trajectory are fixed, as they are specified by the key frames. It would be too tedious for the artist, however, to require that the guide stroke coincide with the initial and final vertex positions. In cases where the drawn path does not match, the guide stroke is automatically retrofit through a similarity transformation (translation, rotation, and uniform scaling) to meet the end constraints. The adjacent inbetweens that are affected by such a change are recomputed immediately. When in playback mode, the result is reflected in the animation in real-time, giving instant feedback and allowing the user to redraw the trajectory repeatedly until the desired result is obtained.

Occluded Lines Drawing: In situations where a stroke is partially or totally occluded in one key frame, but not in the next or previous key frame, the graph connectivity may be locally incompatible. The user may extend a stroke past its occlusion point by drawing a guide stroke to indicate the occluded portion of the stroke.

The added stroke is then intersected with other strokes in the same frame, creating substrokes. Each substroke has an

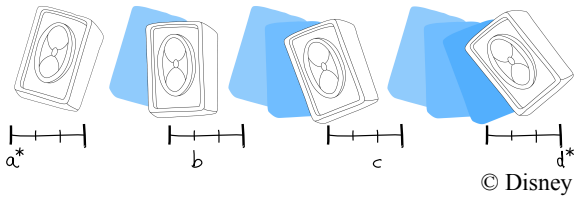

Figure 5: *Speaker: Frames (a) and (d) are the given key frames and (b-c) are generated inbetweens.*

independent visibility toggle. By default, the substrokes toggle visibility at each intersection, which works well in the common simple case, such as in Figure 10. When more control is necessary, the user may select specific substrokes to manually toggle the visibility. The visibility status is transferred automatically to subsequent corresponding substrokes through time. For even further flexibility, the produced inbetween substrokes may also be toggled individually with a tap of the pen on the stroke.

Breakdown Insertion: When two key frames or subsets of key frames are too dissimilar to be classified as "tight," the user has the option of inserting an additional "breakdown" key frame for either the entire frame, or just the subset that is not tight. The user simply draws the strokes of the inbetween and then treats them as key strokes. In this way our system allows the user to produce manual stroke-level breakdowns only where needed.

The goal of BetweenIT is to automate the production of inbetweens where possible. As the level of complexity in the drawings increases, we support intuitive and direct input from the artist to guide the automation. We expect the level of interaction or "touch time" [Cat78] to reflect the complexity of the drawing problems presented by the keys. The following examples are sample results of utilizing BetweenIT on real production data. A timeline noting the relative position of the frame is shown beneath each example. Keys are denoted with an asterisk, all other frames were generated by BetweenIT. In each example, the previous frame(s) in the sequence are indicated as a "shadow" to show the progression of the interpolation.

In the following, we present results for the purpose of evaluating the usability of BetweenIT and the quality of the inbetweens it produces. First, we show a collection of inbetweens generated by BetweenIT given archival key frames. We compare the automatically generated results visually to the hand-drawn inbetweens from the original artwork. We also present sample results from an artist evaluation of BetweenIT by the effects department.

Quantitatively we measure performance in terms of stroke count as well as time, where possible. All of the examples shown produced automated results in roughly 1 second on average. In cases where the user desires, or is required, to guide the system, he/she draws a series of "guide strokes" and/or indicates edits with mouse clicks, as described in Section 4. In the following we use interaction count (the sum of the guide strokes and input mouse clicks) versus the total

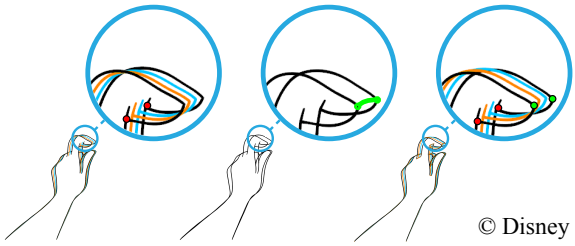

Figure 6: *Hand – User interaction: A trajectory is specified for the finger tip. The arc and resulting salient points are shown in green. The original automatically generated salient points are shown in red.*

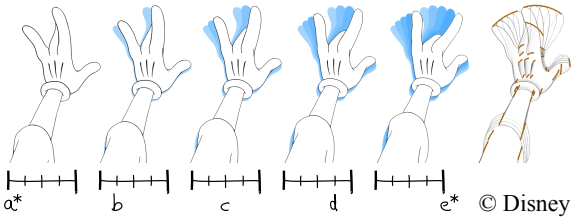

Figure 7: *Glove: Frames (a) and (e) are the given key frames and (b-d) are generated inbetweens. The rightmost figure superimposes the frames and visualizes the trajectories.*

strokes in the frame as a rough measure of efficiency gain. For example, if each of a pair of keys has n strokes and there are k inbetweened frames, then the artist would have to hand draw kn strokes to create the inbetweens. If the system produces fully automated results that are acceptable, that is a 100% gain in efficiency. If the user needs to perform i interactions to guide the system, the gain is $\frac{kn-i}{kn}$. Also note that even when this measure reports 0% gain, the use of BetweenIT may still save time, since the guide strokes need not be precise and may be drawn faster than the strokes that make up the final frame. Table 1 summarizes the results.

5. Results

Archival examples: Figure 5 illustrates an example of pure rigid body motion. In this case, the results shown were produced fully automatically.

In Figure 6 we illustrate a next possible level of interaction. Here the automatic technique has produced a reasonable solution, but the user has chosen to slightly alter the trajectory of the tip of the finger with a single drawn arc. The inbetween is automatically updated after the arc is drawn, giving the user interactive feedback. In addition to specifying the arc, this interaction transparently produces additional salient points at the tip of the finger.

The glove example in Figure 7 presents a more challenging topological problem. For example, folds appear in the glove as the hand closes. These changes in topology are handled with a few salient point insertions, stroke-merge operations, and correspondence edits. Figure 9 shows the initial interpolation and the final results after the user has manually specified 3 additional correspondences. The rightmost

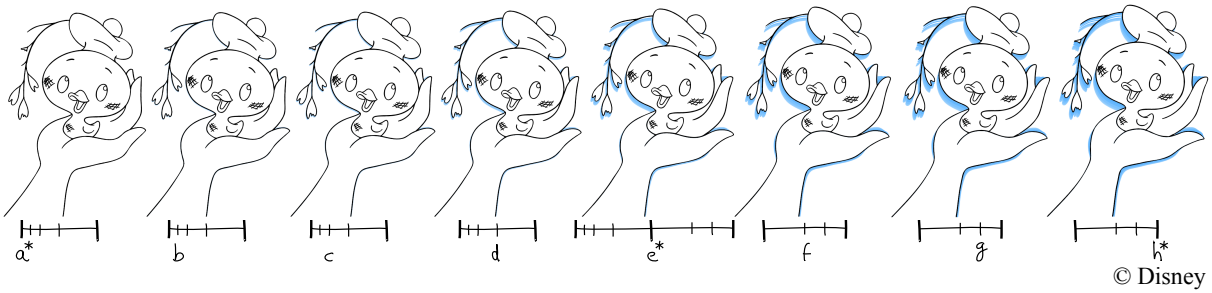

Figure 8: *Duck: Frames (a), (e), and (h) are the given key frames and (b-d) and (f-g) are generated inbetweens.*

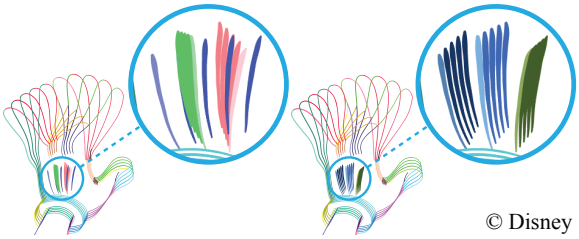

Figure 9: *Glove – User interaction: The correspondence stage mismatched portions of the hand, producing the interpolation on the left. The right image shows the results after 3 additional matches are provided manually.*

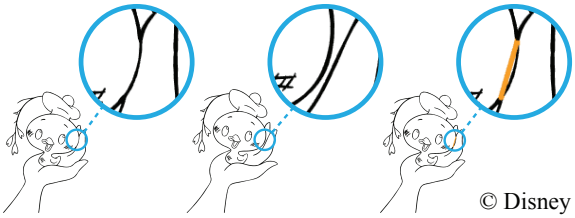

Figure 10: *Duck – User interaction: The first two images show an expanded region in each of the keys, a and e, where the finger is initially occluded and then moved from behind the duck. The user input occlusion stroke is shown on the right. Resulting inbetweens are shown in Figure 8.*

image in Figure 7 visualizes the automatically generated trajectories, illustrating the smooth arcs generated by the interpolation algorithm.

The duck sequence in Figure 8 shows two successive frame ranges. Note that unlike the previous examples the specified timing is not linear. Figure 10 illustrates an occlusion fix for this example, where a single stroke drawn by the user indicates the desired shape of the occluded region.

Figure 1 presents a step higher in the complexity range. Here there are many topological as well as occlusion challenges. The rotating right hand is the most notable drawing problem and can not be handled automatically. In this case a breakdown drawing for only the hand has been inserted for frames $b-d$, and the remaining portions of the frames were produced by the algorithm. Our framework allows the user to select desired areas and do the drawing themselves

Example	# Inb. Strk	Interactions	% Eff. Gain
Speaker	48	0	100
Glove	84	13	85
Duck	770	12	98
Tinker Bell	456	24	95
Tone	145	24	83

Table 1: *Number of interactions required to generate the inbetweens vs. total number of hand-drawn inbetween strokes.*

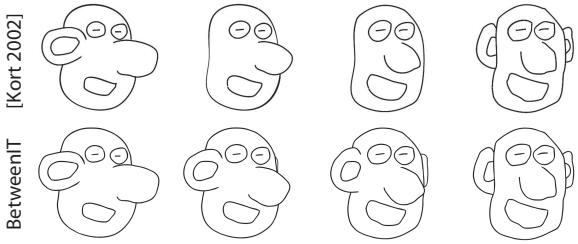

Figure 11: *Comparison of results from [Kor02] (Image © 2002 ACM, Inc.) to our results.*

seamlessly within the automated context. This is important because there will always be components of the drawings that the artist will want to do themselves.

Finally, we compare our results to two existing approaches, one stroke-based and the other image-based. Kort's stroke-based technique [Kor02] attempts to solve occlusion issues algorithmically, but is only successful in simple cases, and therefore for the example shown in Figure 11, the ears present a failure case. In contrast, our system requires user interaction for correspondence, feature identification, and occlusion (totaling 29 interactions) but is able to successfully generate coherent inbetweens. In Figure 12, we compare the results of recent work by Sýkora and colleagues [SDC09] on the keyframes from Figure 7. Due to the non-rigid nature of the animation and sliding occlusion boundaries, it is difficult to achieve clean inbetweens without artifacts using an image-based technique. In addition, we favor a stroke-based approach because it enables preservation of line quality, and fine grain, intuitive editing.

For the purposes of evaluating our system, in addition to looking at potential efficiency gain, we can compare the results of the algorithm to hand-drawn inbetweens. We do not

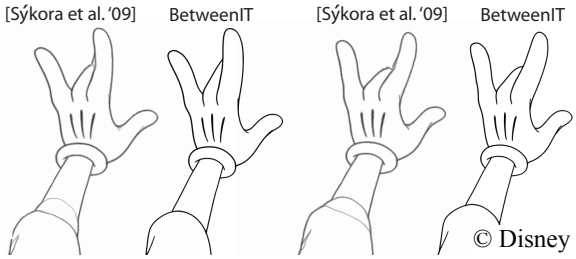

Figure 12: *A comparison of 2 inbetweens generated by [SDC09] and BetweenIT from the keyframes in Figure 7.*

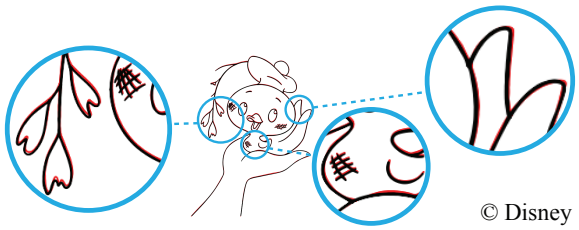

Figure 13: *Duck: An example frame comparing the results shown in Figure 8 by overlaying a hand-drawn frame in red.*

expect in general an exact match, as even between artists, different drawings are produced, but such a comparison can give some indication of the success of our approach. Figures 1 and 13 compare the BetweenIT-generated results to the original hand-drawn inbetweens by overlaying the two.

All of the above results were presented to a lead clean up artist to judge quality – as she would evaluate any of the inbetweens produced in her department. For the duck example and the glove example there was one note each regarding the shape of a single inbetweened stroke. To produce the desired shape for the duck it was sufficient to add one additional user-defined salient point. The note for the glove requested a change of the location of the knuckle bend (Figure 14). To implement this fix, a stroke was drawn as a breakdown to the middle inbetween frame to produce the final results shown in the bottom row of Figure 14.

Trial Evaluation: In addition to comparing the generated inbetweens to existing hand-drawn examples, BetweenIT was evaluated by the effects department who used the tool to inbetween tone shapes. Tones, highlights, and shadows represent only a portion of the types of inbetweening performed in effects, but a time-consuming portion, and one well-suited to automation. Figure 15 shows an example tone. The frog character was loaded in as a reference background on which to draw the tone shown on the right side of her face.

Another example showing the outline of a tone of a character's face and shoulder is shown in Figure 16. A hand-drawn version of the 29 inbetween frames in this shot took approximately an entire day to complete, while the version produced by BetweenIT was completed end-to-end in roughly 30 minutes. These numbers are just a indication of efficiency: the artist completed this example after just a single afternoon to get familiar with BetweenIT. We expect the

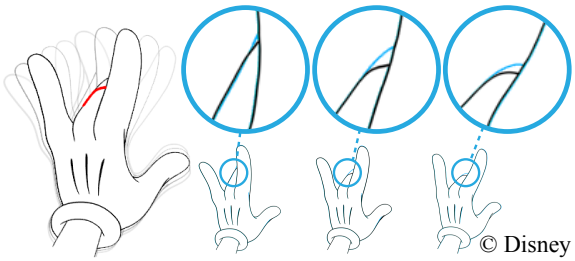

Figure 14: *Glove: The left figure shows an artist's desired fix (in red) to Figure 7. The remaining figures show the results after inserting a breakdown stroke, with.a comparison to the stroke before the fix (blue).*

Figure 15: *Frog Tone: This example contains 9 key frames and 22 inbetweens. The leftmost figure shows the tone shape on top of a background reference image, the middle figure shows the composited frame, and the rightmost figure shows all 31 tone frames superimposed and color-coded by key.*

efficiency gain to increase with more exposure to the tool. Our stroke efficiency measure for this example is 83% (20 salient points and 4 trajectories drawn), which maps to the task being completed 16 times faster than by hand.

6. Conclusion

We have presented the BetweenIT system for the user-guided automation of tight inbetweening. For cases where the user is not satisfied with the automated results, BetweenIT provides a context in which the user can guide the system in a natural way to produce quality results efficiently. The inbetweening is driven by a novel solution for stroke interpolation along natural arcs from only two keys.

We demonstrate workflow and results using BetweenIT on a collection of real production examples of varying complexity. The results generated are comparable to hand-drawn examples in many of our test frames. Where user intervention was necessary, the required input was small relative to the resulting reduction in the number of overall drawn strokes. After a single introductory session with the tool, an effects artist was able to efficiently produce inbetweened tones on multiple shots that were included in the final film.

There are many avenues for improvement in our current system. We have primarily focused on the interpolation algorithms: the correspondence and feature identification modules could benefit from further exploration. High-quality vectorization is also an open area.

Acknowledgements

We would like to acknowledge the artists and production crew of the Walt Disney Animation Studios for their valu-

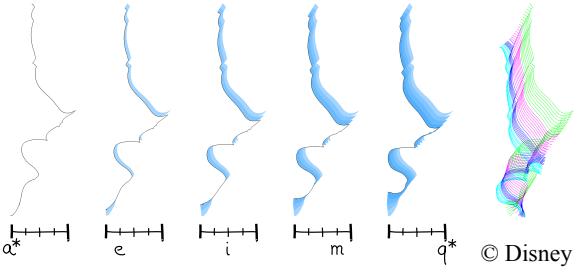

Figure 16: *Face Tone: Frames (a) and (q) are two key frames for the tone shape and (e,i,m) are a subset of the 15 generated inbetweens (b-p). The rightmost figure superimposes tone shapes for all 34 frames, with (a-q) shown in green.*

able input and inspiration, with special thanks to Rachel Bibb. Gene Lee engineered the prototype into a production tool, and Felipe Cerdan headed the testing in effects. We thank Joe Marks for initiating and supporting this project. Thanks to Daniel Sýkora for providing the example in Figure 12.

References

[ACOL00] ALEXA M., COHEN-OR D., LEVIN D.: As-rigid-as-possible shape interpolation. In *Proc. of ACM SIGGRAPH 2000* (2000), Computer Graphics Proceedings, Annual Conference Series, pp. 157–164.

[BBiA09a] BAXTER W., BARLA P., ICHI ANJYO K.: Compatible embedding for 2d shape animation. *IEEE Transactions on Visualization and Computer Graphics 15*, 5 (2009), 867–879.

[BBiA09b] BAXTER W., BARLA P., ICHI ANJYO K.: N-way morphing for 2d animation. *Comput. Animat. Virtual Worlds 20*, 2 (2009), 79–87.

[BiA06] BAXTER W., ICHI ANJYO K.: Latent doodle space. *Computer Graphics Forum 25*, 3 (2006), 477–486.

[BLCD02] BREGLER C., LOEB L., CHUANG E., DESHPANDE H.: Turning to the masters: Motion capturing cartoons. *ACM Transactions on Graphics 21*, 3 (2002), 399–407.

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

[BW76] BURTNYK N., WEIN M.: Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *CACM* (1976).

[Cat78] CATMULL E.: The problems of computer-assisted animation. In *Proc. of SIGGRAPH 1978* (1978), Computer Graphics Proceedings, Annual Conference Series, pp. 348–353.

[dJB05] DE JUAN C. N., BODENHEIMER B.: Re-using traditional animation: Methods for semi-automatic segmentation and inbetweening. In *Proceedings of Symposium on Computer Animation 2006* (2005), pp. 100–102.

[Dur91] DURAND C.: The toon project: Requirements for a computerized 2d animation system. *Computers & Graphics 15,2* (1991), 285–293.

[FBC*95] FEKETE J., BIZOUARN E., COURNAIRE E., GALAS T., TAILLEFER F.: Tictactoon: A paperless system for professional 2d animation. In *Proc. of SIGGRAPH 1995* (1995), Computer Graphics Proceedings, Annual Conference Series, pp. 79–90.

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

[JT95] JOHNSTON O., THOMAS F.: *The Illusion of Life*. Disney Press, 1995.

[KBB82] KOCHANEK D., BARTELS R., BOOTH K.: *A Computer System for Smooth Key Frame Animation*. Tech. Rep. CS-82-42, University of Waterloo, 1982.

[Kor02] KORT A.: Computer aided inbetweening. In *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering* (2002), pp. 125–132, http://doi.acm.org/10.1145/508530.508552.

[Lev77] LEVOY M.: A color animation system: based on the multiplane technique. In *Proc. of SIGGRAPH 1977* (1977), Computer Graphics Proc., Annual Conference Series, pp. 65–71.

[LKG*03] LLAMAS I., KIM B., GARGUS J., ROSSIGNAC J., SHAW C. D.: Twister: a space-warp operator for the two-handed editing of 3d shapes. In *Proc. of ACM SIGGRAPH 2003* (2003), Computer Graphics Proceedings, Annual Conference Series, pp. 663–668.

[MHM*09] MAHAJAN D., HUANG F.-C., MATUSIK W., RAMAMOORTHI R., BELHUMEUR P.: Moving gradients: a path-based method for plausible image interpolation. In *Proc. of SIGGRAPH 2009* (2009), Computer Graphics Proceedings, Annual Conference Series, pp. 1–11.

[MIT67] MIURA T., IWATA J., TSUDA J.: An application of hybrid curve generation: cartoon animation by electronic computers. In *AFIPS '67 (Spring): Proceedings of the April 18-20, 1967, spring joint computer conference* (1967), pp. 141–148.

[MTS*04] MELIKHOV K., TIAN F., SEAH H. S., CHEN Q., QIU J.: Frame skeleton based auto-inbetweening in computer assisted cel animation. In *CW '04: Proceedings of the 2004 Intl. Conference on Cyberworlds* (2004), IEEE Computer Society, pp. 216–223.

[Ree81] REEVES W.: Inbetweening for computer animation utilizing moving point constraints. In *Proc. of SIGGRAPH 1981* (1981), Computer Graphics Proceedings, Annual Conference Series, pp. 263–270.

[SDC09] SÝKORA D., DINGLIANA J., COLLINS S.: As-rigid-as-possible image registration for hand-drawn cartoon animations. In *NPAR '09: Proceedings of the 7th Intl. Symposium on Non-Photorealistic Animation and Rendering* (2009), pp. 25–33.

[SE02] SURAZHSKY T., ELBER G.: Metamorphosis of planar parametric curves via curvature interpolation. *Intl. J. of Shape Modeling 8(2)* (2002), 201–216.

[SGWM93] SEDERBERG T. W., GAO P., WANG G., MU H.: 2-d shape blending: an intrinsic solution to the vertex path problem. In *Proc. of SIGGRAPH 1993* (1993), Computer Graphics Proceedings, Annual Conference Series, pp. 15–18.

[SKK03] SEBASTIAN T. B., KLEIN P. N., KIMIA B. B.: On aligning curves. *IEEE Trans. Pattern Anal. Mach. Intell. 25*, 1 (2003), 116–125.

[SZGP05] SUMNER R. W., ZWICKER M., GOTSMAN C., POPOVIĆ J.: Mesh-based inverse kinematics. *ACM Transactions on Graphics 24*, 3 (2005), 488–495.

[Tho92] THOMPSON D. W.: *On Growth and Form*. Cambridge University Press, 1992.

[Vel01] VELTKAMP R.: Shape matching: similarity measures and algorithms. In *Shape Modeling and Applications, SMI 2001 Intl. Conference on.* (May 2001), pp. 188–197.