Space-time sketching of character animation

Martin Guay, Rémi Ronfard, Michael Gleicher, Marie-Paule Cani

To appear in ACM TOG 34(4).

Space-time sketching of character animation

Martin Guay* Université de Grenoble LJK, INRIA
Rémi Ronfard Université de Grenoble LJK, INRIA
Michael Gleicher University of Wisconsin Madison
Marie-Paule Cani Université de Grenoble LJK, INRIA

Abstract

We present a space-time abstraction for the sketch-based design of character animation. It allows animators to draft a full coordinated motion using a single stroke called the *space-time curve* (STC). From the STC we compute a dynamic line of action (DLOA) that drives the motion of a 3D character through projective constraints. Our dynamic models for the line's motion are entirely geometric, require no pre-existing data, and allow full artistic control. The resulting DLOA can be refined by over-sketching strokes along the space-time curve, or by composing another DLOA on top leading to control over complex motions with few strokes. Additionally, the resulting dynamic line of action can be applied to arbitrary body parts or characters. To match a 3D character to the 2D line over time, we introduce a robust matching algorithm based on closed-form solutions, yielding a tight match while allowing squash and stretch of the character's skeleton. Our experiments show that space-time sketching has the potential of bringing animation design within the reach of beginners while saving time for skilled artists.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: Sketch-based animation, space-time, stylized animation, squash-and-stretch.

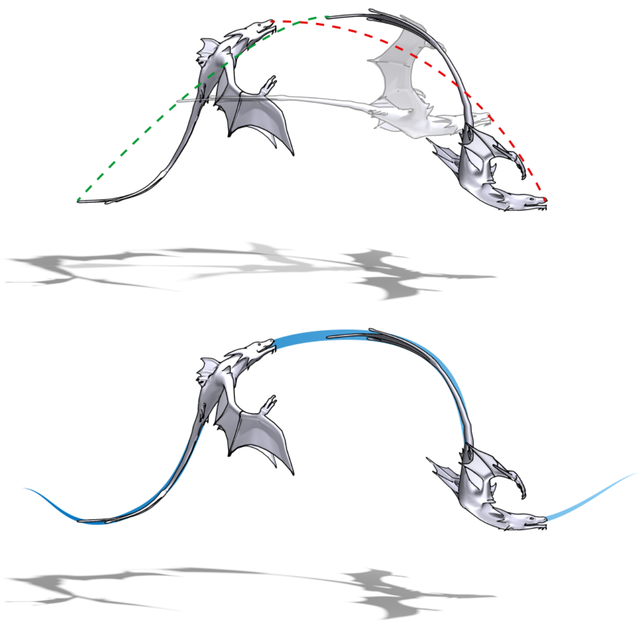

Figure 1: *Current shape interpolation techniques assume point-to-point blending (first row, result shown in grey), making it hard to create path-following motions. In contrast, our space-time sketching abstraction enables animators to sketch shapes and paths with a single stroke (second row).*

1 Introduction

Creating artistic and exaggerated styles of character animation requires flexible tools that allow expressive devices such as squash and stretch, as well as animating imaginary creatures such as dragons—often precluding the use of motion capture or database look-up. Yet, creating quality movements with current free-form animation technologies is a challenge.

The main approach to free-form motion design is keyframing: character poses at specific times are interpolated to produce motion. Over the years, significant advances have been made to more naturally specify key-poses. For example, by sketching skeletons or lines of action, or by handling deformations. However, the standard keyframing approach divides spatial and temporal control, making coordination of shape over time difficult. Hence, achieving quality results with the standard approach remains beyond the ability of unskilled artists and time consuming for skilled ones.

In this work, we introduce a novel space-time sketching concept enabling an animator to draft a full coordinated movement—that includes shape deformation over time—by sketching a single stroke. Further strokes can be used to progressively refine the animation. While strokes have been used in the past to specify both temporal and spatial iso-values of motion—with static lines of action (LOA) serving as shape abstraction at a given time as well as trajectories describing the successive positions over time of a single point—*space-time sketching* was never used to define animations. In our approach, we allow the user to control both the shape and trajectory of a character by sketching a single *space-time curve* (STC).

To illustrate our approach, consider the simple example of animating a flying dragon (Fig. 1). Animating the dragon requires the coordination of its shape over time as to follow the path's shape. With our approach, the basic animation can be created with a single sketched stroke. The stroke is used not only to provide the path of travel, but also to define how an abstraction of the character's shape (its line of action) changes over time. Additional strokes can be used to refine the movement, or add details such as the flapping of the wings. Creating such motion with existing approaches would require coordinating a large number of keyframes that specify deformations and positions along the path, or a method for puppeteering the degrees of freedom of the dragon.

The key to our approach is an interpretation of the space-time curve to define a 2D dynamic skeletal line abstraction, or *dynamic line of action* (DLOA). From the STC, we extract the DLOA's *shape* at *discrete moments in time*, or *continuously over time*—depending on the curve's features. In simple cases, such as Fig. 1, a path-following DLOA is defined as a moving window within the parametric space of the stroke. Looking closely at existing ways of sketching animations (see Fig. 2), we observe that the stroke encodes *shape* near singular points as well as between self-intersections, enabling us to easily define bouncing and rolling motions from the STC. The speed at which the stroke is drawn directly controls squash and stretch deformations. The animation can be

*lemailamartin@gmail.com

further refined by adding wave and twist specifications, by over-sketching lines of action or by adding secondary lines—possibly drawn from other viewpoints.

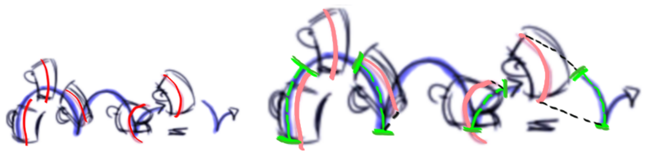

Figure 2: *Inspiration: a hopping mug. The artist made the lines of action "dynamic" by having them match the blue trajectory (green marks were added on the right to show the matching parts). This shows that the blue stroke carries both shape and path information.*

In addition to synthesizing a complete DLOA from a space-time curve, a second challenge is to animate a 3D character matching the shape of the DLOA over time. A frame-by-frame computation of the character's 3D pose from its LOA [Guay et al. 2013] is not sufficient to produce smooth motions in the dynamic case. Additionally, this method solves only for rigid bone transformations and does not allow squash and stretch. To solve these problems, we provide a robust line matching algorithm combining closed-form solutions with dynamic programming yielding a tight match between the character and the line as well as allowing squash and stretch of the character.

We evaluate our approach with an informal pilot user study showing that space-time sketching is much faster than using keyframes alone—even compared with sketching expressive lines of action. Moreover, it makes animation possible for beginners.

2 Related work

Specifying new and complex character movements using only low dimensional input such as a mouse, pen or other hand held device has been an active area of research in recent years. In previous works, partial solutions are proposed to control different aspects of the animation, i.e. the shape (poses) and motion (timing and trajectories), separately.

Keyframing. Inspired from the hand-drawn animation tradition, computer animations are often decomposed into sequences of poses in time. Specifying each pose individually and controlling the timing through a timeline remains to this day the established pipeline. To facilitate the posing of complex character shapes, researchers and engineers have proposed natural subspaces such as skeletons [Burtnyk and Wein 1976] and cages [Sederberg and Parry 1986], parameterizing the surface geometry with fewer degrees of freedom to manipulate. More recently, efforts to allow hand drawings to control the character were brought forward with stick figures [Davis et al. 2003], partial bone lines [Kho and Garland 2005; Öztireli et al. 2013], and lines of action [Guay et al. 2013]—the latter allowing animators to create more expressive poses in a single hand gesture. However, poses are hard to coordinate through a timeline disconnected from spatial realities. In contrast, our approach abstracts several spatio-temporal aspects of the motion such as shape, timing, and trajectories into a single stroke facilitating coordination, while remaining compatible with the keyframing pipeline.

Layered motion. Instead of sequencing poses, researchers have used the idea of layering partial motions [Dontcheva et al. ; Neff et al. 2007; Shiratori et al. 2013]. The user selects body parts [Dontcheva et al.], or degrees of freedom [Neff et al. 2007; Shiratori et al. 2013], and performs while the motion is recorded. Although performing can be very intuitive, coordinating different layers of performance can be hard. As a result, the animator cannot directly control the pose (shape) at a specific time. In contrast, we allow the user to act-out aspects of trajectory and timing, decoding which part of the character should follow the path and how, while allowing the user to refine and control specific poses in time.

Data-driven animation. Because low level control only goes so far, several works propose specifying as input high level motion abstractions like doodles and trajectories, along with pre-existing motion—often humanoid—to "fill" the missing information [Thorne et al. 2004; Min et al. 2009]. In [Min et al. 2009], a small number of point trajectories are sketched to drive characters by using strong priors based on previously captured motions. Closer to our work, [Thorne et al. 2004] use sketching to drive characters in a very intuitive manner. However, they too rely on previously defined motion clips to map the strokes to whole character motions. In contrast, our approach provides direct control over the character without relying on strong priors from previous motions, making possible the creation of novel motions even in the case of unrealistic morphologies or exaggerated movements.

Procedural animation. Nonlinear deformers such as sinae and waves motions can be applied to a character's shape to produce nice-looking animation effects in several authoring softwares such as Blender and Maya. But such effects typically give very little control to the user. More sophisticated methods for physically-based simulation have been proposed to synthesize realistic character animation [Hodgins et al. 1995; Yin et al. 2007], but they suffer from the same problem. One work attempted to combine physical simulation with gesture-based control [Laszlo et al. 2000], but cannot prevent nonlinear chaotic behaviors. In our work, the user directly controls the motion with a gestured stroke, providing predictable movements, not necessarily bound to physical realism.

Paths. Several authors have proposed to create 3D animation by sketching 2D paths [Davis et al. 2008] embedded in a 3D world [Igarashi et al. 1998; Gleicher ; Thorne et al. 2004]. These are typically drawn onto the floor to control the global rigid 2D configuration of the character over time. Our space-time curves are a different concept, as they drive both the temporal trajectory of characters and their shape over time.

3 Overview

The main concept in our method is to enable space-time sketching through a single stroke, called the *space-time curve*. From this curve and the speed at which it was drawn, we automatically initialize a *dynamic line of action* (DLOA), which itself drives a specific *body line* in the character model (e.g. spine, tail, wings, etc—or combination of these). We enable refinement of the resulting coarse motion through over-sketching the DLOA at specific times or adding secondary lines or controls.

Both the space-time curve and the DLOA are planar, defining projective constraints for the 3D animated character. This allows simpler user input and enables us to decouple expressive motion specification, done from a target camera viewpoint, from a specific 3D character model. The same DLOA can be re-applied to other body parts or to different characters, enabling the user to draft animations even before the character is fully modeled.

The different ways a dynamic line of action can be initialized from strokes are discussed in Section 4. In Section 4.1, we lay some foundations were strokes are used to control only shape. Then we build on this foundation to control both shape and trajectory with a single stroke (the *space-time curve* (STC) described in Section 4.2). The STC can be used to produce path-following DLOAs, critical to creating styles of animations such as Fig. 1, where standard keyframing falls short. In the abstract form, the STC initializes DLOAs such as bouncing and rolling. We describe in Section 5 how to refine the animation by mixing and composing the resulting DLOA with other strokes, as well as controlling their twisting orientation via space-time cans. Lastly, in Section 6 we present a robust method to match a 3D character's skeleton to the DLOA.

4 Dynamic lines of action from strokes

A dynamic line of action (DLOA) is a 2D **parametric surface** $x_{dloa}(s,t)$, where t is time and s is the parameter along the line. This surface can be displayed as a moving line by successively showing temporal iso-lines $x_{dloa}(s,t_i), i = 0...I$. In Fig. 3, we show a visualization of the whole DLOA surface.

In this section, we discuss how such a surface can be created from sketches: we first investigate the extension of the standard keyframing concept to LOAs. We then introduce a new abstraction, the space-time curve, and explain how it can be used to quickly draft and then refine the DLOA in a principled way.

The 2D stroke the user draws is denoted $C = [c_0, ..., c_{N-1}] \in \mathbb{R}^{2\times N}$ ($c_j \in \mathbb{R}^2 \,\forall\, j$), where the c_j are the samples recorded by the input pen device. A smooth curve is built with a set of piecewise linear basis functions $\{b_j\}_{j=0}^{N-1} \in [0,\,1]$ centered at the parameterization points $U = [u_0, ..., u_{N-1}] \in [0,1]^N$, resulting in:

$$c(u) = \sum_{j=0}^{N-1} c_j b_j(u),$$

which can be written in matrix form, $c(u) = CB(u)$, where $B(u) \in \mathbb{R}^{N\times 1}$. By default, the parameterization is evenly spaced, i.e. $u_j = \frac{j}{N-1}$. As the user does not sketch at constant speed, squash and stretch will appear. Using an arc-length re-parametrization will produce uniform local lengths (eventually leading to uniform stretch of the character).

4.1 Keyframing lines of action

While this paper is focused on specifying DLOAs from a single space-time curve, we first describe the interpolation of static LOAs at key moments k, $x_{dloa}(s,t_k), k = 0...M-1$. The user can create a DLOA by sketching multiple key-LOAs, as shown in Fig. 3.

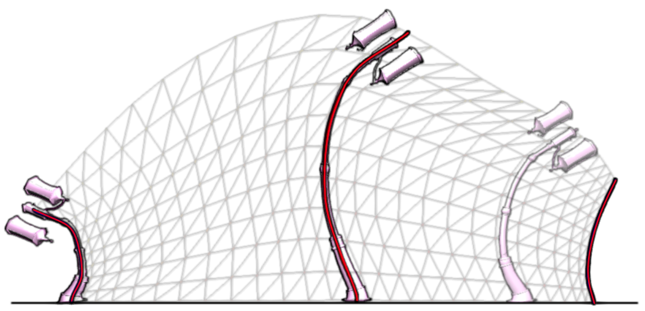
Figure 3: *The user can create the space-time surface (DLOA) by sketching strokes (red) at key moments. Our matching method (Section 6) allows squash and stretch of the character's bones. Note that squash and stretch may require volume preservation which is not provided by linear blend skinning used in these figures.*

Enabling this keyframing process to take place *in screen space* is different from using each key-LOA to pose the 3D character and interpolating the character's configuration space, as done in [Guay et al. 2013]. 2D interpolation is an improvement over the prior approach as it can be used to ensure fluidity in the perceived motion, and has the advantage of decoupling animation specification from the 3D character model. However, two issues need to be solved for good quality interpolation between LOAs: length preservation and C1-smooth motion.

Local length-preserving interpolation. Static LOAs are 1D shapes. Interpolating two of them—best in a rigid as possible manner—requires *local* length preservation: if parts of the two extreme LOAs have the same length, the same part in all the intermediates should have the same length as well. Simply interpolating between their cartesian points (e.g. $x_{dloa}(s,t_k)$ and $x_{dloa}(s,t_{k+1})$) does not preserve length.

We achieve length preservation as in [**?**; Alexa et al. 2000] by decomposing each LOA $x_{dloa}(s,t_k)$ into polar components—local angle and length of each polyline—to perform the interpolation, and then recover the interpolated curve by integrating the rotated and scaled polylines. The angles and lengths are respectively denoted $\Theta = [\theta_0, ..., \theta_{N-2}]$ and $L = [l_0, ..., l_{N-2}]$. Given angles lengths, and a root position x_0, the curve is recovered with:

$$x_j = \sum_{n=0}^{j} l_n R(\theta_n)\,\bar{x} \;+ x_0, \qquad (1)$$

where $\bar{x} = (1,0)$ is the reference x-axis and R is a rotation the 2D plane. We compute the *absolute angle* between each polyline $x_{j+1} - x_j$ and the reference x-axis $\bar{x}$ together with local length:

$$\{\theta_j,\, l_j\} \quad = \{\angle\,(\bar{x}, x_{j+1} - x_j)\,,\;\; \|x_{j+1} - x_j\|\}\,. \qquad (2)$$

C1-smooth motion. Linear interpolation of sparse keyframes can produce discontinuities at keyframe transitions—imagine a robot performing one action at a time. To allow creating fluent motion with few keyframes, we need an interpolation that provides smooth first derivatives (tangents). To do so, we use cubic Hermite spline interpolation, where the tangents at the key frames are computed automatically using Catmul-Rom averaging.

Suppose we want to interpolate between two key-lines $x(s,t_k)$ and $x(s,t_{k+1})$ over a time interval $[t_k, t_{k+1}]$. The polar components $[\theta(s,t_k)\; l(s,t_k)\, x(0,t_k)]^T$ are sequenced into a matrix $\Psi = [\Theta(s)\; l(s)\, X_0]^T \in \mathbb{R}^{3\times M}$ of M lines (in our example $M = 2$). We build a C1-smooth interpolation by stacking the first derivatives (the tangents) $\dot{\Psi}$ to the right of the matrix $\left(\left[\Psi\,\dot{\Psi}\right] \in \mathbb{R}^{3\times 2M}\right)$, and using Hermite basis functions $B(t) \in \mathbb{R}^{2M\times 1}$:

$$\begin{bmatrix} \theta(s,t) \\ l(s,t) \\ x(0,t) \end{bmatrix} = \begin{bmatrix} \Theta(s) & \dot{\Theta}(s) \\ L(s) & \dot{L}(s) \\ X_0 & \dot{X}_0 \end{bmatrix} B(t). \qquad (3)$$

Given the smooth angles, lengths and root positions $[\theta(s,t),\, l(s,t),\, x(0,t)]^T$, we compute a smooth dynamic line of action by following Eq (1):

$$x_{dloa}(s,t) = \int_{s=0}^{1} l(s,t) R(\theta(s,t))\,\bar{x}\,ds \;+ x(0,t).$$

4.2 Space-time curves

Although keyframing is a useful building block for defining DLOAs, it cannot easily handle cases shown in Fig. 1 and Fig. 2, where shape and movement need to be carefully coordinated. The key idea to our approach is to enable the initialization of a complete spatio-temporal surface $x_{dloa}(s,t)$ from a single sketched stroke by re-interpreting the stroke parameter u into separate space and time parameters s and t.

We start by describing a simple path-following behavior, which is critical in generating styles of movement such as the one in Fig. 1. Then we extend our concept to non-degenerate initialization, producing bouncing (hopping) and rolling motions. In all cases, the resulting animation can be immediately shown to the user, and then refined by adding more strokes and controls (Section 5).

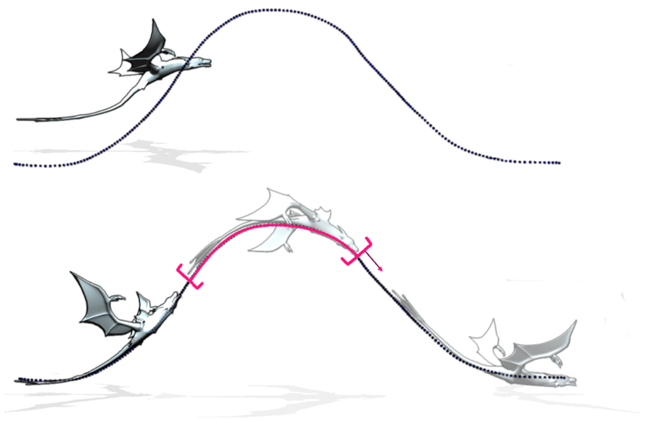

Figure 4: *Top image: character model and space-time curve (STC). Bottom image: we dynamically warp (red window) within the parametric space of the STC to produce a dynamic line of action. Squash and stretch is controlled directly by drawing faster and slower.*

Path-following. Let us consider Fig. 4 (top), where a long curve $c(u)$ is drawn, and the character (or line of action) is expected to move *through* the curve. This is achieved by defining a *moving window* within the parametric space of c:

$$x_{dloa}(s,t) = c(w(s,t)), \tag{4}$$

where $w(s,t)$ is a warping function that scales and shifts in time a parameter s. To compute $w(s,t)$, we first scale s with a constant b. To some extent, b reflects the character's length in parametric space (its computation being detailed below). Then we shift the window forward in time with t; only t has to be scaled, as to ensure the window remains within the STC's parametric space (e.g. $\subseteq [0,1]$)—leading to the following warping function, to be used as the parameter in Equation (4):

$$w(s,t) = t(1 - b\,s) + b\,s, \tag{5}$$

The scale b sets the length of the line $x_{dloa}(s,t_i)$ at a given time step t_i, and ultimately, of the character in screen space.

The choice of parameterization U for the input stroke also has an impact on the local lengths of the dynamic lines: as the user rarely draws at a constant speed (sample points along the curve are not evenly spaced), using a uniform parameterization U leads to local variations in length—useful for controlling squash and stretch along the dynamic lines. This can be removed if desired by re-parameterizing the curve U at constant arc-lengths.

A simple way to estimate b given an arbitrary parameterization U is to define it as the average length ratio between the skeleton's initial rest pose and the full length of the path:

$$b = \frac{\int_{s=0}^{1} \|\frac{\partial P_{vp} x_{body}}{\partial s}(s)\| ds}{\int_{u=0}^{1} \|\frac{\partial c}{\partial u}(u)\| du}. \tag{6}$$

Remembering that the DLOA $x_{dloa}(s,t)$ is a 2D *surface*, we can note that the simple initialization we just described is very specific, since it fits the full surface into a curve. We now describe two, more abstract ways of initializing DLOAs from a curve, inspired by the way people tend to sketch motion (Fig. 2 and Fig. 7). In these sketches, the singular points and the self-interactions along the curve are interpreted as bouncing and rolling indications, used to automatically create a non-degenerated space-time surface for the DLOA. Note that in [Thorne et al. 2004] sketches are also used to specify motions, but only to select pre-defined motion clips. In this work, we formally establish a geometric link between the strokes and the shape of an abstract line; which in turn can be applied to arbitrary character morphologies.

Bouncing Let us consider the sketch in Fig. 2 used to represent a bouncing motion. In this case, semi-circles are drawn to specify contact intervals, interleaved by in-air paths. Important to note is how the semi-circles provide not only a rough trajectory, but also the *shapes* at takeoff and landing (red lines of action).

Inspired from this sketch, we provide an automatic way to synthesize a DLOA from a space-time curve $c(u)$ that holds singular points. From this single stroke, we extract both shape (the red lines) and trajectory, as well as timing—which is determined from the time spent at the contact points when drawing the stroke.

We detect singular points along the curve and define a suitable warping window (Equation (5)) to pick-out individual key LOAs (the red curves touching the singular points in Fig. 2), while providing their respective timing via the speed at which the curve was drawn. Interpolating these key LOAs using Equation (3) gives a first *intermediate* DLOA that is denoted $\hat{x}_{dloa}$. It has the right shape over time, but not the right trajectory yet—at least not in the air stage between contact points.

We now adjust the trajectory of the intermediate DLOA $\hat{x}_{dloa}$ as to approximately match the trajectory conveyed by the stroke. Our solution is based on two steps: first, we detect which point $s^*(t)$ on the line needs to follow the trajectory, based on the off-ground angle (detailed below). The second step is to ensure smoothness between contact and flight stages by constructing a trajectory $T(t)$ that will smoothly link 3 points for each semicircle: takeoff $c(u_1)$, middle of semicircle $c(u_2)$, and landing $u_{air} = (u_1 + u_2)/2$.

Given the point $s^*(t)$ in the dynamic LOA that is to match the trajectory $T(t)$, we define the constraint $T(t) = x_{dloa}(s^*(t),t)$. The constraint is linear and can be met by adding a correction term $\Delta x(t) = T(t) - \hat{x}_{dloa}(s^*(t),t)$, to the intermediate $\hat{x}_{dloa}(s^*(t),t)$; resulting in the final bouncing DLOA:

$$x_{dloa}(s,t) = \hat{x}_{dloa}(s,t) + \Delta x(t). \tag{7}$$

The point in the line (or character's body) $s^*(t)$ that follows the trajectory is computed based on the off-ground angle of the semicircle. For angles smaller than a threshold, $s^*(t)$ is the bottom tip of the LOA, and it goes to the the middle of the line for a vertical takeoff.

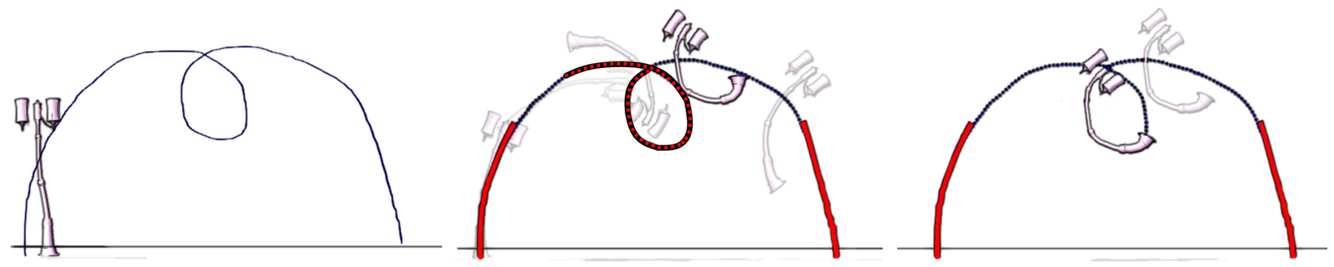

Figure 5: *The user sketches a loop shape (first row). The resulting motion blends between the key frames at the side (second column) and a path-following DLOA in the middle section of the stroke—between self-intersecting points. Third column is the intermediate DLOA before a trajectory correction (solid) and the final DLOA is ghosted.*

The coordinate $s^*(t)$ is the interpolation between the coordinate at the takeoff keyframe s_1^* and the coordinate at the landing keyframe s_2^*, resulting in: $s^*(t) = s_1^* t + (1-t)s_2^*$.

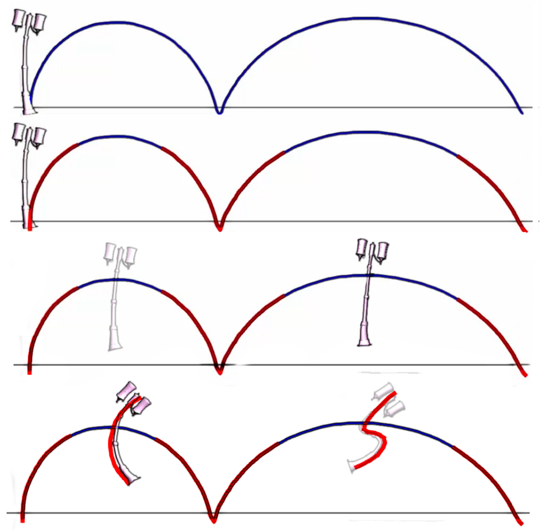

Figure 6: *Our method parses the space-time curve for singular points, and uses a warping step function to pick individual key frames (red lines)—which are then interpolated to provide the shape of the character over time, while the trajectory is determined by the path. First row: the user sketches a space-time curve. Second row: the key frames that are automatically picked out. Third row: interpolated motion with a trajectory correction. Fourth row: the user edits the initial motion by over-sketching keyframes on the space-time curve—automatically providing its timing.*

Rolling Another abstraction is when the user sketches a loop to convey a rolling movement (see Fig. 7). Indeed the loop could be interpreted as a single rigid rotation. However, looking closely, we see how the upper body of the character matches the *shape* of the loop over time (dashed line in the right of Fig. 7). This is similar to the *path-following* behavior described earlier, only the difference is that in the rolling case, the trajectory and shape do not coincide; if a point in the body did follow the red curve in Fig. 7, the body would move backward to the left before continuing its flight to the right. In other words, between self-intersections we have a shape-following constraint and the sketch only roughly indicates a trajectory. We use this abstract interpretation of rolling motions to automatically extract a rolling DLOA from a space-time curve $c(u)$ holding self-intersections as follows.

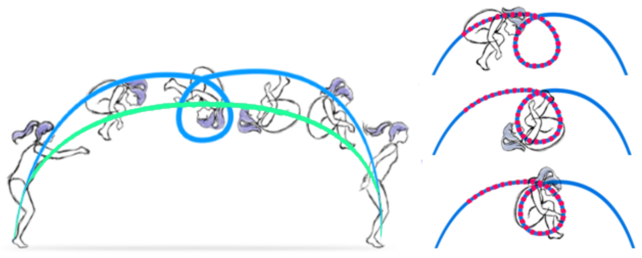

Figure 7: *In these artist sketches, the character's upper body matches the **shape** of the blue curve, between the two self-intersecting points (see the red dashed lines on the right). Note on the left how the character's global trajectory (green overlay) does not strictly match the sketched trajectory.*

To fulfill the dynamic shape-matching constraint between intersecting points while blending with the shapes at the landing and take-off, we blend between a path-following DLOA $\bar{x}_{dloa}$, defined over the looping region, and a bouncing DLOA $\check{x}_{dloa}$ defined from the takeoff and landing keyframes, resulting in an intermediate DLOA $\hat{x}_{dloa}$—that we finally correct for trajectory (to follow the green trajectory in Fig. 7) using eq.(7).

First, we detect singular points (indicating takeoff and landing) and self-intersections in-between. Let (u_0, u_1, u_2, u_3) be the four associated parameter values, where $c(u_0)$ is the takeoff point, $c(u_1) = c(u_2)$ is the self intersection point, and $c(u_3)$ is the landing point.

The path-following DLOA $\bar{x}_{dloa}$ is extracted using a continuous warping (eq. (4)) within the looping region of $c(u)$, i.e. between u_1, and u_2. The bouncing DLOA $\check{x}_{dloa}$ is defined as the interpolated keyframes extracted at takeoff and landing coordinates u_0 and u_3. With these two different DLOA motions, we perform a blending in order to go from the takeoff shape, to the rolling motion, and back to the landing shape, following a classic blending operation:

$$\hat{x}_{dloa}(s,t) = t\ \check{x}_{dloa}(s,t) + (1-t)\ \bar{x}_{dloa}(s,t), \qquad (8)$$

where t is equal to 1 at takeoff (influence interval for the first keyframe), then smoothly drops to zero, which sets the DLOA to $\bar{x}_{dloa}$ (path-following), before the influence of the second keyframe becomes non-zero and increases to 1. To perform this blending, we use our length-preserving interpolation equation (3), where t and $1-t$ in eq. (8) are replaced by their Hermite basis function analogs $B(t)$ in eq. (3).

Lastly, the resulting (intermediate) DLOA $\hat{x}_{dloa}$ needs to be corrected in order to prevent the backward motion we already mentioned. This is done by constraining the center of mass $\hat{m}(t)$ of $\hat{x}_{dloa}$ to follow the green trajectory in Fig. 7. We build the green trajectory $T(t)$ by interpolating the three points $c(u_0)$ (takeoff), $c(u_{air})$ and $c(u_3)$ (landing), with $u_{air} = (u_1+u_2)/2$. We then add the correction term $\Delta x(t) = T(t) - \hat{m}(t)$ to $\hat{x}_{dloa}$, where $\hat{m}(t)$ is the intermediate center-of-mass after the blending operation eq.(8); resulting in both the expected trajectory *and shape*—over time—as shown in Fig. 5.

These constructions show that different DLOAs can be easily blended over time. Next, we re-use and extend this methodology in order to allow the user to refine and extend the rough animation he just created.

5 Editing and refining animations

5.1 Over-sketching keyframes

A natural way for a user to edit an animation is to stop it and over-sketch one or several keyframes. Implementing this is straightforward when the DLOA was created using the keyframing approach in Section 4.1: the user selects or inserts a keyframe at a time t_k, and over-sketches the current line of action $x_{dloa}(s, t_k)$ displayed by the system. Time intervals are automatically re-set when a new keyframe is inserted. The user can edit timing by sliding the t_k values along a timeline.

This type of interaction is also supported when the DLOA is created from a space-time curve (STC) (Section 4.2). A further benefit in this case is that the STC can be used as a visual timeline where keyframe strokes can be directly sketched-over. As each point u_i in the STC maps to a time value t_i, we can identify the closest point u_* to the drawn stroke, providing us with a time t_* where the new keyframe is to be inserted. Sketching over the space-time curve greatly eases shape and motion coordination. However, since the DLOA now holds continuous motion, blending is required to take the new keyframe(s) into account.

Blending DLOAs. Let $[t_{k-1}, t_{k+1}]$ be the interval of influence of a new key-stroke (the user can edit this interval using the timeline). Let $x^1_{dloa}(s, t)$ be an existing DLOA on which is sketched a second stroke $x^2_{dloa}(s, t)$ defined as a constant keyframe shape. The goal is to transition from x^1_{dloa} to x^2_{dloa} over $[t_{k-1}, t_k]$, before transitioning back to x^1_{dloa} over $[t_k, t_{k+1}]$. This is done by using Equation (8), where $\bar{x}$ is replaced by x^1 and $\check{x}$ by x^2. A result is shown in Fig. 9, where the dragon's tail first follows a path, is then constrained by the sketched curve at t_k, and then goes back to the path-following.

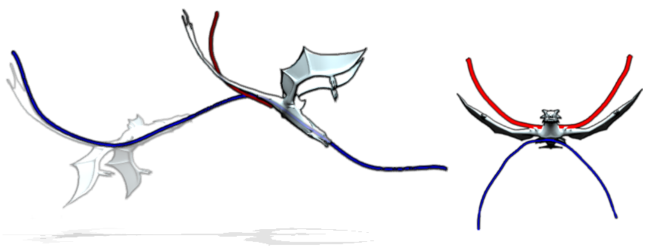
Figure 9: *The user sketches a line of action stroke on top of a path-following DLOA to alter the motion of the tail over a time interval. The path-following motion (a DLOA) blends with another DLOA, the static key frame sketched for the tail, over a time interval. Right: the user edits secondary lines onto a separate plane and view.*

Sketching secondary lines. Another type of refinement consists in animating secondary lines of action by sketching keyframes from other viewpoints. This can be used to animate secondary body-lines such as the wings of a dragon. To do this, the user chooses another viewpoint, plays the animation, stops it when desired and over-sketches a few key-LOAs for the secondary line. The secondary keyframes are thus easily synchronized with the main DLOA. They are interpolated using the method in Section 4.1 and can be played in loop if desired. See Fig. 9, right.

Note that the over-sketching method replaces the motion in an absolute way, i.e. if we re-draw the STC, the tail will remain oriented the same way. Another method is to treat the over-sketching as a displacement map layered on top of the coarse STC, which we do with wave refinements, described next.

5.2 Wave curves

A *wave curve* is a layered refinement enabling the animator to edit the current DLOA in a relative manner. It allows adding periodic wave motion to the DLOA, by composing it with a periodic displacement map. To illustrate this concept, let us consider the following scenario: instead of sketching a path and having a line of action follow it as in the path-following behavior (Section 4.2), we draw a curve and make the curve *move through* the line of action over time, as shown in the first column of Fig. 8. This type of behavior is often seen in spine-driven locomotion where a tail or flag follows an oscillating driving mechanism.

Combining the current DLOA and this wave motion by directly applying the previous blending method would completely replace the DLOA since the wave motion is periodic and covers the full time range. In contrast, we would like to combine them in a way that preserves the existing DLOA and allows subsequent editing of both the existing coarse DLOA (e.g. by re-drawing keyframes) and the wave refinement (see Fig. 8).

To enable this, we define the motion of the wave (i.e. its shape over time defined by angles $\theta(s,t)$) as relative—or as an offset $\Delta\theta(s,t)$—to a reference curve x_{ref}. This reference shape should be coarser and representative of the more complex wave motion. As a reference shape, we use the vector defined between the two end-points of the wave stroke $x_{ref} = c(1) - c(0)$, as it naturally provides a coarse representation of the wave. We compute the relative angle w.r.t. the reference axis as follows:

$$\Delta\theta(s,t) = \angle\left(x_{ref}, \frac{\partial c(w_\pi(s,t))}{\partial s}\right),$$

where w_π is a periodic warping function that evolves over time. Finally, the angle relative to the reference axis $\Delta\theta(s,t)$ is composed on top of an animated DLOA by *adding* the angle to the shape blending Eq. (3), which is done by setting its temporal basis function to $b_{wave}(t) = 1$. Fig. 8 depicts wave curves composed with keyframed DLOAs.

5.3 Twisting around a space-time curve

2D skeletal lines can hardly specify 3D twist. In hand drawings, the twist of characters is often established explicitly in the early stages of the drawing; that is, in the abstract form. Annotations such as lines or can-shape primitives shown in Fig. 11 are drawn over a line of action to establish the orientation. It is worth noting how the cans in Fig. 11 rarely conflict with the line's shape serving only as an extra dimension refinement.

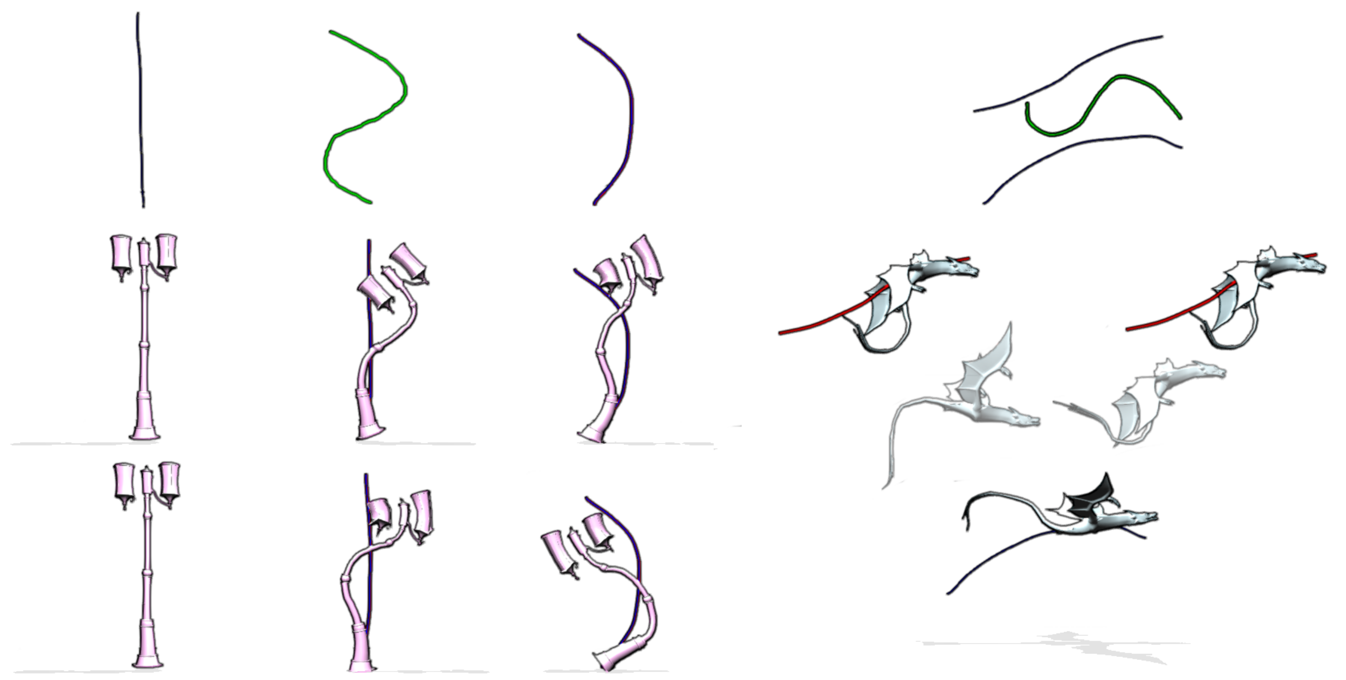

Figure 8: *First column: a straight static line. Second column: a periodic wave curve (green) has its transformation (local angles) defined relative to the straight line. Third column: the relative transformation is transferred to another (bent) line by adding its angle offsets to the bent line. This allows controlling a wave motion by redrawing a single stroke, or creating a complex flying locomotion by layering a wave curve on top of of a basic key framed line motion.*

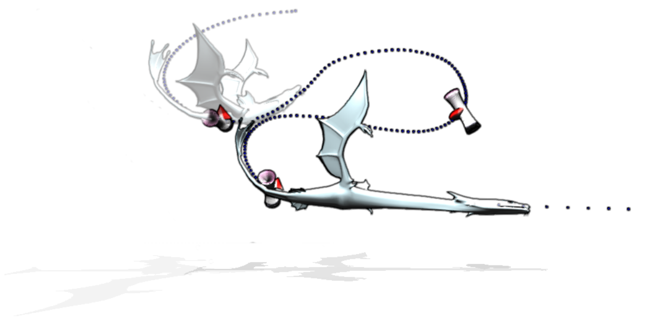

Figure 10: *The cans are specified at space-time coordinates along the space-time curves. In the bottom-right, we can see non-uniform stretching of the character as it gradually enters an area of the curve that was drawn faster.*

We allow the user to specify twist along the strokes, may they be key-LOAs (Section 4.1) or space-time curves (Section 4.2 as shown in Fig. 10). The cans control a single angle γ_i that rotates around the stroke. The user adds cans along the strokes, which are then interpolated into a smooth twist function $\gamma(s)$. Different key-LOA strokes over time define different $\gamma_j(s)$ to be interpolated, while adding cans to a STC directly defines $\gamma_j(s,t)$. We blend between all the active cans at a time t using the same Hermite based interpolation as in Equation (3). The smooth twist angle $\gamma(s,t)$ is then used along the dynamic line of action $x_{dloa}(s,t)$ to control the character's skeleton.

6 Matching a 3D skeleton to a DLOA

The 2D dynamic line (of action) $x_{dloa}(s,t)$ corresponds to the *shape* of a *projected* skeletal line abstraction for parts of the character's body such as: the spine, tail, wings, arms, or combinations of them. As in [Guay et al. 2013], we call this skeletal line the *body line*. The goal of this section is to match the shape of the 3D body line to the shape of the DLOA—over time.

In [Guay et al. 2013], a measure of similarity between a 2D line of action and a projected 3D skeletal line is introduced. They use this definition to formulate an optimization problem and solve for bone orientations, constrained to the viewing plane. Unfortunately, applying their method directly to the dynamic case fails as it does not ensure temporal coherence.

In this paper, we introduce a robust solution that solves for planar transformations of the bones. It follows a dynamic programming approach where, starting from the root of the chain, we solve *analytically* for the planar rotation (in the viewing direction) and length of a bone, transfer the quantities to the 3D bone, and then go down the chain; doings so for each chain constrained by the DLOA.

The bones match a line of action when their tangents (or directions) match in screen space [Guay et al. 2013], while a select bone touches the line (often the root or head). Hence the goal is for the projected direction of each bone to match the 2D direction of the line. In our case, the root is touching the line, while other parts may not be touching depending on the offsets present in the kinematic tree.

We assume there is a skeletal rig providing the bones Ω and the *body line* $\beta \subseteq \Omega$. We also assume the i^{th} bone positions $x_i(t)$ hold a corresponding DLOA parameter s_i.

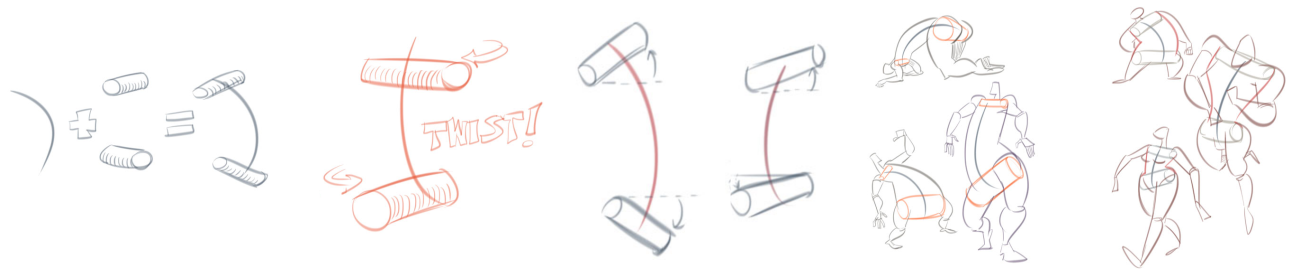

Figure 11: *The Two Cans Technique is a drawing tool often combined with the line of action to help specify the twist and relative depth of the character. We use this visual handle to twist our dynamic lines in various ways. The user manipulates the cans to twist the body part over the space-time curve. Illustration by ©Krishna M. Sadasivam, [Sadasivam 2012].*

To measure the dissimilarity between both lines, we first project both lines onto a single plane located at the root of the body line, and oriented in the viewing direction. We start by inverse projecting the 2D DLOA (red stroke in Fig. 12) onto the 3D plane. We cast rays along the DLOA $x_{dloa}(s_i, t)$ to recover the corresponding 3D positions denoted $z_i(t)$. We then project the bone positions $x_i(t)$ onto the same plane using the projection operator P (blue in Fig. 12).

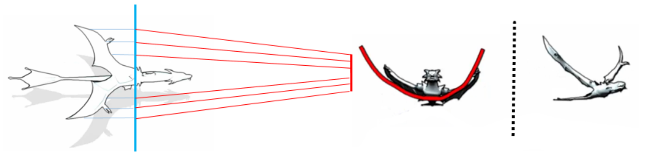

Figure 12: *Both the body line and the DLOA at time t are projected onto a plane located at the root of the body part. For sketching secondary lines such as the wings, the character's main line (e.g. the spine) remains in a fixed reference pose. Last image on the right is the pose at the time of the sketched DLOA key frame.*

For the character's skeleton, we denote the lengths of the bones as $l_i(t)$, the relative orientations $q_i(t)$, and the absolute orientation $Q_i(t)$. To remove depth ambiguities, we assume a viewing plane constraint. Hence, by construction, we are looking for a single (angle $\theta_i(t)$) rotating in the viewing direction v_{dir}, i.e. an orientation $Q(\theta_i(t), v_{dir})$, that rotates the bone as to match its corresponding line direction (see Fig. 13).

For each bone i, we compute the angle $\theta_i(t)$, and then recover the bone's relative orientation w.r.t. to the parent orientation $Q_{i-1}(t)$:

$$\begin{aligned} \theta_i(t) &= \angle(Px_{i+1}(t) - Px_i(t), z_{i+1}(t) - z_i(t)), \\ q_i(t) &= Q_{i-1}^{-1}(t) Q(\theta_i(t), v_{dir}) Q_i(t). \end{aligned}$$

For the length, we seek to preserve the length ratio between the initially projected skeleton bone direction, and the length of the non-projected bone direction. Given the initial *projected* bone direction length $a_i(0) = \|Px_{i+1}(0) - Px_i(0)\|$, the initial bone direction length $l_i(0) = \|x_{i+1}(0) - x_i(0)\|$, and the sketched bone direction length at time t, $a_i(t) = \|z_{i+1}(t) - z_i(t)\|$, we have the relation:

$$\frac{a_i(0)}{l_i(0)} = \frac{a_i(t)}{l_i(t)},$$

and thus by extracting $l_i(t)$, we obtain:

$$l_i(t) = \frac{\|Px_{i+1}(t) - Px_i(t)\|}{\|z_{i+1}(0) - z_i(0)\| \|x_{i+1}(0) - x_i(0)\|}.$$

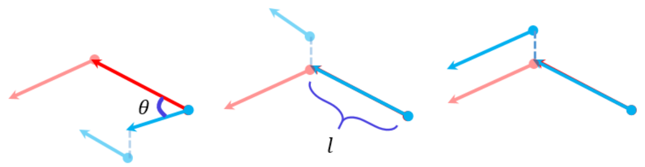

Figure 13: *Our skeletal line matching: blue and red are the projected bones and DLOA directions. Dashed lines are offsets in the character's kinematic tree. For each bone starting from the root, we compute the angle, then adjust its length, and we repeat for each bone down the chains.*

To prevent drift from accumulating, we initialize the orientations and lengths at each frame with an initial pose, i.e. $q_i(t) = q_i(0)$, and $l_i(t) = l_i(0)$.

Including the twist: We incorporate the smooth twist angle $\gamma(s, t)$ from Section 5 by computing the differential twist angle of each segment $\Delta\gamma_i(t) = \gamma(s_i, t) - \gamma(s_{i-1}, t)$, and applying a local rotation around the local bone direction $\bar{x}$:

$$\hat{q}_i(t) = q_i(t) Q(\Delta\gamma_i(t), \bar{x}).$$

Secondary parts are automatically taken into account due to the hierarchical structure of the skeleton. For instance the wings are attached to the spine of the dragon in Fig. 9, and inherit the parent's (spine) orientation, including the twist. Note, the length of parent bones is not propagated to children bones with this formulation.

7 Experimental Results

Experimental results shown in the paper and the accompanying video were created using our prototype implementation of the space-time sketching tools described in Sections 4, 5 and 6. In this section, we briefly describe our prototype and its user interface, then describe a user study we conducted with both novice and expert animators to assess the usability of our space-time sketching tools.

7.1 System description

The user is presented with a canvas for drawing keyframes and space-time curves, and a timeline for changing their timing. By default the user draws keyframes. A special key is pressed to sketch a space-time curve. We automatically classify the space-time curves as bouncing, rolling or path-following based on their geometry. The user draws over the path to automatically insert a keyframe, whose timing is retrieved from the path.

The user is first presented with a character in a reference pose and starts sketching. When the user selects a body part by pressing a key and drawing over the reference pose, the system snaps a plane onto the root of the body part, and is ready to animate the character using our method in Section 6. At this point, the sketches are inverse-projected onto the *action* plane, and linked to a body part.

Because the 2D sketch is decoupled from the 3D counter-part, the user can apply same line motion to the body parts of a different character, given the same viewpoint. To edit secondary parts of the body, the user can add a second sketching panel, and rotate the camera to a side view. The secondary sketches are freely sketched and the user then selects the body part which will snap the plane onto the root of the body part. The system will automatically snap the secondary sketch to the position of the primary sketch. This is done by taking the point at the intersection of the secondary plane and the primary DLOA and assigning it to the root of the secondary line. In our system, the user can chose to see the constrained parent-child sketches or the free-floating sketch; which is the default behavior.

Many of the sketching tools described in this paper evolved as a consequence of user interaction and feedback. For instance, we initially offered only a basic keyframing interface for creating animations such as the one in Fig. 1 and observed that users were attempting to draw paths. After we introduced the path-following STC, users started drawing abstractions such as bounce and roll doodles—expecting our system to understand. We asked people to draw over a dragon flying animation, resulting in the wave shape that we implement as a periodic path to create the animation in the video and Fig. 8.

7.2 Evaluation

As an initial evaluation of our approach, we ran an informal user study with eight participants with varying levels of expertise in animation, ranging from complete beginners to professionally trained animators. We asked each participant to reproduce three short animations, first by keyframing lines of action ; then again by sketching space-time curves. The target animations videos were not made with our system. Animation (1) is a path-following motion of a dragon with a lot of twisting. Animation (2) is a flying locomotion of the same dragon. Animation (3) is a bouncing animation of a lamp post.

Participants had never seen or used our system. They were each given twenty minutes to learn the basic keyframing and space-time sketching tools in our system: the participant was presented with the basic operations of adding keyframes to a timeline; sketching the strokes; manipulating the twist cans; and editing secondary parts (e.g. the wings). Then we showed each target animation and asked the participant to reproduce it, first with keyframing and then with space-time curves. In each case, the participants were allowed to add twist cans. After completing each task, participants were asked to self-evaluate the quality of their results on a scale from 0 to 5. Note that our experiment did not control for order effects. In each case, we computed the time spent to complete the task, the number of pen clicks, and the number of entities (strokes and cans) used to obtain the final animation. The results are as follows:

Animation	time	entities	clicks	score
dragon1	353 s.	13	95	1.6
dragon2	321 s.	9	69	2.1
lamp	260 s.	11	55	3.3

Table 1: *User response to keyframing: time spent, number of keyframes and twist cans, number of clicks, and self-evaluation scores per participant.*

Animation	time	entities	clicks	score
dragon1	139 s.	4	34	4.2
dragon2	229 s.	2	40	3.6
lamp	161 s.	5	48	4.5

Table 2: *User response to space-time curves (STC): time spent, number of curves and twist cans, number of clicks, and self-evaluation scores per participant.*

We can see that space-time curves help the users achieve results quicker than keyframes in all cases. We observed average speed-up factors as high as 20 between sketching keyframes and sketching path-following curves. The effect is less visible when reading the total time because participants can spend more time on refining and twisting the curves. Space-time curves also receive significantly higher self-evaluation scores for all three animation targets. While some of this may be explained by order effects, we feel the effects are strong enough to warrant further study.

We observed novice users tended to be discouraged quickly by keyframing. Most of the time, they were not able to finish the task. Some participants gave up after trying for five minutes or more. In contrast, our space-time curves allowed all participants to quickly obtain results that matched their desires and expectations, encouraging them to spend more time polishing their animations. The twisting tool was used consistently and comfortably by most participants. Some participants had difficulty using the wave curves and found them counter-intuitive, although this may be an issue with the system's interface and our training procedure.

Advanced users performed slightly better with the keyframing tools than novice users, but still preferred to use the space-time curves, as they obtained their results quicker and more naturally. They also liked the fact that they had instantaneous results with space-time curves followed by refinement and twisting.

8 Limitations and future work

Using sketching to freely create and refine 3D animations is a difficult problem. In this paper, we used sketching to specify 3D animations from a given viewpoint, therefore limiting ourselves to projective control. We used the notion of a moving line of action for abstracting a part of the character's body—allowing the same moving line to be applied to different body parts or characters.

Although possible, we showed that defining a DLOA only from keyframes (temporal slices) is not the best way. Taking an analogy from geometric modeling, this method is similar to shape design through sketching slices: getting a smooth result is not easy, and we can only see the result when all the slices have been defined. In contrast, our new space-time abstraction for sketching motion (the STC) enables an animator to initialize a full coordinated motion with a single stroke—providing immediate display and allowing for rapid refinement.

This method brings several limitations. We focused on the motion of a single line and illustrated our concept with simple character morphologies (lamp, cactus and dragon). It would require more work to drive the motion of a multi-legged figure (humanoid, or dog) from a single moving line, which is explored in recent work [Guay et al. 2015].

We defined three specific ways of creating a DLOA from a space-time curve: path-following and two richer mechanisms for extracting coordinated motion and trajectory from intuitive strokes. The resulting motion can then be refined, which increases the range of possible results. While we can detect which behavior to adopt between the bouncing, rolling and path-following based on the geometric features found in the strokes, we cannot automatically differentiate with wave curves and other unit keyframes.

We focused on planar movement plus out-of-plane secondary motions and twist, since many motions meant to be seen by viewers are in fact planar. While we think our DLOA models can be directly extended to 3D curves, the actual editing of 3D curves remains a challenge. Another route to 3D motion could be to use our new 2D dynamic line constraints along data-driven motion priors in a space-time optimization framework [Min et al. 2009]. However, one of the main reasons we devise free-form tools for character animation is for the ability to produce arbitrary movements that may include squash and stretch effects.

9 Conclusion

We introduced the new concept of space-time sketching for free-form animation of 3D characters. Thanks to this new concept, a coarse, but coordinated animation—possibly including squash and stretch—can be drafted with a single stroke. Other strokes and controls can then be added, enabling the user to progressively sculpt and refine motion.

The animation is designed by sketching in 2D from viewpoints of interest: it defines a dynamic line of action used as a projective constraint that robustly drives the 3D character. The resulting independence between animation control and the 3D character model enables the user to simultaneously edit the 3D model during the design stage. In future work, we plan on studying mechanisms for swapping the selected body line over time, in order to extend our formalism to the design of bipedal and quadrupedal motion—driven by a single moving line abstraction.

Acknowledgements

We thank Antoine Begault for help with code, Estelle Charleroy with video editing and Deidre Stuffer with text editing. We also thank the anonymous reviewers for their useful comments and suggestions. This work was partially funded by the ERC advanced grant no. 291184 *EXPRESSIVE* from the European Research Council (ERC-2011-ADG 20110209) and NSF award NRI-1208632.

References

ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, 157–164.

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

DAVIS, J., IGARASHI, M., CHUANG, E., POPOVIC', Z., AND SALESIN, D. 2003. A sketching interface for articulated figure animation. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 320–328.

DAVIS, R. C., COLWELL, B., AND LANDAY, J. A. 2008. K-sketch: A 'kinetic' sketch pad for novice animators. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, 413–422.

DONTCHEVA, M., YNGVE, G., AND POPOVIĆ, Z. Layered acting for character animation. *ACM Trans. Graph. 22*, 3, 409–416.

GLEICHER, M. Motion path editing. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, I3D '01, 195–202.

GUAY, M., CANI, M.-P., AND RONFARD, R. 2013. The line of action: An intuitive interface for expressive character posing. *ACM Trans. Graph. 32*, 6, 205:1–205:8.

GUAY, M., RONFARD, R., GLEICHER, M., AND CANI, M.-P. 2015. Adding dynamics to sketch-based character animations. In *proceedings of the Symposium on Sketch-Based Interfaces and Modeling*, SBIM '15.

HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., AND O'BRIEN, J. F. 1995. Animating human athletics. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, 71–78.

IGARASHI, T., KADOBAYASHI, R., MASE, K., AND TANAKA, H. 1998. Path drawing for 3d walkthrough. In *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*, UIST '98, 173–174.

KHO, Y., AND GARLAND, M. 2005. Sketching mesh deformations. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, 147–154.

LASZLO, J., VAN DE PANNE, M., AND FIUME, E. 2000. Interactive control for physically-based animation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, 201–208.

MIN, J., CHEN, Y.-L., AND CHAI, J. 2009. Interactive generation of human animation with deformable motion models. *ACM Trans. Graph. 29*, 1, 9:1–9:12.

NEFF, M., ALBRECHT, I., AND SEIDEL, H.-P. 2007. Layered performance animation with correlation maps. *Comput. Graph. Forum 26*, 3, 675–684.

ÖZTIRELI, A. C., BARAN, I., POPA, T., DALSTEIN, B., SUMNER, R. W., AND GROSS, M. 2013. Differential blending for expressive sketch-based posing. In *Proceedings of the 2013 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '13.

SADASIVAM, K. M. 2012. Learn to draw dynamic comic characters using the "two cans" technique! *Blog post*.

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, 151–160.

SHIRATORI, T., MAHLER, M., TREZEVANT, W., AND HODGINS, J. K. 2013. Expressing animated performances through puppeteering. *IEEE 3DUI*, 59–66.

THORNE, M., BURKE, D., AND VAN DE PANNE, M. 2004. Motion doodles: an interface for sketching character motion. *ACM Trans. Graph. 23*, 424–431.

YIN, K., LOKEN, K., AND VAN DE PANNE, M. 2007. Simbicon: Simple biped locomotion control. *ACM Trans. Graph. 26*, 3.