Tangent-Space Optimization for Interactive Animation Control

LOÏC CICCONE, ETH Zürich
CENGIZ ÖZTIRELI, DisneyResearch|Studios
ROBERT W. SUMNER, DisneyResearch|Studios and ETH Zürich

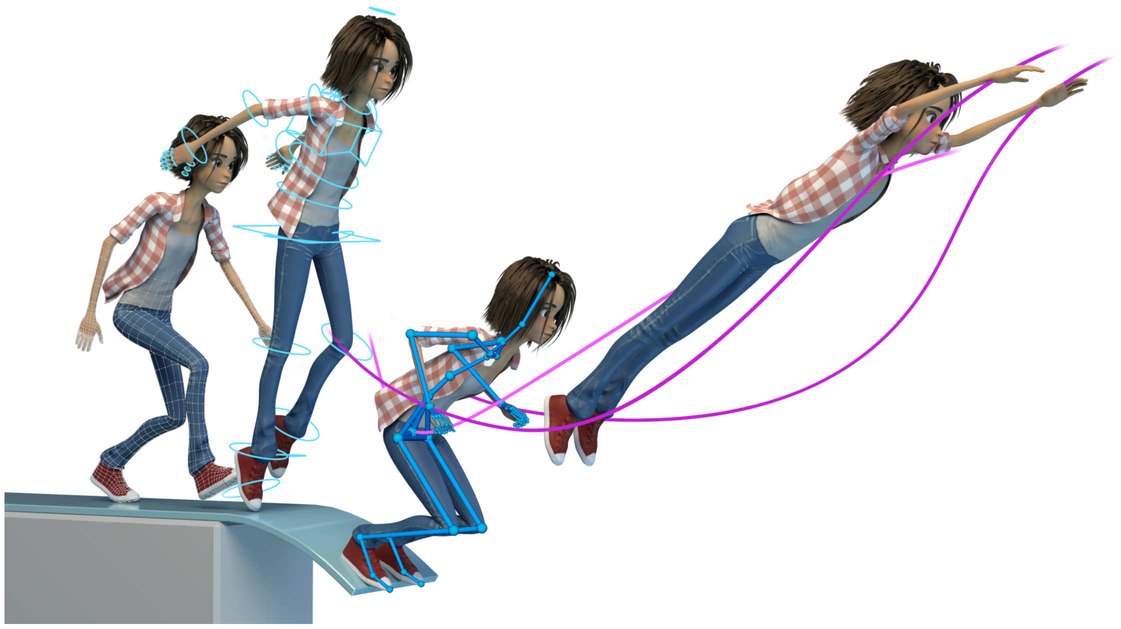


Fig. 1. Left: In traditional animation the character's deformations are driven by rig controls (light blue), which provide fine control but require a granular interaction. Middle: Our system provides an armature (dark blue) that coordinately drives the rig elements for a more natural and flexible character manipulation. Right: We introduce a curve representation (purple) for easily controlling the interpolation between key poses without adding any keyframe.

Character animation tools are based on a keyframing metaphor where artists pose characters at selected keyframes and the software automatically interpolates the frames inbetween. Although the quality of the interpolation is critical for achieving a fluid and engaging animation, the tools available to adjust the result of the automatic inbetweening are rudimentary and typically require manual editing of spline parameters. As a result, artists spend a tremendous amount of time posing and setting more keyframes. In this pose-centric workflow, animators use combinations of forward and inverse kinematics. While forward kinematics leads to intuitive interpolations, it does not naturally support positional constraints such as fixed contact points. Inverse kinematics can be used to fix certain points in space at keyframes, but can lead to inferior interpolations, is slow to compute, and does not allow for positional contraints at non-keyframe frames. In this paper, we address these problems by formulating the control of interpolations with positional constraints over time as a space-time optimization problem in the tangent space of the animation curves driving the controls. Our method has the key properties that it (1) allows the manipulation of positions and orientations over time, extending inverse kinematics, (2) does not add new keyframes that might conflict with an artist's preferred keyframe style, and (3) works in the space of artist editable animation curves and hence integrates seamlessly with current pipelines. We demonstrate the utility of the technique in practice via various examples and use cases.

CCS Concepts: • **Computing methodologies** → **Animation**; *Graphics systems and interfaces.*

Additional Key Words and Phrases: Interpolation, Inverse kinematics

Authors' addresses: Loïc Ciccone, ETH Zürich; Cengiz Öztireli, DisneyResearch|Studios; Robert W. Sumner, DisneyResearch|Studios, ETH Zürich.

1 INTRODUCTION

Character animation software provides the tools, algorithms, and interfaces that artists use to breath life into animated characters. Contemporary software uses a keyframing metaphor inspired by classic hand-drawn animation: Artists pose characters at selected keyframes and the frames in-between are automatically interpolated. While typical software offers many controls and methods for posing, the tools available to adjust the result of the automatic inbetweening are comparatively rudimentary. Artists must manually edit the parameters of interpolation splines for individual animation variables. In this pose-centric view, the least burdensome way to adjust the interpolated movement is to insert additional keyframes, which displeasingly complicates the parametrization of animation curves and impedes further refinement. As a result, artists spend a tremendous amount of time posing and setting keyframes (as exposed in the prestudy of Kytö et al. [Kytö et al. 2017]) in order to achieve a fluid animation that obeys to the principles of movement and timing characteristic of high-quality animation [Johnston and Thomas 1981; Whitaker and Halas 2013].

For articulated movement, the situation is further complicated. The artist is given the choice of either manipulating *angles* with forward kinematics (FK) or *positions* with inverse kinematics (IK). However, the choice made when posing the character will also determine the nature of the interpolation between these poses: FK results in smooth arcs while IK produces linear movement, especially useful for contact points. This forces the artist to carefully plan when to use FK and IK. Furthermore, only identical kinematic configurations can be interpolated since existing systems assume that IK chains have the same bases and end-effectors. This forces riggers to define fixed IK chains on the character and prevents state-of-the art techniques in flexible IK definition to be used in a production environment. In addition, IK-based controls slow down the rig evaluation, so a limited set is typically placed on the characters, which restricts the freedom of interaction. For example, artists rarely put IK on fingers due to the complexity of the hand structure.

Our work addresses these problems with a novel tangent-space optimization framework and a temporal interface for articulated movement that allows artists to intuitively adjust in-betweenings (Fig. 2). The key feature of our technique is that the optimization solves for the tangents of interpolation at existing keyframes. As a result, it does not add any complexity to the animation curves and is non intrusive to the artist workflow. The optimization supports user-provided or character-implied constraints such as joint

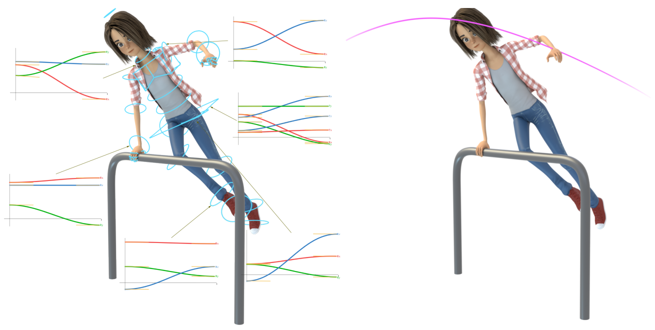


Fig. 2. Left: Traditional interface for editing interpolations — the artist must manually edit the tangents of multiple attributes' animation curves. Right: Our interface — the artist directly manipulates the trajectory of an element, and our solver optimizes for the tangents that match the user inputs.

angle limits and stiffness for more natural deformations. Our system works on top of typical character controls and does not add other complexities or structures to drive the animation. Furthermore, our method provides an abstraction on top of posing and animation that completely eliminates the need for fixed IK chains.

We illustrate how our framework improves current animation pipelines with various examples and use-cases with both professional artists and novice users. In summary, we make the following contributions:

- A formulation of interpolation as optimization in the tangent space of rig controls with given positional, joint angle, and stiffness constraints. This leads to real-time interpolation control, which enables artists to work with fewer keyframes and cleaner animation curves.
- An interactive interface that utilizes the optimization framework and allows editing space-time curves for precise locations of controls during interpolations. This leads to a more fluid and natural interaction than with merely adjusting animation curves for transformations.
- A system that alleviates the need of fixed IK chains for interpolation, and thus unlocks the use of state-of-the art posing techniques in production environments.

2 RELATED WORK

IK based interpolation. As elaborated, professional animators use the concept of keyframing when animating characters. Despite being the most common approach for authoring motions, it provides a very indirect control on interpolations where animators cannot specify positional constraints. IK controllers, which have been widely studied in computer graphics and robotics research [Aristidou et al. 2018], allow fixing the position of some end effectors at keyframes, which are then interpolated to generate the motion. However, IK based interpolation requires having a fixed constraint configuration (fixed bases and end effectors), leads to inferior linear interpolations of positions compared to artistic arcs that naturally arise when interpolating angles, and leaves animators with the difficult choice of when to switch between IK and FK for interpolation. In most cases, it is very challenging to find the right sequence of FK/IK for the desired animation results. In contrast, our technique completely abstracts away the choice of FK/IK, and augments the current animation workflow by allowing the user to simultaneously edit key poses and their interpolations in the comfort of a single viewport.

Systems for posing. Many works explored different interfaces for crafting poses, such as the pin-and-drag metaphor [Shi et al. 2007; Yamane and Nakamura 2003], the use of reference poses [Choi and Lee 2016; Wei and Chai 2011], sketch abstractions [Guay et al. 2013; Hahn et al. 2015; Öztireli et al. 2013] or even physical devices [Glauser et al. 2016; Yoshizaki et al. 2011]. However, none of these techniques tackles the challenge of controlling the interpolation between the crafted poses at keyframes. The mentioned methods apply transformations on FK values of the character controls, and hence are not suitable to constraint positions in the in-between frames (such as for contacts). This makes them impractical in the contemporary professional workflow. The curve representation and algorithm we propose enable the positional control of interpolations, even when the motion is driven by FK controls. This finally permits the integration of such advanced posing techniques into production environments.

Space-time constraints. In 1988, Witkin and Kass introduced space-time constraints [Witkin and Kass 1988]. In this paradigm, a user can specify high-level spatial and temporal constraints and the motion is produced automatically via non-linear optimization. This inspired a lot of further research, including many works focusing on the animation of articulated characters for providing a user interface [Cohen 1992], transitions between motion segments [Rose et al. 1996], edition of existing animations [Gleicher 1997], and even motion retrieval in a large database [Min et al. 2009]. Although they allow for simple editing of motion via high level properties, they poorly integrate in the professional animation workflow because of the lack of fine control. The user needs to indirectly specify motion by balancing hard and soft terms of the cost function to achieve a desired animation. Fine-tuning the result is also laborious as most often the parametrization of the motion curves is altered based on the computational needs of the system. Furthermore, with space-time constraint techniques, the set of achievable motions is restrained by the physical formulations or the library of movements the optimization is based on.

Automatic interpolation. A particular category of space-time constraints techniques seek to automatically generate believable transitions by relying on simulations, databases or heuristics. Important work with probabilistic models of human motion was used for filling gaps of animations [Chai and Hodgins 2007; Lehrmann et al. 2014; Wang et al. 2008], collision detection was used to correct default interpolations [Nebel 1999], and more recent research used deep learning to generate animations that interpolate key poses [Harvey and Pal 2018; Zhang and van de Panne 2018]. To allow for more controllability, Koyama and Goto [2018] provide mathematical formulations for the control of an optimization based on physically inspired energy terms. Their work (OptiMo) has the additional benefit of modifying tangents instead of adding keyframes, which is also our motivation. However, their approach is fundamentally different from ours: While we aim at real-time control of interpolations with an intuitive interface directly in the viewport, Koyama and Goto provide an indirect and offline motion manipulation tool via sliders and graph editors. Furthermore, OptiMo is demonstrated for relatively simple animations of singled-chained characters' sections while our tool integrates well with the rest of the workflow. We provide the animator with full control without making assumptions about their intentions (e.g. obtaining a physically realistic movement).

Differential manipulation. Gleicher and Witkin [1991] introduced the concept of differential manipulation in order to provide an interface for directly manipulating objects without having to edit individual parameters. Conceptually our approach is similar to their work, yet applied to a different domain, since we allow directly manipulating in-betweenings through a graphical interface instead of editing individual tangents of interpolation.

Space-time curves. Some previous works have explored the concept of space-time curves for authoring or editing motions. Guay et al. [2015] enable the creation of a full character motion using a single stroke and refine it using additional types of stroke edits. Unfortunately, their strokes are designed around specific types of motions such as bouncing, rolling, and waving, and their work only demonstrates a few simple characters (e.g. no bipeds nor quadrupeds). Choi et al. [2016] allow editing motion via a sketch-based interface. Using screen-space strokes to define 3D movements, their technique ends up with an underconstrained problem and thus needs to make assumptions on the user's intentions. Ciccone et al. [2017] propose to edit a cyclic motion through the manipulation of a closed 3D spline, but their work exploits the IK chains already present on the rig, which limits the set of trajectories that can be edited on the character. Additionally, those techniques require control on each frame of the animation, which is equivalent to having one keyframe per frame. This does not integrate well with the artists' workflow.

3 APPROACH

3.1 Background and Problem Formulation

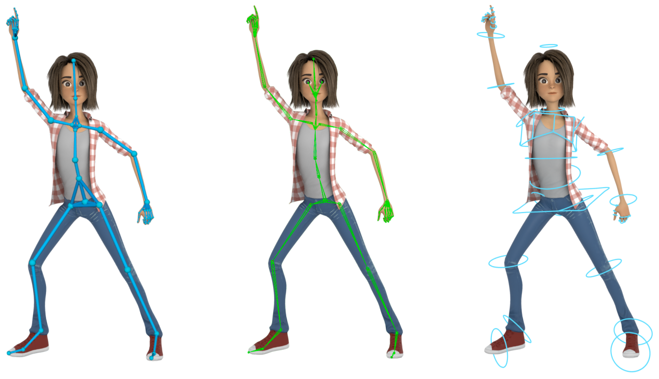


Fig. 3. Left: Our generic abstraction of character controls. It represents the structure that the artist is animating, which could be of any type: Skeleton (middle), rig controls (right) or others.

Our system starts with any rig structure. Some examples are graphs of controls for rotation and translation, more complex rigs with advanced controls, or a skeletal structure with joints (Fig. 3). From there, the user can directly start animating either with traditional tools, or with our system.

Once some keyframes are set, the *motion curves* corresponding to a point's path in time can be visualized (Fig. 4, left) by clicking on any point on the control structure. The motion curve also displays orientations (middle) and spacing of frames, i.e. timing (right). The trajectory, orientation and timing can all be altered by respectively dragging, rotating or scaling the motion curve at a location corresponding to the chosen point at a particular frame. In this paper, we define *state* as the positional and orientational configuration of a point at a specific time. Alterations specified by the user on a point's state are combined with further constraints, such as pinned points (motion curves that the user does not want to be altered) and contact points, in order to craft the resulting motion interactively. Fig. 5 shows a step of an example editing process: The user sets the right hand as contact, pins the trajectory of the hip, and manipulates the trajectory of the left arm as represented with a motion curve.

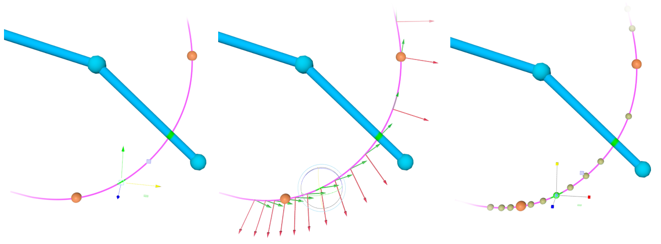


Fig. 4. Motion curve representation that allows to edit the trajectory (left), orientations (middle) and timing (right) during interpolations. The orange spheres represent keyframes, i.e. points that will not be altered by the modifications.

Although the manipulations and constraints described above give the user enough degrees of freedom (DoF) and precise control, we observed that it is often difficult to get character specific deformations as all the above controls are agnostic to the character being animated. This is a common problem also in previous FK/IK based systems. We thus propose to further impose character specific properties. Angle limits are set in order to forbid undesired configurations (such as turning the elbow backward). Angle stiffness defines the resistance of joints to rotations; for example, on a human character, artists would usually set a lower stiffness on the shoulder than on the clavicle because the latter is less involved in arms' movements.

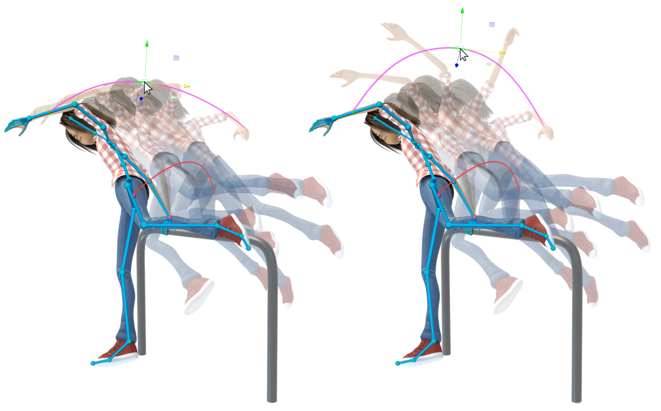


Fig. 5. Manipulation of the interpolation. The user edits the motion curve of the left arm, while ensuring that the trajectory of the hip (red curve) and the right hand (which is a contact) will not be altered.

As in most animation systems, we assume that each control in the underlying rig structure is parameterized, and each parameter is controlled by a different animation curve (Fig. 6). Although our formulation supports arbitrary animation curves, we will assume cubic Bezier curves, since they are heavily used in practice. Once an artist defines keyposes, each parameter of each controller is thus interpolated with a cubic Bezier curve that defines the motion. Using traditional systems, if the resulting animation is not satisfactory, editing the interpolated movement requires to manually edit the tangents of every Bezier curve, a cumbersome process that requires a lot of effort and time to get the desired animation.

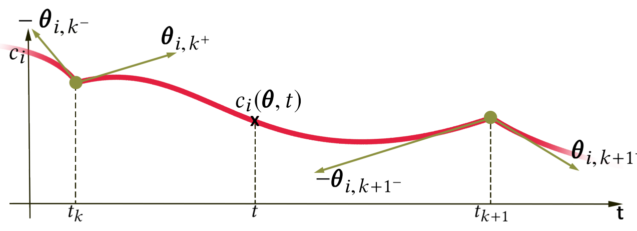

$-\boldsymbol{\theta}_{i,k^-}$
$\boldsymbol{\theta}_{i,k^+}$
c_i
$c_i(\boldsymbol{\theta},t)$
$-\boldsymbol{\theta}_{i,k+1^-}$
$\boldsymbol{\theta}_{i,k+1^+}$
t_k
t
t_{k+1}
t

Fig. 6. Illustration of the introduced variables.

We denote all tangents of all such animation curves with the vector $\boldsymbol{\theta}$, and all animation curves with $\mathbf{c}(\boldsymbol{\theta},t)$. At each editing step, the user can click on a point on the control structure, and change the parameters (position, orientation, and scale) at that point interactively. We denote the state of this point with $\mathbf{s}(\mathbf{c}(\boldsymbol{\theta},t))$, and the new one interactively specified by the user with $\mathbf{s}'(t)$. The goal is then to compute a new set of tangent values $\boldsymbol{\theta}'$ in order to reach $\mathbf{s}'(t)$ (i.e. we want $\mathbf{s}(\mathbf{c}(\boldsymbol{\theta}',t)) = \mathbf{s}'(t)$) while satisfying other constraints, as we will detail in the next section.

3.2 Tangent Space Optimization

As we would like to formulate the problem in terms of the tangents $\boldsymbol{\theta}$, we first express all changes, and in particular $\Delta\mathbf{s}(\mathbf{c}(\boldsymbol{\theta},t)) := \mathbf{s}(\mathbf{c}(\boldsymbol{\theta}',t)) - \mathbf{s}(\mathbf{c}(\boldsymbol{\theta},t))$ in terms of changes in the tangents. This can be easily carried out by a first order approximation:

$$\begin{aligned}\mathbf{s}(\mathbf{c}(\boldsymbol{\theta}',t)) &\approx \mathbf{s}(\mathbf{c}(\boldsymbol{\theta},t)) + \mathbf{J}_s(\mathbf{c}(\boldsymbol{\theta},t))(\boldsymbol{\theta}'-\boldsymbol{\theta})\\ \Delta\mathbf{s}(\mathbf{c}(\boldsymbol{\theta},t)) &\approx \mathbf{J}_s(\mathbf{c}(\boldsymbol{\theta},t))\Delta\boldsymbol{\theta},\end{aligned} \tag{1}$$

where $\mathbf{J}_s$ is the Jacobian matrix that stacks the derivatives of $\mathbf{s}$ with respect to the tangents $\boldsymbol{\theta}$. This is the starting point of most IK formulations, and in our case a well justified approximation as the user will incrementally alter the motion curves. The Jacobian can be further factorized into

$$\mathbf{J}_s(\mathbf{c}(\boldsymbol{\theta},t)) = \frac{\delta\mathbf{s}(\mathbf{c}(\boldsymbol{\theta},t))}{\delta\mathbf{c}(\boldsymbol{\theta},t)}\frac{\delta\mathbf{c}(\boldsymbol{\theta},t)}{\delta\boldsymbol{\theta}}. \tag{2}$$

This form of $\mathbf{J}_s$ is useful as the first term on the right hand side is typically simple. Denoting the ith component of $\mathbf{c}$ with c_i, and the position and orientation of $\mathbf{s}$ with $\mathbf{s}_P$ and $\mathbf{s}_O$ respectively, if c_i defines a translation, $\frac{\delta\mathbf{s}_P}{\delta c_i} = \mathbf{v}$ and $\frac{\delta\mathbf{s}_O}{\delta c_i} = \mathbf{0}$, or if c_i defines a rotation, $\frac{\delta\mathbf{s}_P}{\delta c_i} = \mathbf{v}\times(\mathbf{s}_P - \mathbf{r})$ and $\frac{\delta\mathbf{s}_O}{\delta c_i} = \mathbf{v}$, where $\mathbf{v}$ is the unit vector pointing along the translation or rotation axis, and $\mathbf{r}$ is the rotation center. The second term in the Jacobian factorization is more involved, and often not possible to get analytically. We thus approximate it with finite differences.

For simplicity, we will drop $\mathbf{c}$ and simply write $\mathbf{s}(\boldsymbol{\theta},t)$ and $\mathbf{J}_s(\boldsymbol{\theta},t)$ for the rest of this section.

Energy terms. Given this linear approximation, we can now define three important quadratic energies. The first one aims at making the solution $\mathbf{s}(\boldsymbol{\theta}',t)$ stay as close as possible to the target $\mathbf{s}'(t)$ interactively provided by the user on the motion curve (as in Fig. 5):

$$E_m = \left\|\Delta\mathbf{s}'(t) - \mathbf{J}_s(\boldsymbol{\theta},t)\Delta\boldsymbol{\theta}\right\|^2, \tag{3}$$

where $\Delta\boldsymbol{s}'(t) = \boldsymbol{s}'(t) - \boldsymbol{s}(\boldsymbol{\theta},t)$. Next, to avoid abrupt deformations when manipulating a trajectory, we add an energy term to minimize the changes in tangents:

$$E_d = \|\mathbf{D}\Delta\boldsymbol{\theta}\|^2 . \tag{4}$$

Here, $\mathbf{D}$ is a diagonal matrix that stores stiffness parameters per tangent component, as defined in section 3.1. Finally, the user can break some tangents, meaning that the animation curve for certain parameters can consist of multiple segments of Bezier curves that are not C^1. For such cases, we add an energy term that forces the tangents at consecutive segments to stay close in order to have an as smooth as possible curve:

$$E_b = \left\|\mathbf{T}^+\boldsymbol{\theta} - \mathbf{T}^-\boldsymbol{\theta}\right\|^2 , \tag{5}$$

where the matrices $\mathbf{T}^+$ and $\mathbf{T}^-$ select tangent pairs corresponding to the same keyframe of the same curve, but on different Bezier segments.

Pins and Contact Constraints. In addition to the interactively modified motion curves, the user can specify pins and contact constraints on the structure. This is important to achieve and interpolate the poses as desired. The pinned points' trajectories and orientations — i.e. their state denoted by $\mathbf{s}_j(\boldsymbol{\theta}, t)$ — should not be altered by the modifications on $\boldsymbol{\theta}$. Therefore, we have the desired states $\mathbf{s}'_j(t) = \mathbf{s}_j(\boldsymbol{\theta}, t)$, at all times t. We sample the time into a set of frames, which gives us a constrained state per pin and per frame, that we stack into the vector $\boldsymbol{\rho}(\boldsymbol{\theta})$. The target states for these points are similarly stacked into $\boldsymbol{\rho}'$. We thus require that $\boldsymbol{\rho}' = \boldsymbol{\rho}(\boldsymbol{\theta})$. We elaborate on the sampling of time in section 3.3.

Other points can be set as contact between two keyframes t_k and t_{k+1}, e.g. feet on the floor. This means that those points should not move between the two specified keyframes. Specifically, we have $\mathbf{s}_j(\boldsymbol{\theta}, t_k) = \mathbf{s}_j(\boldsymbol{\theta}, t_{k+1})$, and want $\mathbf{s}'_j(t) = \mathbf{s}_j(\boldsymbol{\theta}, t_k)$, $\forall t \in [t_k, t_{k+1}]$. Once again, we sample the time between t_k and t_{k+1} and append the set of constrained states $\mathbf{s}_j(\boldsymbol{\theta}, t)$ to the vector $\boldsymbol{\rho}(\boldsymbol{\theta})$, and the set of corresponding desired states $\mathbf{s}'_j(t)$ to $\boldsymbol{\rho}'$.

Finally, the state of the manipulated point can also be constrained at some specific time frames by selecting them directly on the motion curve. Similar to pins, at those times t_l we have $\mathbf{s}'(t_l) = \mathbf{s}(\boldsymbol{\theta}, t_l)$. We also append those constrained and desired states to $\boldsymbol{\rho}(\boldsymbol{\theta})$ and $\boldsymbol{\rho}'$.

All the aforementioned pins, contacts, and state constraints then define hard constraints in our optimization problem. Noting that $\boldsymbol{\rho}' = \boldsymbol{\rho}(\boldsymbol{\theta}')$ for some $\boldsymbol{\theta}'$, we can use the linear approximation given in Equation 1 to get

$$\Delta\boldsymbol{\rho}' - \mathbf{J}_{\boldsymbol{\rho}}(\boldsymbol{\theta})\Delta\boldsymbol{\theta} = \mathbf{0}, \tag{6}$$

where $\Delta\boldsymbol{\rho}' = \boldsymbol{\rho}' - \boldsymbol{\rho}(\boldsymbol{\theta})$, and $\Delta\boldsymbol{\theta} = \boldsymbol{\theta}' - \boldsymbol{\theta}$, as before.

Variables limits. Let us define $\boldsymbol{\theta}_{i,k}$ as the tangent for parameter c_i at keyframe t_k. If the tangent is broken, we respectively call $\boldsymbol{\theta}_{i,k^-}$ and $\boldsymbol{\theta}_{i,k^+}$ the tangents corresponding to the Bezier segments before and after t_k. Each of these tangents is composed of two components $\boldsymbol{\theta}_{i,k} = \begin{pmatrix}\theta^X_{i,k}\\ \theta^Y_{i,k}\end{pmatrix}$. The X component (horizontal expansion of tangent in Fig. 6) determines the timing of the interpolation. In order to ensure that the spline interpolation is injective, i.e. at each time t there is only one value of $c_i(t)$, we limit this component to:

$$\begin{aligned} 0 \le \theta^X_{i,k^+} &\le (t_{k+1} - t_k) \\ 0 \le \theta^X_{i,k^-} &\le (t_k - t_{k-1}) . \end{aligned} \tag{7}$$

The Y component (vertical expansion) determines the range of values that $c_i(t)$ takes. In order to limit it within the range u_i to v_i, we have to restrict the Y component of the tangents. This is important especially when $c_i(t)$ represents joint angles (see angle limits as defined in section 3.1). We can then impose the following limits on the tangents:

$$\begin{aligned} \phi(u_i, k^+) \le \theta^Y_{i,k^+} &\le \psi(v_i, k^+) \\ -\psi(v_i, k^-) \le \theta^Y_{i,k^-} &\le -\phi(u_i, k^-). \end{aligned} \tag{8}$$

Depending on the type of interpolation, the functions ϕ and ψ can have complex expressions, or even have no closed form. It is the case for Bezier interpolations, for which we propose the following approximation:

$$\begin{aligned} \phi(u_i, k^\pm) &= \frac{4}{3}\Big(u_i - min\big(c_i(t_k), c_i(t_{k\pm1})\big)\Big) \\ \psi(v_i, k^\pm) &= \frac{4}{3}\Big(v_i - max\big(c_i(t_k), c_i(t_{k\pm1})\big)\Big). \end{aligned} \tag{9}$$

We demonstrate in Appendix A that this approximation satisfies the limits u_i and v_i for $c_i(t)$ at all times.

Quadratic problem formulation. Given the energies, constraints and limits defined above, we finally obtain the following minimization problem:

$$\begin{aligned} \min_{\Delta\boldsymbol{\theta}} \quad & w_m \cdot E_m + w_d \cdot E_d + w_b \cdot E_b \\ \textit{subject to} \quad & \Delta\boldsymbol{\rho}' - \mathbf{J}_{\boldsymbol{\rho}}(\boldsymbol{\theta})\Delta\boldsymbol{\theta} = \mathbf{0} \\ & \left.\begin{aligned} &0 \le \theta^X_{i,k^+} \le (t_{k+1} - t_k) \\ &0 \le \theta^X_{i,k^-} \le (t_k - t_{k-1}) \\ &\phi(u_i, k^+) \le \theta^Y_{i,k^+} \le \psi(v_i, k^+) \\ &-\psi(v_i, k^-) \le \theta^Y_{i,k^-} \le -\phi(u_i, k^-) \end{aligned}\right\} \quad , \forall i, \forall k. \end{aligned} \tag{10}$$

This is a quadratic programming problem that we solve using the Mosek solver [Mosek 2010] with default parameters. In our implementation, we chose $w_m = 100.0$, $w_d = 1.0$ and $w_b = 1.0$ — here, the high value of w_m reflects the prevailing importance given to satisfying the user manipulations.

3.3 Implementation Details

Tangents reduction. We can drastically reduce the size of the optimization by realizing that it is not required to optimize for the entire set of tangents $\boldsymbol{\theta}$. Indeed, many tangent modifications will not affect the state of the manipulated point or of the constrained ones (pins and contacts), so they do not need to be considered. We denote C_s the set of all parameters that affect the state $\mathbf{s}$ of the manipulated point, and C_{s_j} the equivalent set for the state $\mathbf{s}_j$ of each constrained point. With hierarchical structures such as skeletons, those sets correspond to the parameters of the point's parent chain. We further denote the set of parameters whose tangents are actually optimized in Equation 10 with C.

We begin by setting $C = C_s$. Then, for each constrained point, we identify three cases: (1) if $C \cap C_{s_j} = \emptyset$, we ignore the constraint and remove it from the vector $\boldsymbol{\rho}$ because no modification on s will affect s_j; (2) if $C_{s_j} \subset C_s$, we remove the constraint from $\boldsymbol{\rho}$ and reduce the set $C := C \setminus C_{s_j}$; (3) otherwise, we keep the constraint and augment the set $C := C \cup C_{s_j}$. Finally, if a point in the time range $t \in [t_k, t_{k+1}]$ is manipulated, only the tangents corresponding to that time range need to be modified. In conclusion, the tangents that we optimize for in Equation 10 are:

$$\left(\boldsymbol{\theta}_{i,k^+}, \boldsymbol{\theta}_{i,k+1^-}\right), \ \forall i \in C.$$

For the example in Fig. 5 where the left arm is manipulated while the hip and hand are pinned, our solver optimizes for 60 tangents, while the total number of tangents present in this scene, supposing that the entire animation contains only 3 keyframes, is 556. This optimization allows to achieve a real-time interaction, as presented in Section 4.5.

Time sampling. Pinning or setting contact points $\mathbf{s}_j$ means imposing a constraint over a whole time period $[t_k, t_{k+1}]$. To do so, we sample the time. The set of samples is at most at every frame of the animation between t_k and t_{k+1}, but we can reduce its size in case $t_{k+1} - t_k$ is large or C_{s_j} is small. Indeed, there is a limited number of DoF in the spline animation curves that define the state $\mathbf{s}_j$, therefore we only need to sample the time range that number of times, which is: $4 \times size(C_{s_j})$.

Optimization and overconstrained cases. Most of the time, since the manipulations are incremental, one single iteration of the optimization is sufficient to satisfy the constraints. However sometimes it might not be enough, in which case we update the state and Jacobian values and run the optimization again — this happens if $\mathbf{s}'(t)$ is far from $\mathbf{s}(\boldsymbol{\theta}, t)$ and the linear approximation of Equation 1 is not accurate anymore. Our stopping criteria is that the manipulated point's state is close enough to the user given state, i.e. $\|\boldsymbol{s}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}, t) - \boldsymbol{s}'(t)\| < \kappa$.

If after several iterations (we used 5) the solution is not improved — i.e. the curve does not move closer to the user-specified point — we reject it and stay at the current configuration. This can happen when the problem is overconstrained, i.e. the set of constraints (pins, contacts and manipulated point) are unreachable given the set of DoF (tangents); an example of such a case is presented Fig. 7. When the user comes back to a reachable solution, the state is updated, which provides direct feedback about the feasibility of the manipulations.

Solver stability. In order to improve the stability of the quadratic programming solver, we allowed a threshold on the hard constraint corresponding to pins and contacts in Equation 6. It becomes:

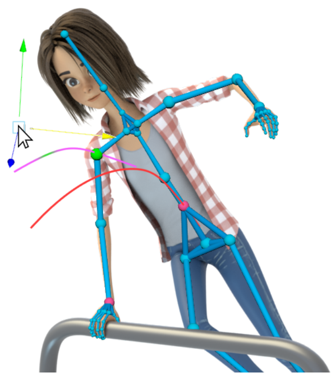


Fig. 7. Example of an infeasible manipulation. Here, the stomach is pinned and the wrist is set as contact, which does not leave enough DoF to move the shoulder as specified. The system thus stays at the previous state.

be obtained if $\boldsymbol{Q}$ is not full rank. We avoid such cases by regularizing $\boldsymbol{Q}$ with $\boldsymbol{Q} := \boldsymbol{Q} + \lambda \boldsymbol{I}$, with $\lambda = 10^{-6}$.

3.4 Timing Manipulations

Another important aspect to consider when manipulating interpolations is the timing. The extent of tangents in the X direction, $\theta^X_{i,k}$, influences the ease-in and ease-out of interpolations. We propose to let the user edit this easing directly on the motion curve, by scaling the timing up or down at any point (see Fig. 4, right). Scaling up means moving the frames apart around that point, i.e. making the motion faster, and scaling down means bringing the frames closer around that point, i.e. making the motion slower. If the user is scaling at time $t \in [t_k, t_{k+1}]$ by a factor σ, we modify the tangents as follows:

$$\begin{aligned} \theta^X_{i,k^+} &:= \theta^X_{i,k^+} + \sigma \frac{t - t_k}{t_{k+1} - t_k} \\ \theta^X_{i,k+1^-} &:= \theta^X_{i,k+1^-} + \sigma \frac{t_{k+1} - t}{t_{k+1} - t_k}, \end{aligned} \tag{12}$$

while still limiting the values to stay between 0 and $(t_{k+1} - t_k)$ in order to keep an injective function.

3.5 Static Case

It is interesting to notice that the system we introduce for the control of interpolations can be reduced to a posing system, similar to what we find in the literature [Shi et al. 2007; Yamane and Nakamura 2003], if we remove time from the equations. Indeed, $\mathbf{c}(\boldsymbol{\theta}, t)$ simply becomes $\mathbf{c}$, meaning that we directly optimize for attributes' values instead of tangents. The minimization problem of Equation 10 then becomes:

$$\begin{aligned} \min_{\Delta\mathbf{c}} \ & w_m \cdot \|\Delta\boldsymbol{s} - J_s(\mathbf{c})\Delta\mathbf{c}\|^2 + w_d \cdot \|\mathbf{D}\Delta\mathbf{c}\|^2 \\ \textit{subject to} \ & \left\|\mathbf{J}_{\boldsymbol{\rho}}(\boldsymbol{\theta})\Delta\boldsymbol{\theta}\right\|^2 = 0 \\ & u_i \le c_i \le v_i, \quad \forall i, \end{aligned} \tag{13}$$

where $\boldsymbol{\rho}$ is the vector of pinned (i.e. fixed) points and the Jacobian $J_s(\mathbf{c})$ is given by:

$$J_s(\mathbf{c}) = \frac{\delta \mathbf{s}(\mathbf{c})}{\delta \mathbf{c}}. \tag{14}$$

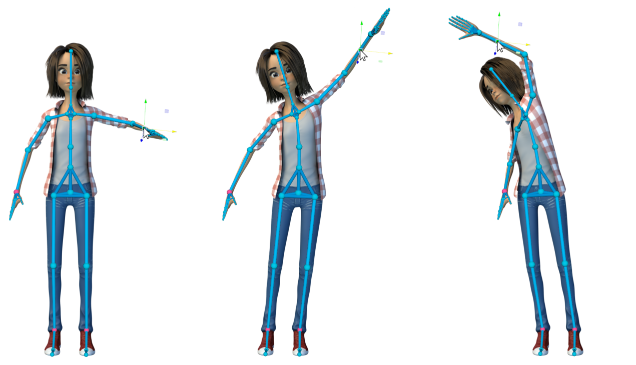


Fig. 8. The user is manipulating the character's left arm for a static pose. The feet and right hand are pinned, ensuring that they do not move during manipulations.

The obtained system provides similar interaction abilities as our interpolation control, always with angle limits and stiffnesses (Fig. 8). This shows that our problem formulation can be generalized to cover a wide range of the animation pipeline, and that it completely removes the need for fixed IK chains in the rig. It is the system we use in Section 4 to design the keyposes of our results.

4 EVALUATION

4.1 Examples of Authoring Difficult Animations

We implemented our system as an Autodesk Maya [Autodesk 2018] plugin. To demonstrate its potential, we let two artists design several animations for different types of characters. While our method is designed to work in harmony with traditional tools — such as rigs and graph editors — these animations were entirely authored using our system, from the design of key poses to the editing of interpolations. Please refer to the accompanying video to see the final animations, as well as some steps of the creation process and a comparison between the default and edited interpolations.

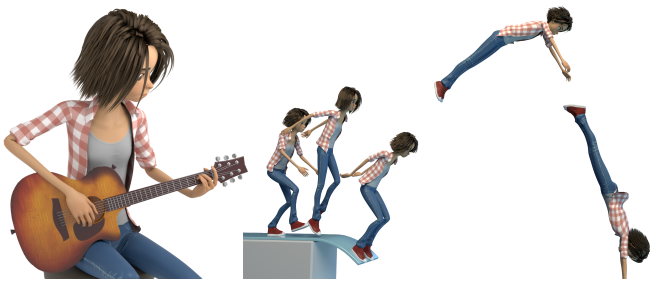


Fig. 9. Results on a human character. Left: Playing guitar animation, which necessitates specific contacts on fingers. Right: Diving animation, requiring foot contacts and a believable falling trajectory.

Complex interactions of hands with objects, such as playing the guitar, are some of the most difficult animations to create due to fingers' particular gestures and contacts. Animating this case typically involves IK chains on the fingers, which drastically slows down the rig evaluation, further hindering the creation process. In contrast, creation becomes very natural using our system, as one can pin some fingers, and freely move any other part of the character. Contacts can be specified for certain time ranges to ensure that the fingers won't move during interpolation while a note is being played. The animation presented in Fig. 9-left and in the accompanying video was created using our system without requiring any IK chain.

In a similar vein, the diving animation shown in Fig. 1 and Fig. 9-right necessitates contacts between the feet and the ground, which are easily handled with our system. Additionally, the interpolation for the jump requires a particular trajectory, with careful orientation and timing control, in order to achieve a realistic falling movement. Our system allows for a natural interaction with direct trajectory control for this case.

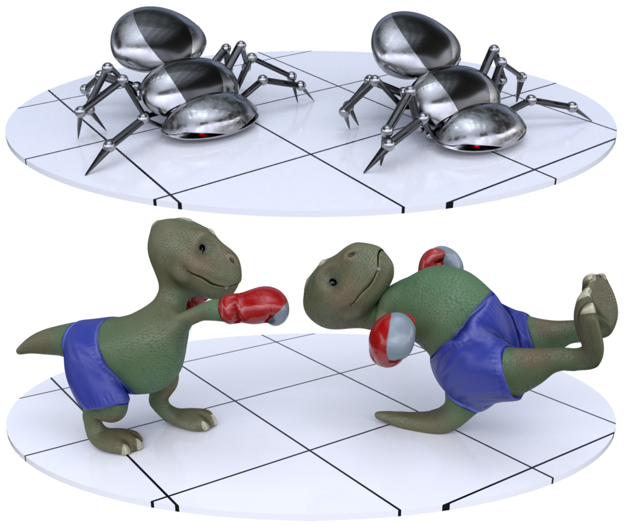


Fig. 10. Results on non-human characters. Top: Walking animation of a spider robot, generated using only the shown two key poses. Bottom: Punching animation of a dinosaur, who switches his contact component with the ground from the feet to the tail.

Our system is not limited to human models. In Fig. 10, we show results with a spider robot and a dinosaur. The spider animation is a walk cycle that was created using only two key poses, demonstrating the effectiveness of the ability to control interpolations with tangents. The dinosaur punching animation is another example where the control of timing plays an important role. Furthermore, near the end of the animation, the dinosaur is standing on a part of his tail. Handling this contact with traditional tools would be a challenging task, especially since a classic rig would most likely not contain any IK chain with an end effector at the middle of the tail. These issues are effortlessly solved using our system.

4.2 User study: Simpler Curves for Complex Motions

One objective of our system is to help animators work with clean animation curves with the least amount of keyframes, making further editing as easy and fast as possible. As the animations are typically refined and altered many times in production, this is crucial for an effective workflow. To evaluate this aspect on complex animations, we asked the same two professional artists, who have a long experience using Autodesk Maya, to animate their scenes from Fig. 9 and Fig. 10 using traditional tools. For each animation, we counted the number of keyframes used, which are listed in Table 1.

Table 1. For each animation, we compare the number of used keyframes (for the most keyframed control) when animating with a traditional system, and with ours. We also compare the speed of rig evaluation in fps.

Animation	Num keyframes		Framerate	
	Traditional	Ours	Traditional	Ours
Human Guitar	37	13	20.0	32.8
Human Dive	24	10		
Spider Walk	4	2	44.9	82.1
Dino Punch	19	12	36.9	47.7

The number of keyframes naturally depends on the animation style, but for all cases we can clearly see that our system allows to animate using fewer keyframes (on average, 2.2 times less). Moreover, since with traditional tools artists require IK chains, the rig evaluation is considerably slowed down, further hindering interaction. In Table 1, we show that since our system does not require IK chains, the speed of the playback is on average 60% faster.

4.3 User study: Faster Editing Process

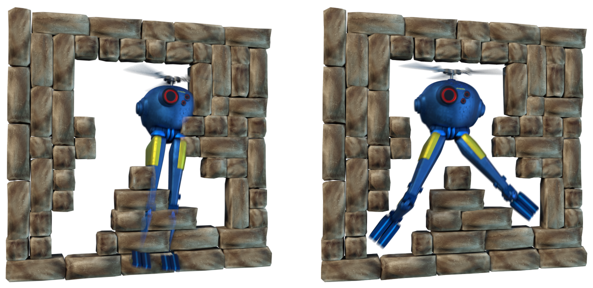


Fig. 11. 16 users were asked to edit a robot flying animation (left) in order to avoid obstacles (right), first using traditional tools on Maya and then using our plugin.

To evaluate the effectiveness and accessibility of our system, we conducted a further user study, similar to the ones in previous works by Guay et al. [2015] and Ciccone et al. [2017]. We invited twenty people unexperienced with 3D animation who were presented with a simple robot flying animation composed of only 4 keyframes, where the character is hitting obstacles during interpolations. The objective of the exercise was to modify the animation in order to avoid those obstacles (see Fig. 11). Using Maya, they were free to use FK or IK, add keyframes, or edit the curves of interpolations in the graph editor. Using our tool, they only edited the motion curves in the viewport.

Each user received a ten-minute introduction to each system. For counterbalancing, half of the participants had to start using Maya while the other half started with our tool. We measured the time, number of clicks, and keyframes to produce the final animations. Furthermore, each user was asked to self-evaluate the quality of his/her results on a scale from 0 to 5. Table 2 presents the mean and standard deviation of the obtained results. Examples of resulting animations are shown in the accompanying video.

Table 2. For each participant who edited the flying robot animation using Maya and our system, we measured the required time, number of clicks, number of keyframes, and self-evaluation score. This table shows the mean and standard deviation values.

	Time		Clicks		Keyframes		Score	
	Mean	St. dev.	Mean	St. dev.	Mean	St. dev.	Mean	St. dev.
Maya	10m13s	5m19s	209.6	83.7	11.4	2.8	2.4	0.8
Ours	7m06s	1m58s	140.7	39.9	4.0	0.0	4.1	0.6

We observe that participants were able to edit the animation more effectively with our system than with traditional tools. This demonstrates the accessibility of our system. On average, both the time and number of clicks were reduced by more than 30% using our system. We also notice that this efficiency does not come at the cost of poorer animations. Indeed, our approach received significantly higher self-evaluation scores in all cases. This is partially due to the reduced number of keyframes, which yields a smoother motion. Note that only one user attempted to edit the curves of interpolation in Maya's graph editor, a choice he eventually regretted. We believe that with additional experience with our tool, these gains will be further increased.

4.4 Qualitative Assessment from Professional Animators

We further conducted a survey and gathered observations on our system from three professional animators, along the lines of Koyama and Goto's evaluation [2018]. Each of them had the chance to try the tool before answering our questions. First, we asked what are the main shortcomings of the traditional animation process according to them. Then, we asked how our system responds to that, what are its benefits and drawbacks, and how they would further improve the tool. Finally, we asked if they would imagine such a system being integrated into their production tools. The obtained comments are summarized below and sorted by category.

Shortcomings of traditional animation. According to all three artists, the predominant difficulty with animation comes from the number of parameters. Each production character possesses hundreds of rig controls and, due to the unpredictability of interpolations, most of those controls are keyed every few frames. That represents a lot of values to specify through a very granular interaction. It makes the motion tedious to define, and even more cumbersome to refine later.

Benefits of our method. Artists were impressed by the flexibility that our system provides thanks to the pin-and-drag approach, yielding a very dynamic interaction. This approach allows not only reducing the number of keyframes, but also eliminating several controls and simplifying the architecture of the rig. As a result, animations require fewer parameters overall, which can be edited in a coordinated fashion through the simple manipulation of space-time curves. One animator even pointed out that being able to apply the tool directly on the skeleton could save a lot of rigging time and would be especially useful for secondary characters. Finally, all three animators appreciated the intuitiveness of the tool, which they found easy to learn and easy to use.

Drawbacks of our method. The main reservation of the interviewees was about working with global keyposes — i.e. assuming that all elements are keyed at the same frames — as they are used to working with different keyframes for different parts of a character. Also, it was sometimes frustrating for them to encounter infeasible states; they admitted that automatically adding keyframes to increase the DoF would be undesirable, but they could not agree on a preferable solution.

Integrability into production. All three animators expressed a clear desire to see such a system available in their workflow. One of them mentioned that the current state of the tool would already be particularly useful for quickly crafting motions in the early phase of production, and for animating secondary characters that require less granularity.

Other comments. One of the interviewees concluded with the following words: "Similar to the way ZBrush revolutionized modeling, animation will soon need its own revolution. And I could totally see a system like yours being part of it."

Speed gain after a longer experience. One of the animators was particularly involved in the development of this project, and we could see that he was continuously getting faster in using it. To evaluate that gain, we asked him to perform the same task as the user study of Section 4.3. As a result, he was almost 3 times faster using our system than with the Maya tools he is trained on.

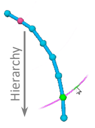
4.5 System Performance

We tested the performance of our approach on a machine with an Intel Core i7-4930K CPU and 32GB of RAM (we do not specify the graphics card since we do not use any GPU optimization, neither does the optimization library we use in our system [Mosek 2010]). Table 3 shows the computing times required to obtain the tangent values from user input, depending on the number of DoF (i.e. tangents), the length of the interpolation (i.e. spacing between two keyframes), and constraint configuration (number of pins and contacts, lower or upper in the hierarchy of character controls). Note that our system does not depend on the total length of the animation since we only consider a section of it during our optimization (as discussed in Section 3.3).

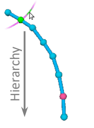
Given that artists rarely space their key poses by more than a few frames, and that most configurations do not exceed 100 degrees of freedom, we observe high enough frame rates for fluid interactions for typical animations in all cases. Some latency can be observed when a very large number of tangents are involved over a long interpolation. Even for such rare cases, we can maintain an interaction speed feasible for editing. Currently, creating the matrices of the QP problem — i.e. computing the Jacobians — represents on average 20% of the computing time in our tests, and this number goes up to 38% when a large number of DoFs and constraints are involved. Therefore, using a GPU optimization to parallelize the computation of all derivatives (from Equation 2) can significantly speed up the overall computation.

Table 3. Each table presents the time, in milliseconds, required for optimizing tangents to satisfy user manipulations, under different constraint configurations, as illustrated on their right. The values depend on the number of dof (i.e. 2× number of tangents) and the length of the interpolation. Each exposed value is the median over more than a hundred tests.

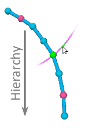
	Degrees of freedom			
Frames	24	72	120	168
2	10	12	16	21
7	10	13	16	21
20	10	12	16	20



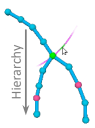
	Degrees of freedom			
Frames	24	72	120	168
2	52	54	57	63
7	59	68	81	101
20	63	103	144	200



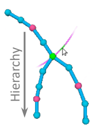
	Degrees of freedom			
Frames	24	72	120	168
2	52	57	69	86
7	58	80	94	130
20	67	106	158	227



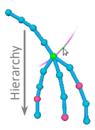
	Degrees of freedom			
Frames	24	72	120	168
2	55	57	59	64
7	60	67	77	92
20	60	99	137	167



	Degrees of freedom			
Frames	24	72	120	168
2	53	59	63	68
7	63	80	91	113
20	65	103	144	187



	Degrees of freedom			
Frames	24	72	120	168
2	56	59	61	66
7	61	76	89	99
20	63	101	162	191



5 LIMITATIONS AND FUTURE WORK

It is possible that there is no solution for tangents in Equation 10 that would satisfy the user manipulations, especially if there are numerous contacts and pins specified. This means that there is no configuration of Bezier interpolations on joints' rotations that result in a certain trajectory for the end effector, while satisfying the constraints. For these cases, what we currently have is similar to that of IK systems: The optimization gives the closest possible solution. A possible alternative is enforcing the creation of additional keyframes. We leave such mixed optimization of parameters and keyframes as future work.

In this work, we assumed that animators use global keyposes. For certain animations, it might be more convenient to use different keyframes for different parts of a character. Computationally, this can be achieved by using different keyframes for different attributes when optimizing tangents in Equation 10 (i.e. attribute-dependent t_k values). However, the interaction might suffer from the inability to visualize clear segments of motion, and therefore yield unexpected behavior when editing. We plan to develop alternative visualization strategies for these cases.

We have worked with connected structures such as skeletons and body rig controls. But some structures, such as blendshape deformers on a facial rig, consist of a set of independent handles. Our system naturally extends to such cases, with a simplified optimization as only the tangents of that handle's attributes would need to be optimized. Visualization of motion curves would again be the main challenge for these cases.

6 CONCLUSION

We proposed a new optimization-based keyframed animation system. Formulating common constraints and user interactions as an optimization problem in the tangent space of animation curves allowed us to handle the problem with fast quadratic programming based solvers. The result is an efficient real-time system that abstracts away the difficult choice of FK/IK from the user, without adding keyframes or complicating animation curves. These properties make the proposed system practical and easy to incorporate into existing animation processes. We believe that with its stable and fast implementation, our system is an important addition to the current animation tools.

ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 642841. We would like to thank all the participants of our user studies for their time and fruitful discussions, and in particular professional artists for their help and useful inputs during the whole project.

REFERENCES

Andreas Aristidou, Joan Lasenby, Yiorgos Chrysanthou, and Ariel Shamir. 2018. Inverse Kinematics Techniques in Computer Graphics: A Survey. *Computer Graphics Forum* 37, 6 (2018), 35–58.

Autodesk. 2018. Maya.

Jinxiang Chai and Jessica K. Hodgins. 2007. Constraint-based Motion Optimization Using a Statistical Dynamic Model. *ACM Trans. Graph.* 26, 3 (2007).

Byungkuk Choi, Roger B. i Ribera, J. P. Lewis, Yeongho Seol, Seokpyo Hong, Haegwang Eom, Sunjin Jung, and Junyong Noh. 2016. SketchiMo: Sketch-based Motion Editing for Articulated Characters. *ACM Trans. Graph.* 35, 4 (2016), 146:1–146:12.

Myung G. Choi and Kang H. Lee. 2016. Points-based user interface for character posing. *Computer Animation and Virtual Worlds* 27, 3-4 (2016), 213–220.

Loïc Ciccone, Martin Guay, Maurizio Nitti, and Robert W. Sumner. 2017. Authoring Motion Cycles. In *Proceedings of the 16th ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 8:1–8:9.

Michael F. Cohen. 1992. Interactive Spacetime Control for Animation. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques.* 293–302.

Oliver Glauser, Wan-Chun Ma, Daniele Panozzo, Alec Jacobson, Otmar Hilliges, and Olga Sorkine-Hornung. 2016. Rig Animation with a Tangible and Modular Input Device. *ACM Trans. Graph.* 35, 4 (2016), 144:1–144:11.

Michael Gleicher. 1997. Motion Editing with Spacetime Constraints. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics.* 139–ff.

Michael Gleicher and Andrew P. Witkin. 1991. Differential Manipulation. In *Proceedings of Graphics Interface.*

Martin Guay, Marie-Paule Cani, and Rémi Ronfard. 2013. The Line of Action: An Intuitive Interface for Expressive Character Posing. *ACM Trans. Graph.* 32, 6 (2013), 205:1–205:8.

Martin Guay, Rémi Ronfard, Michael Gleicher, and Marie-Paule Cani. 2015. Space-time Sketching of Character Animation. *ACM Trans. Graph.* 34, 4 (2015), 118:1–118:10.

Fabian Hahn, Frederik Mutzel, Stelian Coros, Bernhard Thomaszewski, Maurizio Nitti, Markus Gross, and Robert W. Sumner. 2015. Sketch Abstractions for Character Posing. In *Proc. of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 185–191.

Félix G. Harvey and Christopher Pal. 2018. Recurrent Transition Networks for Character Locomotion. In *SIGGRAPH Asia 2018 Technical Briefs.* 4:1–4:4.

Ollie Johnston and Frank Thomas. 1981. *The illusion of life: Disney animation.* Disney Editions New York.

Yuki Koyama and Masataka Goto. 2018. OptiMo: Optimization-Guided Motion Editing for Keyframe Character Animation. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.* 161:1–161:12.

Mikko Kytö, Krupakar Dhinakaran, Aki Martikainen, and Perttu Hämäläinen. 2017. Improving 3D Character Posing with a Gestural Interface. *IEEE Computer Graphics and Applications* 37, 1 (2017), 70–78.

Andreas M. Lehrmann, Peter V. Gehler, and Sebastian Nowozin. 2014. Efficient Nonlinear Markov Models for Human Motion. In *2014 IEEE Conference on Computer Vision and Pattern Recognition.* 1314–1321.

Jianyuan Min, Yen-Lin Chen, and Jinxiang Chai. 2009. Interactive Generation of Human Animation with Deformable Motion Models. *ACM Trans. Graph.* 29, 1 (2009), 9:1–9:12.

APS Mosek. 2010. The MOSEK optimization software. *Online at http://www. mosek. com* 54, 2-1 (2010).

Jean-Christophe Nebel. 1999. Keyframe interpolation with self-collision avoidance. In *Computer Animation and Simulation '99.* 77–86.

A. Cengiz Öztireli, Ilya Baran, Tiberiu Popa, Boris Dalstein, Robert W. Sumner, and Markus Gross. 2013. Differential Blending for Expressive Sketch-based Posing. In *Proc. of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 155–164.

Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. 1996. Efficient Generation of Motion Transitions Using Spacetime Constraints. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques.* 147–154.

Xiaohan Shi, Kun Zhou, Yiying Tong, Mathieu Desbrun, Hujun Bao, and Baining Guo. 2007. Mesh Puppetry: Cascading Optimization of Mesh Deformation with Inverse Kinematics. *ACM Trans. Graph.* 26, 3 (2007), 81–89.

Jack M. Wang, David J. Fleet, and Aaron Hertzmann. 2008. Gaussian Process Dynamical Models for Human Motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 2 (2008), 283–298.

Xiaolin Wei and Jinxiang Chai. 2011. Intuitive Interactive Human-Character Posing with Millions of Example Poses. *IEEE Computer Graphics and Applications* 31, 4 (2011), 78–88.

Harold Whitaker and John Halas. 2013. *Timing for animation.* CRC Press.

Andrew Witkin and Michael Kass. 1988. Spacetime Constraints. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques.* 159–168.

Katsu Yamane and Yoshihiko Nakamura. 2003. Natural Motion Animation through Constraining and Deconstraining at Will. *IEEE Transactions on Visualization and Computer Graphics* 9, 3 (2003), 352–360.

Wataru Yoshizaki, Yuta Sugiura, Albert C. Chiou, Sunao Hashimoto, Masahiko Inami, Takeo Igarashi, Yoshiaki Akazawa, Katsuaki Kawachi, Satoshi Kagami, and Masaaki Mochimaru. 2011. An Actuated Physical Puppet As an Input Device for Controlling a Digital Manikin. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* 637–646.

Xinyi Zhang and Michiel van de Panne. 2018. Data-driven Autocompletion for Keyframe Animation. In *MIG'18: Motion, Interaction and Games (MIG 2018).* 1–11.

A APPENDIX: ATTRIBUTES LIMITS

In this appendix, we demonstrate that if a parameter c is constrained to stay between u and v, the limits on the tangents we defined in Equations 8 and 9 satisfy the parameter's constraints at all times. Between two keyframes t_k and t_{k+1}, we consider that the value of $c(\boldsymbol{\theta}, t)$ is interpolated by a Bezier cubic spline:

$$\begin{bmatrix} t \\ c(\boldsymbol{\theta}, t) \end{bmatrix} = \begin{bmatrix} B_c^X(\lambda) \\ B_c^Y(\lambda) \end{bmatrix} = \begin{bmatrix} t_k \\ c(t_k) \end{bmatrix} (2\lambda^3 - 3\lambda^2 + 1) + 3\lambda(1-\lambda)^2 \boldsymbol{\theta}_{k^+} + \begin{bmatrix} t_{k+1} \\ c(t_{k+1}) \end{bmatrix} (3\lambda^2 - 2\lambda^3) - 3\lambda^2(1-\lambda)\boldsymbol{\theta}_{k+1^-},$$

where $\lambda \in [0,1]$. Therefore, imposing $c(\boldsymbol{\theta}, t) \leq v \; \forall t \in [t_k, t_{k+1}]$ is equivalent to imposing $B_c^Y(\lambda) \leq v \; \forall \lambda \in [0,1]$. By injecting the upper limits of Equation 8 into $B_c^Y(\lambda)$, with the definitions of ϕ and ψ proposed in Equation 9, we obtain the following inequality, where we note $mx = max(c(t_k), c(t_{k+1}))$ for easier reading:

$$B_c^Y(\lambda) \leq \begin{array}{l} c(t_k)(2\lambda^3 - 3\lambda^2 + 1) + 4(v - mx)\lambda(1-\lambda)^2 \\ \quad + c(t_{k+1})(3\lambda^2 - 2\lambda^3) + 4(v - mx)\lambda^2(1-\lambda). \end{array}$$

Knowing that both $(2\lambda^3 - 3\lambda^2 + 1)$ and $(3\lambda^2 - 2\lambda^3)$ are positive for $\lambda \in [0,1]$, and by definition both $c(t_k)$ and $c(t_{k+1})$ are lower than mx, we obtain:

$$B_{c,i}^Y(\lambda) \leq 4(v - mx)\lambda(1-\lambda) + mx.$$

Finally, since $\lambda(1-\lambda) \leq 0.25$ on $[0,1]$ and $v - mx$ is positive (we suppose that the limit is respected at keyframes), we end up with the desired result: $B_{c,i}^Y(\lambda) \leq v$.

The same demonstration is valid for the lower limit. Indeed, if u is the lower limit of c, that means that $-u$ is the upper limit of $-c$, which brings us back to the above computation.