Editing and Constraining Kinematic Approximations of Dynamic Motion

Cyrus Rahgoshay

Master of Science

Computer Science

McGill University

Montréal,Quebec

December 2011

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Science

ACKNOWLEDGEMENTS

Of the many people who have been enormously helpful in the preparation of this thesis, I am especially thankful to Prof. Paul G. Kry for his help and support in guiding me through to its successful completion. I would also like to express my gratitude to the members of the computer graphics lab at McGill University for their time and valuable feedback during the primary investigation of this project.

I would also like to warmly acknowledge Prof. Karan Singh (School of Computer Science, University of Toronto) for his guidance and input throughout the process of this research, as well as Amir H. Rabbani (lab partner, CG lab at McGill University) who went out of his way and invested time in providing me with necessary additional material for the implementation of the project.

In addition, a special thanks to all my friends for their consideration and motivation. I dedicate this dissertation to my family for their unconditional love and support in every way possible throughout the process of this course, this dissertation and beyond.

ABSTRACT

This thesis presents inverse kinodynamics (IKD), an animator friendly kinematic workflow that both encapsulates short-lived dynamics and allows precise space-time constraints. Kinodynamics (KD), defines the system state at any given time as the result of a kinematic state in the recent past, physically simulated over a short temporal window to the present. KD is a well suited kinematic approximation to animated characters and other dynamic systems with dominant kinematic motion and short-lived dynamics. Given a dynamic system, we first formulate an appropriate kinodynamic window size based on kinematically defined accelerations in the kinematic trajectory and physical properties of the system. We then present an inverse kinodynamics (IKD) algorithm, where a kinodynamic system can precisely attain a set of animator constraints at specified times. Our approach solves the IKD problem iteratively, and is able to handle full pose or end effector constraints at both position and velocity level, as well as multiple constraints in close temporal proximity. Our approach can also be used to solve position and velocity constraints on passive systems attached to kinematically driven bodies. We show IKD to be a compelling approach to the direct kinematic control of character, with secondary dynamics via examples of skeletal dynamics and facial animation.

ABRÉGÉ

Cette thèse présente kinodynamique inversé (IKD) qui est un procédé kinematique très pratique utilisable pour l'animateur qui consiste à la fois d'une dynamique de courte durée et qui permet des contraintes spatio-temporelles précises. Kinodynamique (KD) définit l'état du système à un moment donné comme le résultat d'un état kinematique dans un passé récent, physiquement simulé dans une fenêtre temporelle de courte durée du temps présent. KD est une approximation kinematique bien adaptée aux caractères animés et à d'autres systèmes dynamiques avec un mouvement kinematique dominant et une dynamique de courte durée. En ayant un système dynamique on peut d'abord formuler une taille de fenêtre kinodynamique appropriée, basée sur des accélérations définies kinematiquement dans la trajectoire kinematique et sur les propriétés physiques du système. Nous présentons ensuite un algorithme kinodynamique inversé (IKD) dans lequel un système kinodynamique peut satisfaire un ensemble de contraintes des animateurs à des moments précis. Notre approche résout le problème IKD de manière itérative et permet de gérer une pose complète ou des contraintes des points fixés sur le corps à la fois au niveau de la position et de la vitesse ainsi que de multiples contraintes dans une courte proximité temporelle. Notre approche peut également être utilisée pour résoudre des contraintes de position et de vitesse dans des systèmes passifs attachés à des corps kinematiquement entrainés. Nous démontrons qu'IKD peut être une approche convaincante pour le contrôle kinematique direct des caractères avec des dynamiques secondaires par des exemples de dynamiques du squelette et d'animation faciale.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
Introduction

Physical simulation is now a robust and common approach to recreating reality in virtual worlds and is almost universally used in the animation of natural phenomena, ballistic objects, and character accessories like clothing and hair. Despite these strides, the animation of primary characters continues to be dominated by the kinematic techniques of motion capture and above all traditional keyframing. Two aspects of a primary character in particular, skeletal and facial motion, are often laboriously animated using kinematics.

We note from conversations with about half a dozen master animators that there are perhaps three chief reasons for this. First, kinematics, unencumbered by physics, provides the finest level of control necessary for animators to breathe life and personality into their characters. Second, this control is direct and history-free, in that the authored state of the character, set at any point in time, is precisely observed upon playback and its impact on the animation is localized to a neighborhood around that time. Third, animator interaction with the time-line is WYSIWYG (what-you-see-is-what-you-get), allowing them to scrub to various points in time and instantly observe the character state without having to playback the entire animation.

The same animators expressed the utility and importance of secondary dynamics overlaid on primarily kinematic character motion to enhance the visceral feel of their characters. Various approaches to such secondary dynamics have been proposed

in research literature [7, 11], some of which are available in commercial animation software. Overlaid dynamics, unfortunately compromise the second and third reasons animators rely on pure kinematic control.

A kinematic solution incorporating secondary dynamics called *kinodynamic* skinning [3] was suggested in the context of volume preserving skin deformations. With this approach, a kinodynamic state at any time is defined as a kinematic state in the recent past, physically simulated forward to the given time. In this thesis we develop this idea of kinodynamics (KD) as a history-free kinematic technique that can incorporate short-lived dynamic behavior. Note that the above usage of the term "kinodynamic", while similar in spirit, is distinct from its use in the context of robot motion planning where it addresses planning problems where velocity and acceleration bounds must be satisfied [9].

We begin by formulating an appropriate KD window size for a given kinematic motion and physical parameters: both long enough to ensure a temporally coherent KD trajectory that captures the nuances of system dynamics, and short enough for interactive WYSIWYG computation and temporal localization of the influence of animation edits on system state. Many goal directed actions such as grasping, reaching, stepping, gesticulating, and even speaking, however, involve spatial relationships between the character and its environment, that are best specified directly, as targets states that the character (or parts of the character) must observe at given times. Techniques such as inverse kinematics (IK) and space time optimization algorithmically infer the remaining system states and animation parameters from these animator specified spatio-temporal targets. Though IK does not give the secondary

dynamics, and space time optimization is typically computationally expensive. Analogous to these techniques we develop an inverse kinodynamics (IKD) algorithm allowing animators to prescribe position and velocity constraints at specific points in time within a KD setting.

The contribution of this thesis is thus the development of a usable kinodynamic framework for interactive character animation with real-time performance, where animators can leverage a direct history-free kinematic workflow, coupled with the benefits of arbitrary physically simulated secondary dynamics. Toward this, we develop a formulation for KD window size and present the first IKD algorithm. While we impose no explicit restrictions on the physical simulation of characters, our approach is largely suited to well-conditioned and continuous simulations.

CHAPTER 2
Related work

Secondary dynamics provides a significant amount of visual realism in kinematically driven animations and is an important technique for animators. In the case of tissue deformations produced by the motion of an underlying skeleton, various methods can be used to produce this motion through simulation or using precomputation [7, 11]. With respect to secondary dynamics of skeletal motion, it has similarly been demonstrated that tension and relaxation of the skeletal animation can be altered through physically based simulation [17]. These techniques provide an important richness to an animation; while the style of the results are controllable by adjusting elastic parameters or gains of controllers used for tracking, precise control of the motion itself to satisfy given constraints or key frames is typically left to be treated as a separate problem.

The related work can be categorized into two groups. First, there are approaches which try to control a physically based simulation to have it meet some desired constraints. Second, there are approaches which use kinematic editing techniques to produce animations that meet desired constraints and exhibit physically plausibility.

2.1 Controlling physically based simulations

There has been a significant amount of work in this area on controlling rigid bodies [20, 19], fluids [24, 14], and cloth [27, 5]. Other recent successes on controlling physically based animation use gentle forces to guide an animation along a desired

trajectory, accurately achieving desired states, but also allowing physical responses to perturbations [4]. Physically based articulated character control has received a vast amount of interest. Building on the seminal work of locomotion control [21], it is now possible to have, for instance, animation of physically based motions that respond naturally to perturbations [29, 28, 30], and editable animations of dynamic manipulation which respects the dynamic interaction between characters and objects [1]. Allen et al. [2] change PD control parameters to produce skeletal animations that interpolate key-frames at specific times. In contrast, we keep the control parameters fixed and alter the kinematic trajectory. Jain and Liu [10] show a method for interactively editing interaction between physically based objects and a human. In this work, it is the motion of the dynamic environment which is edited through kinematic changes of a captured human motion. In comparison, we focus on altering and editing a kinodynamic motion with different styles (tension and relaxation) and different constraints. Directly related to the problem of authoring motion, physically correct motion can be achieved by solving optimizations with space-time constraints [26].

2.2 Kinematic editing techniques

In contrast to the work on controlling fully dynamic simulations, we are addressing a simplified problem due to the finite temporal window involved in simulating the state at a given time in a kinodynamic trajectory [3]. This leads to benefits in the context of animation authoring, and allows for a straightforward solution to the inverse kinodynamic problem we present in this thesis. In a different approach, with similar objectives to our own work, Kass and Anderson [12] propose a method for

including physically based secondary dynamics in a key frame style editing environment through interactive solutions of space time optimization problems. They focus on linear or linearized space-time constraints problems, while our work in contrast looks primarily at non-linear skeletal animation problems. Within a purely kinematic setting, Coleman et al. [8] create handles to edit motion extrema of different joints clustered in time. The visual impact of secondary dynamics is often captured in these temporal relationships. Such an approach can, however, only exaggerate or diminish a dynamic effect already present in the motion and cannot introduce new forces and dynamic behaviors that the mixing of kinematics and dynamics allows. Such mixing to get the best of both worlds also has promise for authoring motion in real time. For instance, Nguyen et al. [18] blend kinematic animation and dynamic animation via a set of forces which act like puppet strings to pull the character back to the kinematic trajectory. Also of note is the work on editing kinematic motion through momentum and force [22], or with biomechanically inspired constraints [13]. While these different approaches use dynamic principles to control accelerations and velocity, they deal with dynamic systems which are not necessarily short lived, and these approaches do not share our objective of a scrubbing interface for animation editing which computes states largely in a history free manner.

CHAPTER 3
Overview

In this chapter we provide an overview of our approach. The animation is principally driven by a kinematic trajectory $\boldsymbol{x}_K(t)$, typically authored and edited using traditional keyframe and motion capture techniques. The kinodynamic trajectory of the system $\boldsymbol{x}_{KD}(t)$ at a time t is the result of a physical simulation run over a time window δ starting from an initial position $\boldsymbol{x}_K(t-\delta)$ and velocity $\dot{\boldsymbol{x}}_K(t-\delta)$. The simulation uses a PD (Proportional-Derivative) controller to follow the kinematic trajectory, so the $\boldsymbol{x}_K(t)$ can be thought of as the target or desired trajectory. The PD controller output (f_{out}) is proportional to the error which is the difference between the set point (x_k) and the process variable (x). In our simulation x_k is the desired target or trajectory and x is the current state of the system. So the equation of the PD controller can be written as follows:

$$f_{out} = K_p * error = K_p * (x_k - x), \tag{3.1}$$

where the tension and relaxation gain is denoted by the parameter K_p.

We will have kinodynamic states which deviate from the kinematic trajectory because we are using a simulation with control forces to generate the KD trajectory. This is desirable because we want to include the effects of secondary dynamics in the animation (see Figure 3–1). However, there may be specific times in the animation where we need constraints to be met.

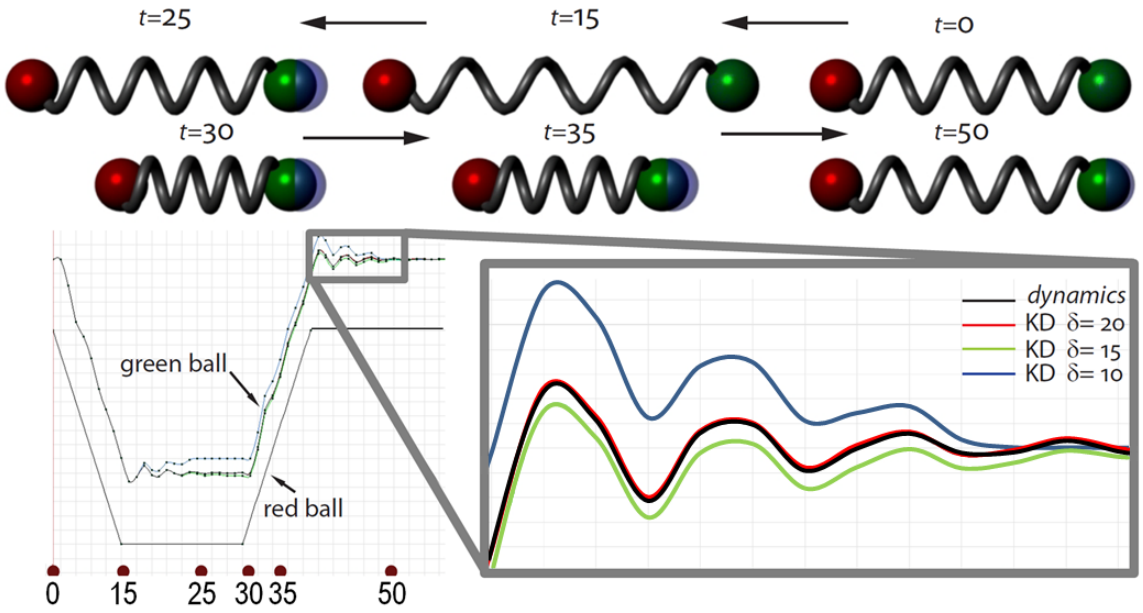


Figure 3–1: KD trajectories for the green ball: Kinematically the green ball is rigidly connected to the keyframed red ball, with spring dynamics overlaid. A number of frames of the KD trajectory ($\delta = 15$) are shown, with the full dynamics solution for the green ball overlaid in blue (top). KD trajectories with 3 window sizes are shown in relation to a full dynamics solution (bottom). Note how the history-free KD trajectories capture the visual behavior of the actual dynamics over a wide range of window sizes.

Suppose we have an target pose $\boldsymbol{x}_i$ that must be produced at time t_i. This target state could be a pose in the original kinematic trajectory, or something different. If the pose belongs to the original kinematic trajectory, a simple solution would be to stiffen the PD control in the vicinity of the desired pose so that it is tracked precisely. Note, however, that stiffness is an inherent attribute of the motion's secondary dynamics under animator control and altering it to interpolate a target pose imbues the animation with a different style. Instead, we iteratively compute a modification to the kinematic trajectory which results in a KD state that satisfies the constraint.

Example trajectories are illustrated in Figure 3–2, where a red kinodynamic trajectory follows a green kinematic trajectory, but is lower due to gravity pulling the system down. At left we can see an illustration of how the temporal window for computing kinodynamic state must be long enough for any impulse (smaller than a given maximum) to come sufficiently close to rest that it is not perceptual (e.g., based on screen pixels). At right in the figure we can see a dotted green kinematic trajectory with an added bell shape correction, which produces the dotted red kinodynamic trajectory satisfying the constraint at time t_i.

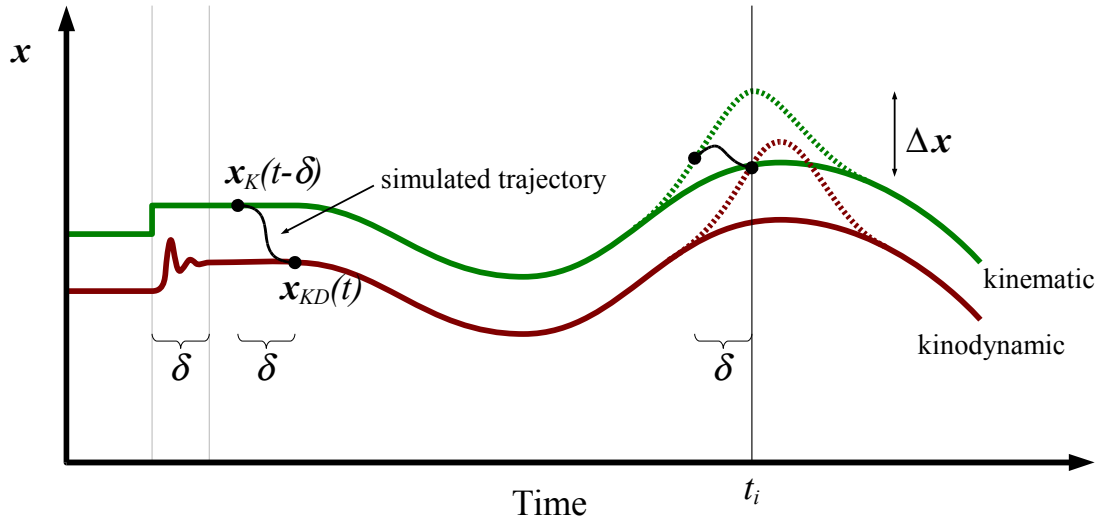


Figure 3–2: An illustration of how we modify a kinematic trajectory to create a kinodynamic trajectory that satisfies the constraint that the original kinematic state be produced at time t_i.

3.1 Inverse kinodynamics

Let SimulateKD($\boldsymbol{x}_K$, t_i, δ) (see Algorithm 1) be the procedure of computing $\boldsymbol{x}_{KDi}$, the KD state at t_i for kinematic trajectory $\boldsymbol{x}_K$. We first compute the IKD

error in meeting the target as

$$\boldsymbol{e}_i = \boldsymbol{x}_i - \boldsymbol{x}_{KDi}, \tag{3.2}$$

and from this we form bell shaped correction curves $\boldsymbol{e}_i\phi_i(t)$ (i.e., the state is a vector and each coordinate of the state will have a bell shaped correction of a different magnitude) that we will add to kinematic trajectory. The bell shaped basis function $\phi_i(t)$ provides a local correction, has its peak value of 1 at t_i, and can be defined as a low degree polynomial or Gaussian. More importantly, it has a local support (i.e., a small temporal width) which is selected by the artist.

Algorithm 1 SimulateKD

Input: $\boldsymbol{x}_K$, t_i, δ
Output: $\boldsymbol{x}_{KDi}$, the KD state at t_i for $\boldsymbol{x}_K$

1: $\boldsymbol{x}_{Kinit} = \boldsymbol{x}_{K(t_i-\delta)}$
2: $\dot{\boldsymbol{x}}_{Kinit} = \dot{\boldsymbol{x}}_{K(t_i-\delta)}$
3: $\boldsymbol{x}_{KDi} \leftarrow$ simulate and compute the state $\boldsymbol{x}_{KD}$ up to time t_i

Conceptually, this IKD error correction introduces an additional spring force proportional to $\boldsymbol{e}_i\phi_i(t)$ in a small temporal neighborhood around t_i. This correction will not be sufficient, however, and our modified KD state $\tilde{\boldsymbol{x}}_{KDi} = \text{SimulateKD}(\boldsymbol{x}_K + \boldsymbol{e}_i\phi_i, t_i, \delta)$ will not meet the constraint. This is because the correction did not take into account the dynamics of the system, but we can fix this by boosting the correction to account for the dynamics, assuming that the system dynamics are approximately locally linear (see Figure 3–3). Letting $\boldsymbol{d}_i = \tilde{\boldsymbol{x}}_{KDi} - \boldsymbol{x}_{KDi}$, we simply project the error onto this initial correction result to get an improved correction function

$$\boldsymbol{f}(t) = \Delta\boldsymbol{x}_i\phi_i(t), \tag{3.3}$$

where $\Delta \boldsymbol{x}_i = (\boldsymbol{e}_i \cdot \boldsymbol{d}_i / ||\boldsymbol{d}_i||)\, \boldsymbol{e}_i$.

Using $\boldsymbol{x}_K + \boldsymbol{f}$, the process is repeated, until the system state converges to within a numerical threshold of $\boldsymbol{x}_i$ at t_i. That is, we find the new kinodynamic state at t_i, compute the error $\boldsymbol{e}_i$, the modified kinodynamic state using $\boldsymbol{x}_K + \boldsymbol{f} + \boldsymbol{e}_i \phi_i$, the correction result $\boldsymbol{d}_i$, and finally an update to the correction function

$$\Delta \boldsymbol{x}_i \leftarrow \Delta \boldsymbol{x}_i + (\boldsymbol{e}_i \cdot \boldsymbol{d}_i / ||\boldsymbol{d}_i||)\, \boldsymbol{e}_i. \tag{3.4}$$

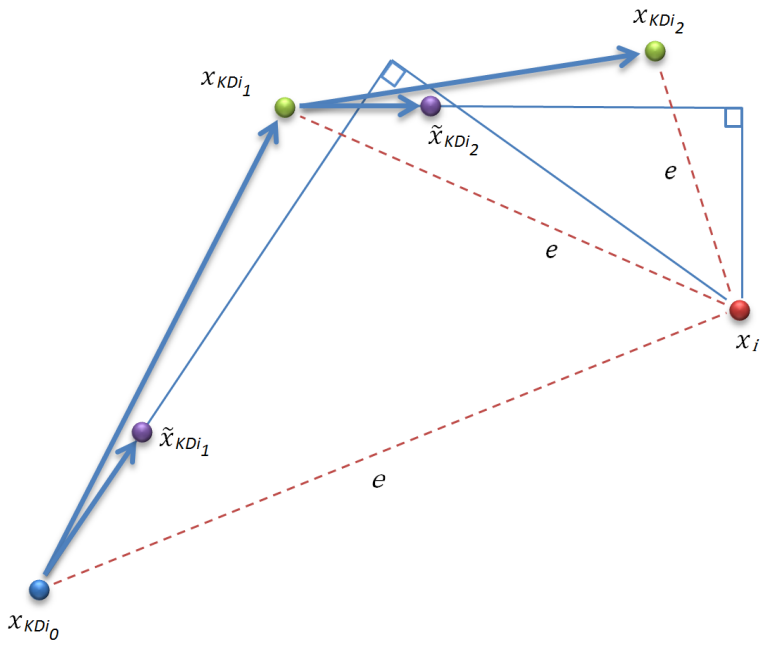


Figure 3–3: Illustration showing one step of the IKD iteration at time t_i. $\boldsymbol{x}_{KDi_0}$ shows the initial KD state of the system at time t_i trying to reach target $\boldsymbol{x}_i$. A correction function computes the modified KD state $\tilde{\boldsymbol{x}}_{KDi_1}$ which does not take into account the dynamics of the system. We can find the projection of the error onto the initial correction result (a prediction of the state we would get on adjusting the correction if the system was linear). Using the scaled correction (Equation 3.4), we produce the new KD state $\boldsymbol{x}_{KDi_1}$. The next iteration involves computing the new correction function resulting in the modified KD state $\tilde{\boldsymbol{x}}_{KDi_2}$ to reach target $\boldsymbol{x}_i$. The iteration continues until the error $\boldsymbol{e}$ falls below a numerical threshold.

3.2 Inverse kinematics

We use Inverse Kinematics (IK) to compute the joint displacements that will account for the error of a skeletal pose, i.e., the difference between the end effector of the articulated character and the desired target. IK involves solving a nonlinear equation for the joint angles that an articulated character must have in order to reach a given target.

We use Gauss-Newton iteration to solve the nonlinear equation $f(\theta) = f_{desired}$. Specifically, each step involves linearizing the system, so we compute a Jacobian which relates joint velocities to a velocity of the end effector given the end effector position of the character, meaning $\dot{p} = J\dot{\theta}$. The end effector can be the hand, feet or any other part of the character body. Now assume that we have a character pose θ with an end effector $p(\theta)$ that is not reaching p_i, the desired target constraint set at time i. The Jacobian can be used to compute the joint angles' changes to get the end effector to the target such that $p(\theta) = p_i$. For linear cases of $p(\theta)$ the Jacobian $J = \partial d_p / \partial d_\theta$ will give us the desired solution. note that J is typically not square, so we will typically need to use a pseudoinverse to find a least squares solution.

But in our skeletal examples the relationship between joint angles and the end effector is nonlinear so we need to find the final solution by using the iteration. Since the transpose of our Jacobian relates forces at the end effector to torques at the joints, as an alternate way of using pseudoinverse at each iteration we can use the transpose of the Jacobian. That can be used as a way of controlling joint angles, because applying a force at the end effector to reach the target would cause joint torques which is what we need to update joint angles.

In cases where the target is out of reach, the Jacobian can become singular and the better solution will be to use regularized pseudoinverse or truncated singular value decomposition. We use above techniques in our SolveIK algorithm (see Chapter 4.1) to compute the end effector error or state displacement $\boldsymbol{e}_i$ for our IKD algorithm (see Algorithm 2). For more information, refer to a survey by Buss et al. [6].

CHAPTER 4
KD animation and IKD scenarios

In this thesis, we look at a number of scenarios that can largely be described as either pose constraints (as described earlier and Chapter 4.2) or end effector constraints (Chapter 4.1). For instance, we may want a kinodynamic skeletal animation of a dance to produce some key poses (see Figure 4–3), or a kinodynamic skeletal animation of a punch that actually hits the desired target at a specific time. Alternatively, another scenario which is important to consider is the case where we drive a deformable mesh animation to follow a target mesh animation. In contrast to joint angles, in this case the state is the Cartesian position of verticies in the mesh. In Chapter 4.5, we show how this approach can be used for facial animation. Later in Chapter 4.6 we show how the techniques used for skeletal and facial animation can be used to control secondary dynamics of deformable objects attached to kinematically driven bodies.

We note that the blending of the correction can be done in a number of ways. If we only have one position constraint to satisfy in the entire animation, then it would be possible to naively apply a constant offset to the kinematic trajectory in order to meet the constraint at time t_i. Typically we will have several constraints at different times, so we only make a local edit to the desired trajectory (see Chapter 4.3). Any number of smoothly shaped curves with compact support will serve this purpose, as discussed in Chapter 7. The shape and width of the correction basis functions are

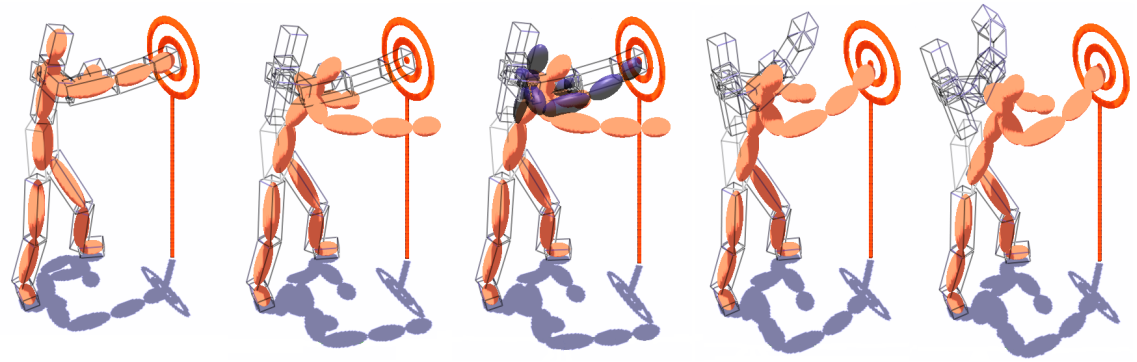
an important artist control, much like setting ease in ease out properties in a key frame animation.

4.1 Skeletal animation end effector IKD

In the case of an articulated character, the state $\boldsymbol{x}$ is a set of joint angles, and the simulation uses a PD (Proportional-Derivative) controller to follow the kinematic trajectory (we can think of it as the target or desired trajectory). The gains of the controller set the level of tension or relaxation of the character [17].

When editing a skeletal motion, we may wish to set constraints on the entire pose, as described earlier, but it is also important that we are able to constrain only part of the state, for instance an end effector at the time of a contact event. Suppose end effector position $p(\boldsymbol{x})$ of an articulated character must reach position p_i at time t_i. In this case we have the constraint $p(\boldsymbol{x}_{KDi}) = p_i$, and we use an inverse kinematics solution to map the end effector error to an error in the state.

Figure 4–1 shows an example of how we solve the IKD problem of punching a target. While the motion in Figure 4–1(a) hits the target at the desired time, we change the motion style by adjusting the tracking gains of the physically simulated character shown in orange, to produce the more relaxed KD motion show in Figure 4–1(b). This relaxed motion fails to hit the target, but we can solve an inverse kinematics problem to adjust the joints of our relaxed character so that the end effector does hit the target. This IK solution pose is shown in dark blue in Figure 4–1(c). We could make a purely kinematic fix to our KD trajectory by simply layering this IK solution on top our KD trajectory, using a bell shaped curve to slowly ease the correction in and out. However, this does not respect the relaxed dynamics of the



(a) (b) (c) (d) (e)

Figure 4–1: Illustration of how IKD is used to produce an animation of a relaxed character that punches a target. (a) shows the motion capture at the time of contact in both wire-frame and solid orange. (b) the solid orange character shows the KD state of the relaxed character, which fails to reach the target at the time of contact. (c) inverse kinematics produces the pose of the character in dark blue. (d) iteratively computing the error and modifying the kinematic trajectory produces a KD state which hits the target (orange). Here the modified motion capture pose is shown in wire-frame. (e) shows the result of using a smaller temporal width for the bell shaped correction curve, which results in more of an upper cut.

character as seen in the accompanying video[1] . Instead, we modify the kinematic trajectory used to produce the kinodynamic animation. By editing the kinematic trajectory, we ensure a natural looking motion that exhibits a relaxed style and produces a natural follow through motion. This modification is shown in Figure 4–1(d) and (e) for two bell shaped curves of different widths.

[1] http://www.cs.mcgill.ca/~crahgo/thesis/

The algorithm iterates as described in the previous chapter, using an update to the correction curve that is based on an inverse kinematics solution,

$$\boldsymbol{e}_i = \text{SolveIK}(\boldsymbol{x}_{KDi}, p_i) \tag{4.1}$$

where SolveIK computes a state displacement $\boldsymbol{e}_i$ such that $p(\boldsymbol{x}_{KDi} + \boldsymbol{e}_i) = p_i$. The IK solver has many possible options for satisfying the constraint, and we take the solution which has the minimum norm, but we can change the norm so that some joints are favoured over others, and we could choose that weighting based on the stiffness of the joint controllers. Note that the correction function update must be modified to use the end effector error, $\Delta p_i = p_i - p(\boldsymbol{x}_{KDi})$, instead of the state displacement $\boldsymbol{e}_i$. The update becomes

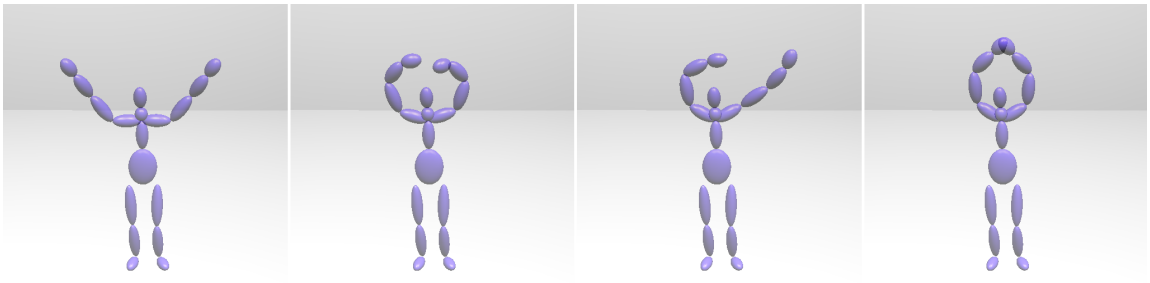
$$\Delta \boldsymbol{x}_i \leftarrow \Delta \boldsymbol{x}_i + (\Delta p_i \cdot d_i / ||d_i||)\, \boldsymbol{e}_i. \tag{4.2}$$

where $d_i = p(\tilde{\boldsymbol{x}}_{KDi}) - p(\boldsymbol{x}_{KDi})$.

4.2 Skeletal animation full pose IKD

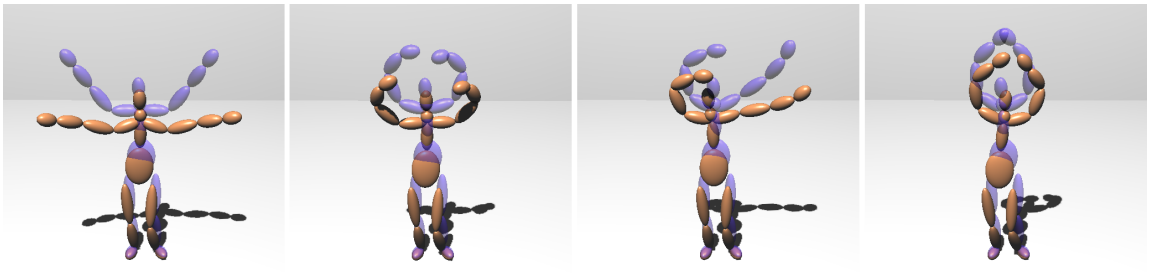
As mentioned earlier in the beginning of the chapter, we can set constraints on the state of the characters while editing the skeletal motion. So the constraint becomes the set of all desired joint angles of the character that must be achieved to satisfy the desired pose. The IKD can be used to compute a correction function to satisfy the constraints as mentioned in previous section. In our examples we tested the pose constraint on the skeletal animation of the YMCA dance.

We initially started by designing four YMCA key poses of the four letters of the dance (see Figure 4–2) and created an animation that blends between the four



(Y) (M) (C) (A)

Figure 4–2: The keyframed pose of the four letters of the YMCA dance.



(Y) (M) (C) (A)

Figure 4–3: YMCA dance Pose Constraints on skeletal animation. Blue character shows the desired pose and orange character shows the KD without the Pose Constraints.

poses. As seen in Figure 4–3, the KD animation follows the keyframe in a relaxed way and secondary dynamics in character motion is visible due to the nature of KD animation; however, the motion fails to achieve the correct pose. The IKD solution gives the correction functions to satisfy the YMCA poses at the specified keyframes and therefore a desired dance scenario.

4.3 Multiple constraints

The process of designing an animation typically involves setting multiple constraints at different times throughout the animation. If these events are sufficiently

far apart, we can treat each as an independent IKD problem. However, constraints in close temporal proximity may need to be solved simultaneously. This can happen in a variety of ways. If the bell shaped correction curve necessary to satisfy one constraint modifies kinematic states that fall within the temporal window used to simulate the KD state at a another constraint, then the solution of the latter constraint will depend on the solution of the former. This dependence can be one way, or both ways, depending on the temporal width of the bell shaped curves used for each constraint, and the temporal window size used for the kinodynamic simulation. While we may be able to solve some constraints independently, or in a specific order, for simplicity we will make the assumption in this section that all of our constraints are temporally coupled and must be solved simultaneously. In our multi-constraint examples, we typically choose correction function widths that provide an ease-in trajectory with a duration of approximately one or two seconds, so constraints that fall within one or two seconds of one another will need to be addressed simultaneously.

The correction $\boldsymbol{f}$ that we must add to the kinematic state to satisfy a number of constraints can now be seen as an interpolation function. That is, $\boldsymbol{f}$ interpolates a set of corrections $\Delta\boldsymbol{x}_i$ at t_i, for $i = 1..N$ where N is the number of constraints. The correction function interpolates using a sum of basis functions,

$$\boldsymbol{f}(t) = \sum_{i}^{N} \lambda_i \phi_i(t), \tag{4.3}$$

where the basis function coefficients λ_i are computed by solving a linear system of equations,

$$\boldsymbol{f}(t_i) = \Delta\boldsymbol{x}_i, \quad \text{for } i = 1..N. \tag{4.4}$$

Note that the coefficients λ_i are vectors with the same dimension as $\boldsymbol{f}$, i.e., the dimension of the state.

The adjustment necessary for the multi-constraint IKD solver is that each iteration must produce an appropriate update to each $\Delta \boldsymbol{x}_i$. In Chapter 3.1, we computed numerical partial derivatives with respect to the bell shaped basis magnitude, and found a least squares solution for the update with a projection (computed with a dot product). In the case of two constraints i and j we have d_i influenced by a magnitude adjustment for constraint j, but we avoid the expense of computing all the numerical partial derivatives by fixing only one constraint at a time. Thus we have an inner loop that consists of computing the KD state at t_i, the error $\boldsymbol{e}_i$, the updated KD state for trajectory $\boldsymbol{x}_K + f + \boldsymbol{e}_i \phi_i$, the update for $\Delta \boldsymbol{x}_i$ (using Equation 3.4 or Equation 4.2), and then finally recomputing the interpolation function weights. This approach, similar to Gauss Seidel iteration, works well because the effect of ϕ_i on $\boldsymbol{x}_{KDi}$ is typically much larger than at $\boldsymbol{x}_{KDj}$.

The technique we use to solve the multi-constraint IKD problem is summarized in Algorithm 2, and consists of a nested loop of adjusting the correction $\boldsymbol{f}$ to fix each of the violated constraints, until all constraints are sufficiently satisfied or a maximum number of iterations is reached.

Solving for the basis function coefficients λ_i is fast. To solve the interpolation function, we can compute an LU decomposition [23], from which we can find λ_i using a back solve. Repeated solves of the interpolation function can be done quickly because we can reuse the same decomposition (the basis functions and their centers do not change).

Algorithm 2 Inverse Kinodynamics Multi-Constraint Solve

Input: constraints $\boldsymbol{x}_i$ or p_i at t_i, for $i = 1..N$, δ
Output: state correction curve f

$itr \leftarrow 0$
$E \leftarrow \infty$
$\Delta\boldsymbol{x}_i \leftarrow 0$, for $i = 1..N$
$\boldsymbol{f} \leftarrow$ SolveInterpolation($\Delta\boldsymbol{x}$)
while $itr{+}{+} <$ *maximum* and $E >$ *threshold* **do**
  for $i = 1 \rightarrow N$ **do**
    $\boldsymbol{x}_{KDi} \leftarrow$ SimulateKD($\boldsymbol{x}_K + \boldsymbol{f},\ t_i,\ \delta$)
    $\boldsymbol{e}_i \leftarrow$ compute using Equation 3.2 or 4.1
    $\tilde{\boldsymbol{x}}_{KDi} \leftarrow$ SimulateKD($\boldsymbol{x}_K + \boldsymbol{f} + \boldsymbol{e}_i\phi_i,\ t_i,\ \delta$)
    $\Delta\boldsymbol{x}_i \leftarrow$ compute using Equation 3.4 or 4.2
    $\boldsymbol{f} \leftarrow$ SolveInterpolation($\Delta\boldsymbol{x}$)
  end for
  $E \leftarrow 0$
  for $i = 1 \rightarrow N$ **do**
    $\boldsymbol{x}_{KDi} \leftarrow$ SimulateKD($\boldsymbol{x}_K + \boldsymbol{f}$, t_i)
    $E \leftarrow E + \|p(\boldsymbol{x}_{KDi}) - p_i\|$ or $\|\boldsymbol{x}_i - \boldsymbol{x}_{KDi}\|$
  end for
end while

Note that the inner loop update could skip an update for a given constraint if its contribution to the error was known to be small. However, the size of this error can only be verified by recomputing the KD state as it is influenced by other changes to $\boldsymbol{f}$. The computation of x_{KDi} is the bulk of the cost.

4.4 Constraining velocities

When constraining a pose or an end effector position, we might also want to set constraints on velocities. For instance, we may want the hand of a character to touch the surface of an immobile object. The hand end effector must satisfy both position and velocity constraints, meaning it must reach the target at the time of contact

and have zero velocity. The desired velocity can follow the original velocity of the animation or can be set to achieve a different velocity at the time of the constraint.

We can solve the IKD problem for constrained velocities in a similar manner to the position problem, and likewise solve for simultaneously constrained position and velocity. Again, the IKD solution comes from layering a correction overtop of the kinematic trajectory.

Suppose that at time t_i we have desired state velocity $\dot{\boldsymbol{x}}_i$, or alternatively, a desired end effector velocity $\dot{p}_i$. Instead of adding a bell shaped curve to change the velocity, we will add a wiggle to change the velocity $\dot{\boldsymbol{x}}_K(t_i)$ without changing $\boldsymbol{x}_K(t_i)$. We use the derivative of the bell shaped position correction basis function as a basis function for setting the derivative,

$$\psi_i(t) = \frac{\partial}{\partial t}\phi_i(t), \tag{4.5}$$

though this function could likewise be selected by the animator.

For simplicity, suppose we are dealing with a set of N constraints on both position and velocity at times t_i, for $i = 1...N$. To deal with N position and velocity constraints in close proximity we use an interpolation of the corrections $\Delta\boldsymbol{x}_i$ with velocities $\Delta\dot{\boldsymbol{x}}_j$ necessary to correct the kinematic trajectory. Thus, the interpolation function has the form

$$\boldsymbol{f}(t) = \sum_{i}^{N}(\lambda_i\phi_i(t) + \beta_i\psi_i(t)). \tag{4.6}$$

Again, the basis function coefficients λ and β can be found by solving the system of $2N$ linear equations for each dimension of the state, given by the required corrections

and correction velocities:

$$\boldsymbol{f}(t_i) = \Delta \boldsymbol{x}_i, \quad \text{for } i = 1..N, \tag{4.7}$$

$$\frac{\partial \boldsymbol{f}(t_i)}{\partial t} = \Delta \dot{\boldsymbol{x}}_i, \quad \text{for } i = 1..N. \tag{4.8}$$

It is important to observe that we update the desired velocity correction $\Delta \dot{\boldsymbol{x}}_j$ by comparing the desired velocity $\dot{\boldsymbol{x}}$ with the velocity of the KD trajectory. The velocity of the dynamic simulation which produces $\boldsymbol{x}_{KD}(t)$ does *not* give us this KD velocity (i.e., it is not the dynamic simulation velocity which we want to control). Instead, we must approximate this KD state velocity from successive frames of the KD state,

$$\dot{\boldsymbol{x}}_{KD}(t_i) \approx \frac{1}{h}(\boldsymbol{x}_{KD}(t_i) - \boldsymbol{x}_{KD}(t_i - h)). \tag{4.9}$$

We measure the difference to set the velocity error $\dot{\boldsymbol{e}}_i$, with which we compute a new KD state, and ultimately find an update to the required velocity correction $\Delta \dot{\boldsymbol{x}}_i$ (with a computation similar to Equation 3.4).

In the above example, we are considering a target velocity on the entire state. If instead our constraint is only on the end effector of a skeleton, then the approach is slightly different. In this case, we compute the approximate KD end effector velocity,

$$\dot{p}_{KD}(t_i) \approx \frac{1}{h}(p(\boldsymbol{x}_{KD}(t_i)) - p(\boldsymbol{x}_{KD}(t_i - h))). \tag{4.10}$$

The difference between this velocity and the artist requested end effector velocity $\dot{p}_i$ is then mapped to a state error,

$$\dot{\boldsymbol{e}}_i = J^{+}(\dot{p}_i - \dot{p}_{KD}(t_i)). \tag{4.11}$$

where J^+ is a pseudoinverse of the end effector Jacobian $J = \partial p / \partial \boldsymbol{x}$ evaluated at pose $\boldsymbol{x}_{KD}(t_i)$. Again, this error is used to update the required velocity correction $\Delta \dot{\boldsymbol{x}}_i$, and the process is repeated until our IKD algorithm has converged or we reach a maximum number of iterations.

4.5 Dynamic blend shape IKD

While the previous sections focus on skeletal animation, the same ideas are applicable to elastic tissue deformation. Particularly in the context of facial animation, this articulated deformation is tediously authored by animators by keyframing linearly blend shape targets. Overlaying secondary jiggle and other dynamic nuance currently comes at the cost of letting dynamics have the "final word" on the animation, with no guarantees of hitting certain expressions. IKD allows one to overlay this desired secondary dynamics in a kinematic setting and further specify critical poses as target shapes to be precisely interpolated, independent of the kinematically authored blend shape animation. A loosened facial animation can also be kept in sync with the environment (like taking a puff from a cigarette or sip from a glass) or an audio track by adding checkpoints from the kinematic trajectory as IKD targets, so the final facial trajectory has a limited deviation from the kinematic input. We implement IKD as a deformation that tracks control points on a shape using springs and dampers as in Müller et al. [16]. Figure 4–4 shows examples of pose constraints applied to a kinodynamic trajectory for two different characters.

The inverse kinodynamic solution follows the same algorithm presented above. Note that the correction update follows Equation 3.4, and is very easy to compute as we simply need the difference between the kinodynamic state and the target. The

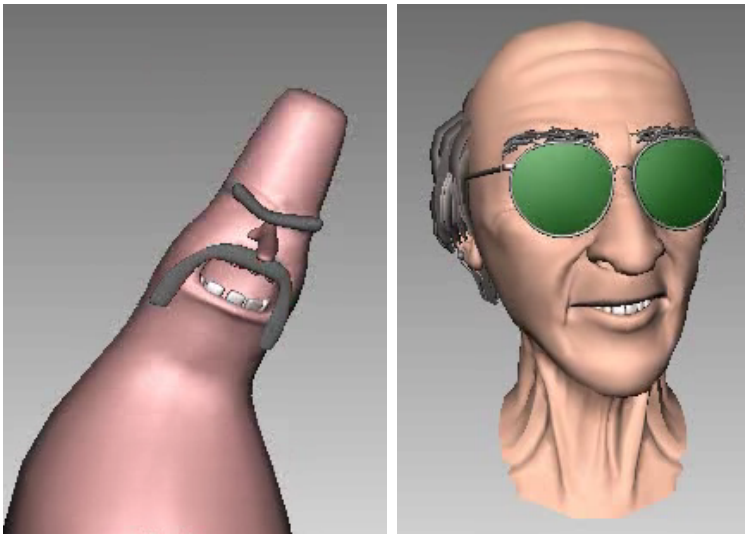


Figure 4–4: Two facial animation IKD examples (see accompanying video). Left, a temporal pose constraint produces a head tilt. Right, a temporal pose constraint produces a smile.

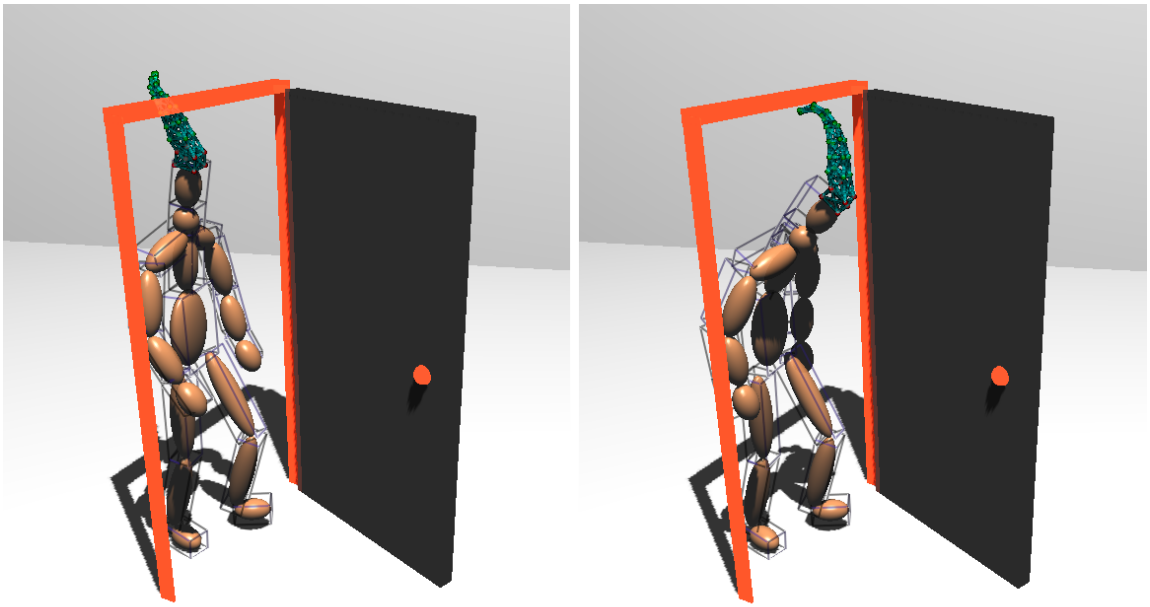
techniques for dealing with multiple constraints and velocity constraints are likewise similar to those describe in previous sections.

4.6 Deformable objects secondary dynamics IKD

While skeletal motion plays an important role in character animation, the motion of the character is not the only concern of the animators. Animators need to have complete control over the elements in their scenes. Such elements can consist of passive deformable objects attached to kinametically driven bodies.

Controlling secondary dynamics of such deformations can be tricky since the motion purely relies on the kinematic motion driving them and creating their deformation. For example, consider a scenario where a character with a floppy hat must walk through a door at a specific time in the animation without the floppy hat hitting any part of the door frame.

In such cases, the IKD can be used to control the deformable object by altering the kinematic motion which is driving the secondary dynamics. In our test case, a deformable hat (see Figure 4–5 a) was driven by the motion of our character's head. The technique for skeletal IKD described in Section 4.1 can be used here, to control the position and velocity of the hat.



(a) (b)

Figure 4–5: KD Deformation of the hat before (a) and after (b) the IKD solution.

In order to control the position of the tip of the hat, at each time frame the tip was considered as an end effector (see Figure 4–6) with a defined distance from the head and the IKD was deriving the appropriate kinematic deformation to achieve the desired scenario. As shown in the results (see Figure 4–5 b), a change in the character's posture allows the hat to clear the door frame. The time windows selection plays an important role in the simulation result. A good time window will produce

a KD approximation that end up matching well enough the true dynamics and the produced trajectory will appear to be physically correct. For the hat example, we chose the time window based on a visual threshold where the hat vibrations caused by the maximum acceleration of the walk motion were not visible anymore. By this the trajectory of the hat looked natural. (see Chapter 5 for more details).

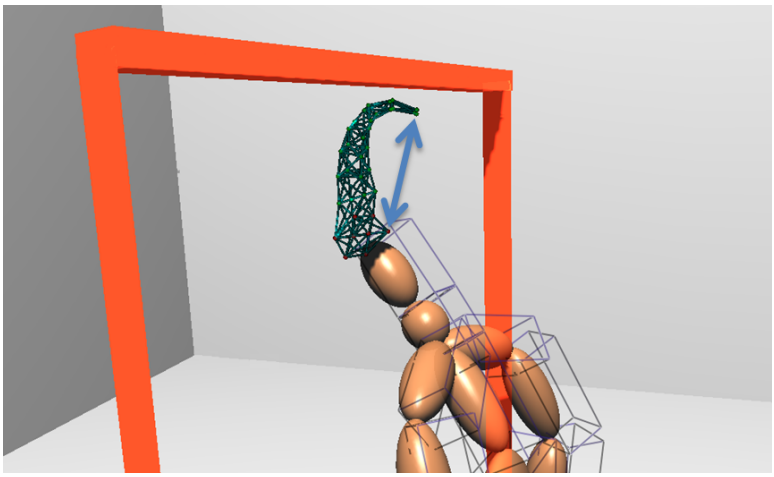


Figure 4–6: The tip of the hat is considered as an end effector of the character to solve the IKD problem.

CHAPTER 5
Temporal window selection

Setting the size of the temporal window δ has an important influence on the quality and cost of the kinodynamic trajectory. We want a small window to make it cost effective to simulate kinodynamic states on the fly, but the window also needs to be long enough to produce the desired secondary dynamics effects. In this chapter, we show how analysis can guide us in the selection of the temporal window size though in many cases, it is possible to select the temporal window by hand.

5.1 Selecting temporal windows manually

While it is useful to have guidelines for choosing a good temporal window, it is often easy to select a reasonable window by hand. This can be done by simulating the physical system in response to an impulse and visually selecting the time at which vibrations are no longer visible. This is the technique we use for all of our skeletal animations. For the passive deformable systems attached to kinematically driven bodies (our hat example in Chapter 4.6), we selected our temporal window based on a visual threshold where vibrations caused by the maximum acceleration of the system (i.e., the walk motion) were not visible.

5.2 Selecting temporal windows based on system's physical analysis

The temporal window can also be computed based on the physical parameters of the system. Consider the one dimensional example of a particle of mass m attached

to a damped spring,

$$m\ddot{x} + c\dot{x} + kx = 0, \tag{5.1}$$

and let the initial position be zero, $x_0 = 0$. Let $J_{\max}$ be the maximum impulse that we require to follow a given kinematic trajectory. We can also view this as the maximum acceleration in the trajectory where $\ddot{x}_{\max} = \frac{J_{\max}}{hm}$ with h being the simulation step size. Selecting this maximum impulse as acting just before time zero gives us a velocity initial condition for our differential equation, $\dot{x}_0 = J_{\max}/m$, and the exact solution (see [15] for more details) will have the form

$$x(t) = Ae^{\gamma t}\sinh(\omega t), \tag{5.2}$$

where γ is the decay rate, ω is the frequency, and A is the amplitude,

$$\gamma = -\frac{b}{2m}, \tag{5.3}$$

$$\omega = \frac{\sqrt{b^2 - 4mk}}{2m}, \tag{5.4}$$

$$A = \frac{2J_{\max}}{\sqrt{b^2 - 4mk}}. \tag{5.5}$$

Based on the physical parameters of the system, when the trajectory is under damped or critically damped, the function $Ae^{\gamma t}$ provides a bound on magnitude of the simulation. Given a minimum perceptual magnitude $x_{\min}$, we can choosing our temporal window δ by solving $Ae^{\gamma\delta} = x_{\min}$,

$$\delta = \frac{1}{\gamma}\ln\left(\frac{x_{\min}}{A}\right). \tag{5.6}$$

When the solution is over damped, the only difference is that we must then solve for δ as the root of a sum of exponential decays minus $x_{\min}$.

Systems with multiple degrees of freedom such as our hat example in Chapter 4.6, can be analyzed in a similar way, provided that we can assume that the equations of motion are a linear ODE, or that a linearized version of the system provides good predictive behavior. Using linear (or linearized) forces at the equilibrium pose of the system, we can diagonalize the system to compute decay rates γ_i and frequencies ω_i for a set of independent oscillators (see [25] for more details).

For systems with multiple degrees of freedom the equation of motion can be written as follows:

$$M\ddot{x} + C\dot{x} + Kx = 0 \tag{5.7}$$

where M, K and C are mass, stiffness and damping matrix. In order to simplify the solution of our multiple DOF system to multiple superimposed linear 1 DOF systems, we diagonalize the system by taking the eigenvalue decomposition of the stiffness matrix (with the assumption that, the mass matrix is identity and masses are all equally distributed). So the general form of the equation of motion can be written such that

$$\tilde{M}\ddot{x} + \tilde{C}\dot{x} + \tilde{K}x = 0 \tag{5.8}$$

in which $\tilde{M}$ and $\tilde{K}$ are diagonal matrix and $\tilde{C}$ is dense, but can be sparse if we use Rayleigh damping. That is, the damping will be a proportion of diagonalized M and

K matrices:

$$C = (\alpha M + \beta K) \Rightarrow \tilde{C} = diag(\alpha m_i + \beta k_i). \tag{5.9}$$

The eigenvalues of the stiffness matrix will tell us about the natural vibration frequencies of the system. We can do a change of coordinates using the vibration modes Φ (the eigenvectors of our stiffness matrix) such that $x = \Phi q$. By applying the coordinate change in our ODE we will have a new equation of motion:

$$m_i \ddot{q}_i + (\alpha m_i + \beta k_i)\dot{q}_i + k_i q_i = 0. \tag{5.10}$$

In this case the system will consist of superimposed linear systems which are independent from each other and they can be each solved as 1 DOF systems therefore we can rewrite the equation of motion for our system as

$$q_i(t) = Ae^{\gamma_i t} \sinh(\omega_i t) < Ae^{\gamma_i t} \tag{5.11}$$

where q_i is the ith mode displacement at time t. Given the maximum impulse (based on the accelerations of the kinematic trajectory) on each diagonalized degree of freedom, we can then compute a conservative bound for our temporal window by solving for δ as the root of a sum of exponential decays such that

$$\sum_i (Ae^{\gamma_i \delta})^2 < {x_{\min}}^2. \tag{5.12}$$

CHAPTER 6
Workflow interface

In this chapter we present a workflow example. We explain how animators can use IKD to create desired animations and we describe the types of controls they have for editing the animations. Figure 6–1 shows a workflow for creating an animation of a character punching a target.

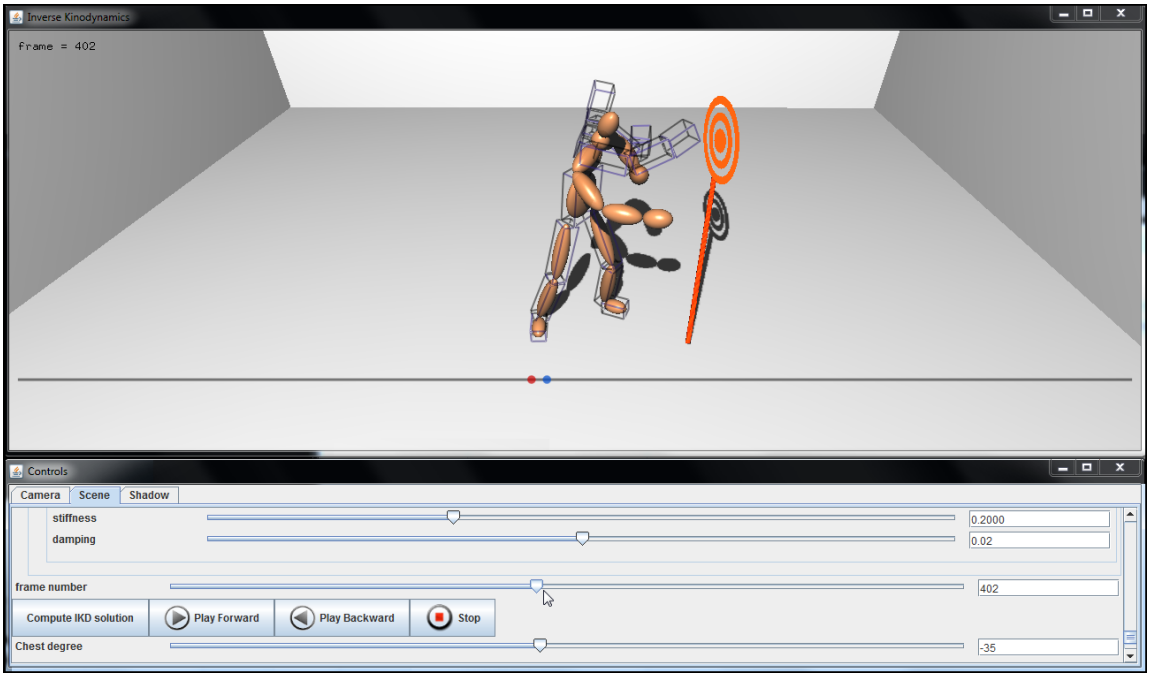


Figure 6–1: Workflow frame work showing tools for animators to scrub back and forth in animation and change animation parameters.

The interface has a time frame slider (see Figure 6–2) that the animators can scrub back and forth to get to a desired frame of the animation. To play the animation, three buttons can be used: play forward, play backward, and stop. They allow the animator to play the animation and observe the result. The play speed of the animation can be set manually. Finally, a Compute IKD Solution button is used to compute the new IKD solution for the animation after the edits. Other parameters such as the physical parameters of the character, correction function parameters and animation parameters, are included within the simulation platform.

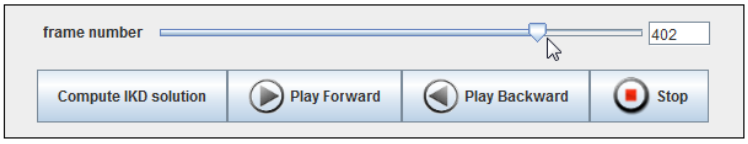
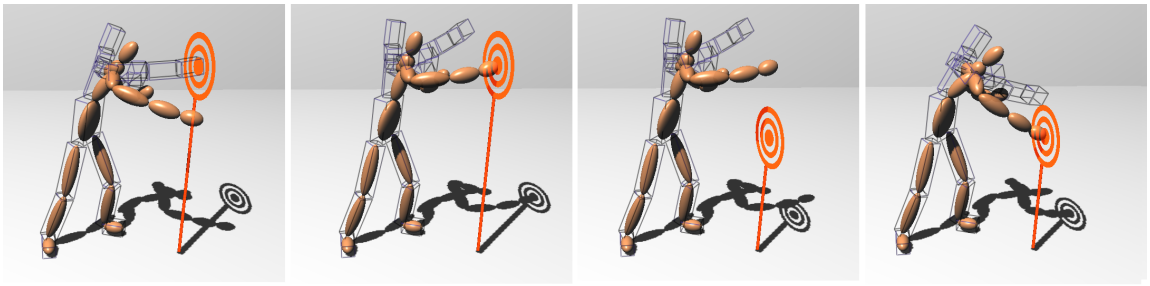


Figure 6–2: Workflow Control Panel

To start, the animator loads the punch example either by keyframing it or importing the motion capture. The kinematic trajectory is then shown by a wirecube character and the KD character is displayed by an orange character. The animator can change the parameters which will influence the final pose and dynamics (for instance, to produce a drunk character). To see the effect of the edits on the final KD animation the animator can scrub back and forth between animation frames. If further local edits must be done they can be performed in real-time on the character joint angles and the KD results will be displayed realtime to the animator. This is the WYSIWYG concept that was introduced in Chapter 1.

The animator might also need more control over the animation such as setting a specific target in the animation for the punch (See Figure 6–3). Imagine the motion

capture initially punches a target p_i at time t_i but then after changing relaxation and tension parameters, it would not satisfy the target anymore (Figure 6–3-a). The animator scrubs to the specific frame (t_i) and sees the error in the punch and can correct it by computing the correct IKD solution (Figure 6–3-b). If new target points must be set, the animator sets them (Figure 6–3-c) and computes new IKD solutions and interface displays them (Figure 6–3-d).



(a) (b) (c) (d)

Figure 6–3: (a) shows a KD state (orange) failing to punch a target. (b) is a modified motion capture pose (wire-frame) that produces a KD state which hits the target. The previous solution fails to punch a modified target position (c). A satisfied punch can be achieved by recomputing the IKD solution for the new target (d).

The ability to see the effect of edits in real-time is what makes the interface a powerful tool for animators. Besides character animation, we also include deformable objects in our simulation and control their pose (see Chapter 4.6). We use the interface to generate all our results as explained in the next chapter.

CHAPTER 7
Results and discussion

Please refer to the accompanying video for examples of our results. We provide a short description of these results, referring to Figure 7–1 which shows snapshots of the different scenarios.

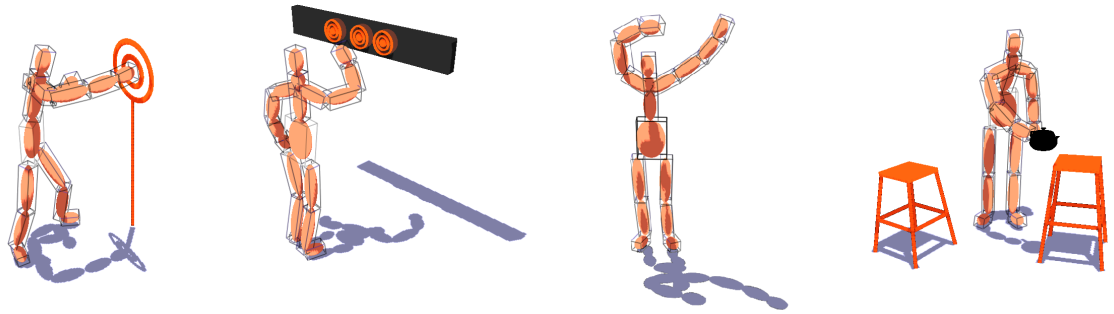
7.1 Convergence

It is important to discuss issues with the convergence of our IKD algorithm. If there are lots of abrupt motions in the kinematic trajectory, then the resulting simulation could be chaotic in nature. As such, we might not expect a small change in the joint angles to produce a predictable result, even if we smoothly and slowly blended in and out of this desired trajectory. While we do not assume linear dynamics, we do assume that the function mapping $\boldsymbol{x}_K$ to $\boldsymbol{x}_{KD}$ is smooth "enough".

While the convergence rate of our IKD algorithm depends on the actual scenario, Figure 7–2 compares the convergence rates achieved using different temporal widths of the correction function. Here, the IKD problem is the target punching example from Figure 4–1, and the example motions with the two different σ can be seen in the accompanying video.

7.2 Limitations

We note that in articulated character scenarios where we are constraining the end effector at specific moments in time, our implementation of the IKD algorithm



(a) (b) (c) (d)

Figure 7–1: From left to right, (a) punch, (b) control panel, (c) YMCA's "C" pose , (d) position constraint for grasp.

converges quickly, but does not make any progress once the error falls into the sub-millimeter range. This is because we are using the Open Dynamics Engine (ODE) to compute the simulations that produce our KD states. While repeating simulations using the same initial conditions should produce the same results, aggressive optimizations within ODE make use of randomization. This does not present a problem as the error in end effector placement is significantly smaller than the overall size of the articulated character.

We note that there are sometimes situations where the algorithm does not converge, specifically in the case where position and velocity constraints are solved together, for instance, when we ask for a high velocity punch that hits a target. We can address this by only using partial updates to $\Delta \boldsymbol{x}$ and $\Delta \dot{\boldsymbol{x}}$, and we have observed successful convergence using half steps, though at the cost of slower convergence.

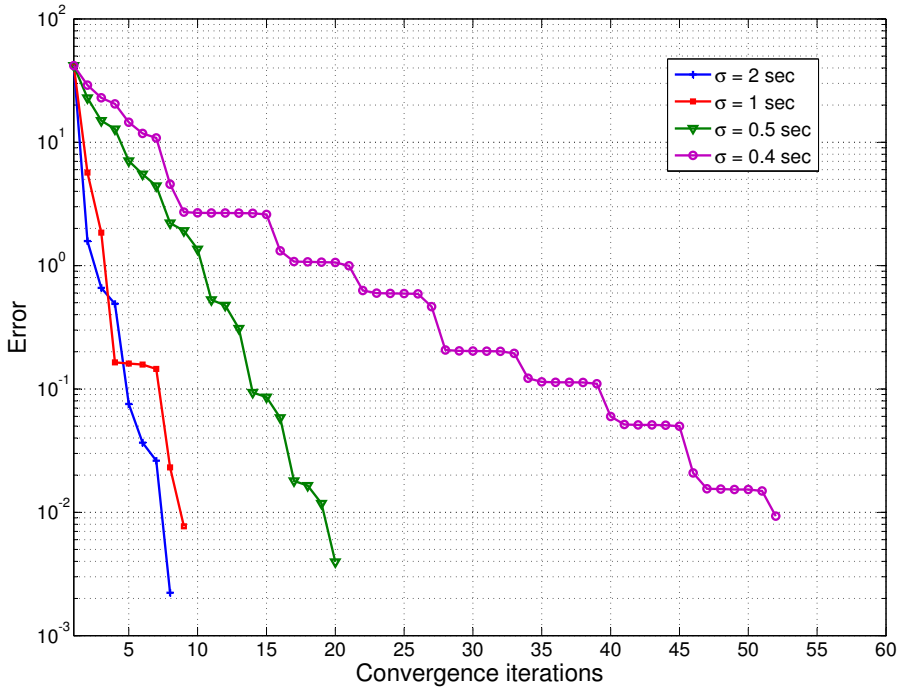


Figure 7–2: IKD convergence rates for the punch scenario using a bell shaped correction function with different temporal widths (2, 1, 0.5 and 0.4 sec). The error represents end effector error, measured in cm. IKD convergence can be slower when a small temporal width is used for the correction function. Error threshold was set to be 10^{-2} cm.

7.3 Timings

In our discussion of time window selection, we noted that the maximum acceleration in the kinematic trajectory will influence the size of the temporal windows (large accelerations will require longer temporal windows in order for oscillations produced by these accelerations to become imperceptible). This is also true for the altered trajectory which includes the correction to solve a given IKD problem. We are using smooth bell shaped curves to add this displacement, so generally the accelerations

due to the correction will be small. But if we set the temporal width of this curve to be small, then the IKD solution will need to involve a very large displacement to the kinematic trajectory to force the dynamic trajectory to the desired target, thus requiring larger temporal windows for computing a KD state.

IKD Skeletal animation examples were generated with our Java implementation which uses ODE (Open Dynamics Engine) to simulate the forward dynamics. Motion capture data was obtained using a 12 camera optical tracking system. On an Intel(R) Core i7, 3.2 GHz processor, the KD takes roughly 0.01 s to generate the resulting frame for a time window of 0.3 s (30 frames), which allows for interactive scrubbing of the time line. The IKD solution however depends on the complexity of the scenario and the size of the time window and it may vary from case to case. In the example of punch scenario the IKD takes roughly 0.2 s to compute the new solution for a time window of 0.3 s. Figure 7–2 shows the number of iterations (i.e., time elapsed) for the IKD algorithm to converge for correction functions with different temporal widths.

IKD has also been implemented as a Maya 2011 deformer for control point shapes that track a kinematic trajectory using a spring and damper simulation. On an Intel i7, 1.87 GHz processor, the model in Figure 4–4 (approximately 1200 vertices) takes 13.56 s (of which 7.34 s is external to the KD algorithm) to update 50 frames with a KD window of 1 s (25 frames) resulting in a reasonable interactivity of 8.03 fps.

7.4 Discussion

The shape of the bell shape curve we use to modify the kinematic motion directly affects the motion which is produced. We use Gaussian shaped curves in our examples

because they are simple and smooth. We effectively treat them as if they have compact support, and could easily use any other ease-in-ease-out curve of a desired shape and support, and we leave the selection of this curve to the animator. That is, the width of the Gaussian is selected by the animator; a wide curve will produce a smooth anticipatory motion, while a short curve will produce a motion that abruptly moves to meet the constraint with a larger acceleration (and in turn, a larger follow through). While we only look at symmetric curves, any smooth artist created ease-in ease-out curve can be used. For instance, a non-symmetric correction curve can be designed to create a quick reaction followed by a slow return to the unmodified trajectory.

While we are adding constraints to deal with contact, we require the artist to specify these constraints. While contacts may naturally happen in the dynamic simulations that produce our kinodynamic states, we will only have a "memory" of contacts that happen in the temporal window. For instance, we cannot correctly handle a braid of hair which is normally at rest down the back of a character but flips over a shoulder with the turn of a head.

CHAPTER 8
Conclusions

In this thesis, we developed an inverse kinodynamics algorithm and created a usable interactive kinodynamic framework for animators allowing them to control their animation. A demonstration of the Maya implementation to a few keyframe animators was positively received. From a workflow standpoint, the animators felt they would have to consciously omit keyframing dynamic nuances but this would be a welcome change allowing them focus on the primary motion. For the approach to be used in practice they expressed a need for interface tools that make the addition and management of IKD targets user friendly. Our current implementation, while interactive for skeletal animation, is only interactive for shapes up to around 1000 control vertices. The vectorizable nature of our algorithm, however, makes it a good candidate for an faster GPU implementation.

The techniques in this thesis can be used in domains where history of the simulation can be omitted in order to achieve faster and computationally less expensive results. IKD can be used in computer games where secondary dynamics need to be added to the animation of the game objects and characters. Since computing deformations and new states is history free, the game engines can benefit from IKD to have a more physically realistic and visually plausible animation even though characters and objects are changing motion by abrupt user decisions and without the anticipation of the game engine.

As explained in Chapter 6, IKD can also play an important role in the domain of animation. Animators can have full power over their animations and receive instant feedback about their changes and perform further edits. The platform can be used as editing tools for animators to control and create new animations without worrying about the history of their past edits and give them freedom of working on purely kinematic animations and creating the secondary dynamics results desired in their animation.

8.1 Future work

In future work we would like to address the coupling of kinodynamic trajectories with fully dynamic environments via adaptive kinodynamic window sizes that are aware of collision events and other discontinuities in a full physical simulation. The IKD algorithm must determine the dynamic environment impulses to the system and based on them chose appropriate time windows. The adaptive window size calculation can be also used for convergence improvement with some limitations (discussed in Chapter 7).

Another direction for future work includes the problem of controlling velocity. Animators should be able to express their needs of positional constraints and the IKD should be able to give them not only solution for those constraints and optimum trajectory but also further edits over the proposed trajectory. They should be able to edit the proposed path while still maintaining all the constraints.

Other future work can be in exploring the biomechanic knowledge of human body postures in the IK solve mechanism of the IKD algorithm. The algorithm should give the most efficient solution while taking into account the biomechanical

human body postures. Further to this, the study of the biomechanic information can be used as a tool to create predefined human poses for different expressions and feelings so the algorithm can use such data while finding the optimum solution for the problem. As an example, the solution for the punch scenario can be tailored to represent people of different ages, body posture and state of mind while the algorithm is using the same motion capture data and still giving the accurate results.

An implementation of the IKD algorithm can also be done in animation software platforms such as Maya. This can help animators create their desired animations and control the edits while adding secondary dynamics to their keyframed animation. In summary, we propose the concept of Inverse Kinodynamics and present a first algorithm which opens up new possibilities for editing traditional keyframe animations that are augmented with secondary dynamics.

References

[1] Yeuhi Abe and Jovan Popović. Interactive animation of dynamic manipulation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '06, pages 195–204, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[2] Brian Allen, Derek Chu, Ari Shapiro, and Petros Faloutsos. On the beat!: Timing and tension for dynamic characters. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 239–247. Eurographics Association, 2007.

[3] Alexis Angelidis and Karan Singh. Kinodynamic skinning using volume-preserving deformations. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '07, pages 129–140. Eurographics Association, 2007.

[4] Jernej Barbič and Jovan Popović. Real-time control of physically based simulations using gentle forces. *ACM Trans. on Graphics (SIGGRAPH Asia 2008)*, 27(5):163:1–163:10, 2008.

[5] Miklós Bergou, Saurabh Mathur, Max Wardetzky, and Eitan Grinspun. TRACKS: Toward Directable Thin Shells. *SIGGRAPH (ACM Transactions on Graphics)*, 26(3):50, Jul 2007.

[6] Samuel R. Buss and Jin-Su Kim. Selectively damped least squares for inverse kinematics. *Graphics Tools*, 10:37–49, 2005.

[7] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive skeleton-driven dynamic deformations. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 586–593, New York, NY, USA, 2002. ACM.

[8] Patrick Coleman, Jacobo Bibliowicz, Karan Singh, and Michael Gleicher. Staggered poses: a character motion representation for detail-preserving editing

of pose and coordinated timing. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 137–146. Eurographics Association, 2008.

[9] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *J. ACM*, 40:1048–1066, November 1993.

[10] Sumit Jain and C. Karen Liu. Interactive synthesis of human-object interaction. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pages 47–53. ACM, 2009.

[11] Doug L. James and Dinesh K. Pai. Dyrt: dynamic response textures for real time deformation simulation with graphics hardware. In *SIGGRAPH*, pages 582–585, 2002.

[12] Michael Kass and John Anderson. Animating oscillatory motion with overlap: wiggly splines. *ACM Trans. Graph.*, 27:28:1–28:8, August 2008.

[13] Noah Lockwood and Karan Singh. Biomechanically-inspired motion path editing. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, pages 267–276, New York, NY, USA, 2011. ACM.

[14] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. *ACM Trans. Graph.*, 23:449–456, August 2004.

[15] L. Meirovitch. *Analytical methods in vibrations*. Macmillan series in advanced mathematics and theoretical physics. Macmillan, 1967.

[16] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24:471–478, July 2005.

[17] Michael Neff and Fiu Eugene. Modeling tension and relaxation for computer animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '02, pages 81–88, New York, NY, USA, 2002. ACM.

[18] Nam Nguyen, Nkenge Wheatland, David Brown, Brian Parise, C. Karen Liu, and Victor Zordan. Performance capture with physical interaction. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 189–195. Eurographics Association, 2010.

[19] Jovan Popović, Steven M. Seitz, and Michael Erdmann. Motion sketching for control of rigid-body simulations. *ACM Trans. Graph.*, 22:1034–1054, October 2003.

[20] Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. Interactive manipulation of rigid body simulations. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 209–217, 2000.

[21] Marc H. Raibert and Jessica K. Hodgins. Animation of dynamic legged locomotion. *SIGGRAPH Comput. Graph.*, 25:349–358, July 1991.

[22] Kwang Won Sok, Katsu Yamane, Jehee Lee, and Jessica Hodgins. Editing dynamic human motions via momentum and force. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 11–20, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[23] G. Strang. *Introduction to applied mathematics*. Wellesley-Cambridge Press, 1986.

[24] Adrien Treuille, Antoine McNamara, Zoran Popović, and Jos Stam. Keyframe control of smoke simulations. *ACM Trans. Graph.*, 22:716–723, July 2003.

[25] Andrew Witkin. Physically based modeling: Principles and practice – constrained dynamics. In *COMPUTER GRAPHICS*, pages 11–21, 1997.

[26] Andrew Witkin and Michael Kass. Spacetime constraints. *SIGGRAPH Comput. Graph.*, 22:159–168, June 1988.

[27] Chris Wojtan, Peter J. Mucha, and Greg Turk. Keyframe control of complex particle systems using the adjoint method. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 15–23. Eurographics Association, 2006.

[28] KangKang Yin, Michael B. Cline, and Dinesh K. Pai. Motion perturbation based on simple neuromotor control models. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, PG '03, pages 445–, Washington, DC, USA, 2003. IEEE Computer Society.

[29] Victor Brian Zordan and Jessica K. Hodgins. Motion capture-driven simulations that hit and react. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics*

symposium on Computer animation, SCA '02, pages 89–96, New York, NY, USA, 2002. ACM.

[30] Victor Brian Zordan, Anna Majkowska, Bill Chiu, and Matthew Fast. Dynamic response for motion capture animation. *ACM Trans. Graph.*, 24:697–701, July 2005.