Chapter 1

Kinematic editing

> If all you have is a hammer, everything looks like a nail.
>
> Abraham Maslow

In classical mechanics, kinematics is the study of the motion of a system of rigid bodies without considerations of dynamical quantities, such as mass or force. In the field of computer animation, kinematic editing of a skeletal model refers to the use of kinematics equations in order to change the configuration of the skeleton through manipulating its joint parameters. Computer animation software systems are generally based on kinematic editing of the character, where the animated motion is synthesized by interpolating the joint parameters between critical postures known as keyframes.

Through varying the joint angles of a character over time, keyframe animations can create a variety of natural motions, from waving hand to running. In many cases it is not convenient to control the individual joints of a character, a technique also known as forward kinematics, as it is the position of a hand or foot that is more important than the joint angles. In this case, inverse kinematics (IK) can provide a solution for changing a collection of joint angles to satisfy the given position constraint. This is useful for placing feet on the ground, for creating reaching motions, or any motion that involves controlling the posture of a character in relation to its surrounding environment.

The current modern IK software solutions (such as Maya) offer extensive toolsets specific to humanoids to ease the creation of natural motion. Furthermore, data-driven IK solutions are an excellent way to ensure that an animation exhibits natural postures and motion. However, data and heuristics used for humanoids will not generally apply to characters of different morphologies, such as quadrupeds or imaginary creatures. Additionally, they do not explicitly include dynamical solutions to help the animator in creating physically plausible character animation. For example, creating a basketball slam dunk motion can be a very demanding task for three reasons. First, in accordance to the physical law of conservation of momentum, the center of mass of the character should follow a ballistic path during the jump. Second, any postural edits to this sequence will require re-adjustment of the whole motion as new postures will likely change the trajectory of the

center of mass in the flight phase. With the current keyframing systems, such cases often require an iterative and somewhat cumbersome refinement process in order to ensure the realism of the synthesized motion. Lastly, the preservation of linear momentum demands a constant horizontal and an accelerated vertical velocity due to gravity. As a result it is critical to set a physically consistent duration for the jump motion interval, a parameter that is hard to find through trial and error by the keyframe animator.

In this chapter, we present a novel animation solution for physically plausible and intuitive keyframing (PHYSIK). Our framework describes high level physics-based IK handles, such as center of mass and angular inertia, that can help generate character animations with complex dynamics, including swinging, jumping and diving. The underlying IK solver does not rely on the character morphology. Additionally, our work provides physics-based template trajectories to assist the animator in authoring physically based motions with fewer keyframes. The presented work in this chapter is the first dynamics-aware keyframing system that allows for creating physically plausible animations from scratch through animator-friendly control handles and interface.

1.1 Overview

PHYSIK describes a collection of techniques that not only allows animators to easily control the center-of-mass (CM), angular velocity and inertia of a character, but also gives them complete freedom to create exaggerated or impossible motions. The CM, angular velocity and inertia become IK handles and can be animated in the same way that hands and feet are constrained in existing IK solvers. The challenge is that most IK solvers use joint-space approaches to parameterize control in terms of individual joint actions, yielding only local IK solutions. A physics-based IK handle is a function of all joints, and hence will require a global IK solver. We show how using a prioritized global IK solver helps with this issue, where a mixture of high priority physical and low priority postural constraints are set by the animator to synthesize character motions.

Another aspect of PHYSIK is dealing with various morphologies. While in conventional IK solvers any changes to the morphology of the character depends on retuning the keyframed trajectories, in our system, in contrast, adding a backpack, large boots, or a tail, need not have an adverse effect on the momentum control of a currently keyframed trajectory.

Figure 1-1 highlights a simple application of our technique to keyframe a monkey bar example. In this case, the animator selects a pendulum motion template trajectory for the swing phase and a projectile motion template for the flight phase before landing. The pendulum template has the constraint that the CM trajectory should remain on a circular path whose center is set to be the pivot of the pendulum. Likewise the ballistic motion has the constraint that the linear momentum of the character is only affected by gravity, and thus the center of mass will undergo a constant vertical acceleration of $-9.8\ \mathrm{ms}^{-2}$. The animator can adjust the arc of the pendulum and the jump through a variety of controls such as the pivot position, maximum height, initial velocity, initial position, end position, and time of flight. Because these parameters are coupled, our tool automatically adjusts the parameters left unconstrained by the animator in order to satisfy the constraints of the template trajectories. Edits to the character's pose can be made during the swing and flight phases while preserving the desired CM trajectory.

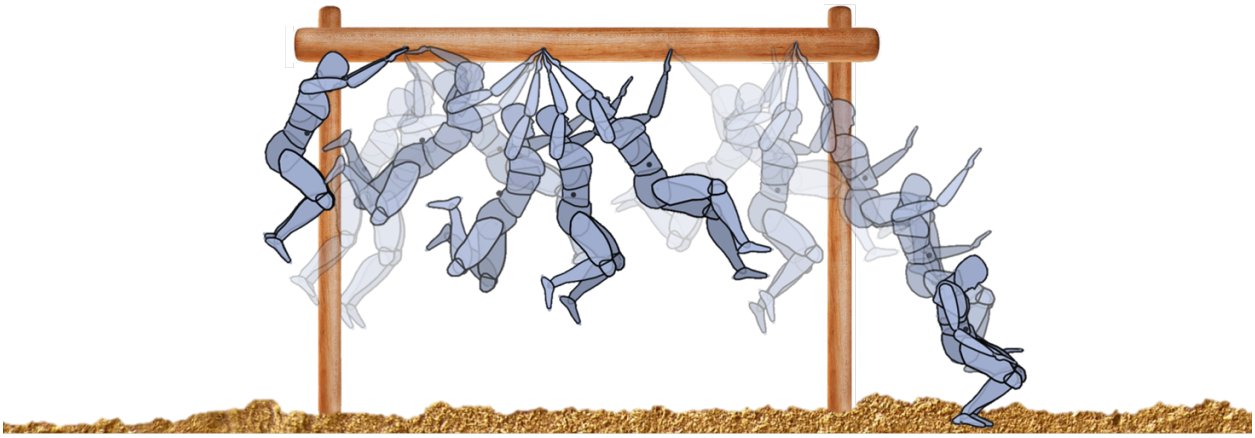

Figure 1-1: *Creating successive swing motions followed by landing via constraining the interpolated pose keyframes with seven physically plausible pendulum trajectory templates (swing) as well as a ballistic path at the end of the motion. The trajectory templates control the center of mass of the character. Hand and feet pinning constraints help the character to achieve natural grasps onto the monkey bar as well as a plausible landing posture. To avoid clutter in the image, the initial and the end poses as well as the first swing motion are shown by dark blue, while the other poses are shown by lighter shades.*

To achieve natural grasping of the monkey bar as well as to set the stance at landing, we set appropriate pinning constraints for the hands and feet (heel and toes). We regulate the weights of these pins through keyframes to produce a graceful transition between swings, and similarly, into the final landing phase. Finally, the center of mass is smoothly animated with Hermite curves to prepare for transitions between the swing motions, and similarly to stop the character in landing. See Sections 1.4 and 1.5 for other examples that demonstrate the different template trajectories that we provide in our interface.

Creating physically plausible character motion through keyframing is a challenge for animators, and we see the results of our work being relevant to character animators everywhere. Five components make up the main contributions of our work:

- center-of-mass and rotational inertia IK handles;
- conservation of angular momentum in a ballistic motion;
- regulation of IK constraint compliance in a global solver;
- arbitrary re-rooting for simple sub-structure pose control;
- handle trajectory generation from physics-based templates.

While most of the demonstrations in this work are 2D, we also show how our work generalizes to 3D by presenting preliminary results for three-dimensional characters. The underlying principles of our interactive interface as well as the keyframing system make no assumption about the character dimension. To address both planar and 3D cases, we will first show how to compute control quantities in 3D and then explain how to adapt them to a 2D inverse kinematics solver.

1.2 Handles for physically plausible keyframing

In many IK problems, the objective is to control points at the end of limbs, such as a hand or foot. This can be solved by working with joints in isolated limbs. Hence, positioning of end points is often treated as a local IK problem. In contrast, there are cases where IK problem cannot be solved locally. For example, the CM is a function of the position of all limbs. In particular, changing the CM position becomes challenging in the presence of additional constraints at hands or feet. Thus, a solution that constrains the CM in addition to hands and feet is ideally best found with a global IK solver.

In order to provide the animator with tools to animate the rotational inertia and CM, we estimate the size and physical properties of the character's parts. We assume that this can be done automatically from the character's geometry. If the geometry is formed by set of closed meshes, then mass and rotational inertia of each segment can be computed from volume integrals with a reasonable material density. In the case of non-closed or more complex character geometries, we believe that it is not unreasonable to assume that the artist will also create a collection of simple collision proxies from which the mass and rotational inertia can be computed instead.

1.2.1 Derivation

In the following parts we identify the momentum related functions and show that we can use the gradient as a linear map in the control quantities. In order to derive equations for the inverse kinematics solver we need differential quantities of the desired control handles with respect to the joints. That is, given a forward kinematics map $f : Q \to \mathbb{R}^n$ and a desired configuration $f_d \in \mathbb{R}^m$, we would like to solve the equation $f(\theta) = f_d$ for some $\theta \in Q$, where n and m vary based on the type of joints and the controlled features. This is a root finding problem, and it may have multiple solutions, a unique solution, or no solution at all, as discussed by Murray et al. (1994). Solutions to this mapping are obtained by iterative solvers that require differentiating the forward kinematic map with respect to the control quantities.

For an articulated character made up of many bodies, each body has a coordinate frame, and the position of the frame in world coordinates is a function of the position and orientation of the root along with the joint angles of the character. We assemble all parameters of a character's pose into a vector θ, including the root position and orientation, and thus we can think of orientation and position of each body as functions of θ.

Controlling the CM position. The CM position is the weighted average of the CM positions of all bodies. Suppose we would like to know this quantity in some world coordinate frame w. Given the position of the CM of a link, we can write this as a function of θ,

$$c(\theta) = \frac{1}{m_\sigma} \sum_{b=1}^{N} m_b {}^w r_b(\theta) \tag{1.1}$$

where r_b and m_b are the position and mass of body b respectively, N is the number of bodies, m_σ is the total mass and leading superscript w on r denotes that the quantity is expressed in world coordinates. We may want to control the CM of the character so that it stays in a given position, or

follows a desired trajectory. For this we take partial derivative of Equation 1.1 with respect to θ

$$\frac{\partial c(\theta)}{\partial \theta} = \frac{1}{m_\sigma} \sum_{b=1}^{N} m_b \frac{\partial r_b(\theta)}{\partial \theta}. \tag{1.2}$$

Defining the terms for the CM and body Jacobians

$$\frac{\partial c(\theta)}{\partial \theta} = J_c \tag{1.3}$$

$$\frac{\partial r_b(\theta)}{\partial \theta} = J_b \tag{1.4}$$

we can rewrite Equations 1.2 as

$$J_c = \frac{1}{m_\sigma} \sum_{b=1}^{N} m_b J_b. \tag{1.5}$$

The jth block of J_b relates the change of joint angle j to the position of body b,

$$J_{bj} = \widehat{p_{bj}} R_{bj} \tag{1.6}$$

where p_{bj} and R_{bj} are the translational and rotational components of the transformation from joint coordinate frame j to body coordinate frame b, and $\wedge$ is a cross product operator in the form of a skew symmetric matrix for a 3×1 vector a:

$$\widehat{a} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}. \tag{1.7}$$

To adapt the center of mass handle to 2D we exclude the third dimension from the positional vectors, and the cross product operator will reduce to

$$\widehat{a} = \begin{bmatrix} a_y \\ -a_x \end{bmatrix}. \tag{1.8}$$

Note that the $\widehat{\ }$ notation is overloaded, yet it will be clear from context when it is used on 2D or 3D vectors. The primary use of the 2D cross product is for multiplication by a scalar angular velocity to get its contribution to compute a linear velocity of a point.

Controlling the rotational inertia. Total rotational inertia at the CM frame is the lower right 3×3 block of the generalized inertia tensor

$${}^{c}M = \begin{bmatrix} m_\sigma \mathbf{I} & \mathbf{0} \\ \mathbf{0} & I \end{bmatrix} \tag{1.9}$$

where $\mathbf{I}$ and $\mathbf{0}$ are constant identity and zero sub-matrix blocks, and I is the rotational inertia matrix.

We obtain ${}^{c}M$ by transforming body inertia matrices to the CM coordinate frame and summing,

$$ {}^{c}M = \sum_{b=1}^{N} {}^{b}_{c}Ad^{T} M_b \, {}^{b}_{c}Ad \tag{1.10} $$

where Ad is an *adjoint transformation* matrix [Murray et al. (1994)]

$$ {}^{b}_{c}Ad = \begin{bmatrix} R & \widehat{p}R \\ \mathbf{0} & R \end{bmatrix} \tag{1.11} $$

given R and p are the rotation and translation of the rigid transformation between the CM frame and the body frame, and the CM frame is aligned with the world frame. Since there might be off-diagonal non-zero elements in I due to the correlation of the inertia axes, we perform singular value decomposition (SVD) on I to extract the rotational inertia elements along each axis. For a planar character the rotational inertia reduces to a single scalar quantity (as opposed to a 3×3 matrix in 3D), which corresponds to the element along the axis that is orthogonal to the plane.

In 2D we can think of the rotational inertia as a disk with a radius that is proportional to the square of off-plane scalar element. Hence the expression for computing the total inertia becomes

$$ I(\theta) = \sum_{b=1}^{N} m_b (r_b(\theta) - c(\theta))^2 + I_b \tag{1.12} $$

where scalar I_b is the rotational inertia of body b. In order to compute the Jacobian we take partial derivative of Equation 1.12 with respect to θ

$$ \frac{\partial I(\theta)}{\partial \theta} = 2 \sum_{b=1}^{N} m_b (r_b(\theta) - c(\theta))^T \left(\frac{\partial r_b(\theta)}{\partial \theta} - \frac{\partial c(\theta)}{\partial \theta} \right) \tag{1.13} $$

and replace the relevant terms from Equation 1.3 and Equation 1.4

$$ J_r = 2 \sum_{b=1}^{N} m_b (r_b(\theta) - c(\theta))^T (J_b - J_c). \tag{1.14} $$

Linear and angular momentum. It is also useful to consider expressions for linear and angular momentum of the entire character. In the absence of contact, the angular momentum will be a conserved quantity, and linear momentum in the vertical direction will decrease at a constant rate due to gravity while linear momentum in the horizontal direction is conserved. Of note, the linear momentum is directly related to the CM velocity.

In 3-dimensional space, the velocity of a rigid body with respect to an inertial frame is written as a *twist*, and can be written as a 6 components vector in a given coordinate system. For instance, the twist of body b expressed in the coordinate frame attached to body b can be written as

$$ {}^{b}\zeta_b = \begin{bmatrix} v \\ \omega \end{bmatrix} \in \mathbb{R}^6, \tag{1.15} $$

where v and ω are linear and angular velocities, respectively. Likewise, the momentum of body b in coordinate frame b is a 6 component vector that is linearly related to the body velocity

$$ {}^{b}\Phi_b = \begin{bmatrix} L \\ H \end{bmatrix} = {}^{b}M\,{}^{b}\zeta_b \tag{1.16} $$

where L and H are the linear and angular momentum vectors, and ${}^{b}M$ is the inertia tensor.

Just like twists transform with the adjoint transformation, we use the adjoint transform (Equation 1.11) to change momentum from one coordinate to another, except that we use the inverse transpose. The total momentum of a multibody character, in coordinates at the CM of the character, is the sum of the momentum of all rigid bodies

$$ {}^{c}\Phi_\sigma = \sum_{b=1}^{N} {}^{c}_{b}Ad^{-T}\,{}^{b}M\,{}^{b}\zeta_b. \tag{1.17} $$

In two dimensions linear velocity v and linear momentum L become 2×1 vectors and angular velocity w and angular momentum H are scalar. We can likewise define the 2D adjoint transformation that transforms twists from coordinate frame a to coordinate frame b,

$$ {}^{b}_{a}Ad = \begin{bmatrix} R & \widehat{p} \\ \mathbf{0} & 1 \end{bmatrix} \tag{1.18} $$

where R becomes a 2×2 block and $\widehat{p}$ is the same as Equation 1.8. Let ${}^{b}\Gamma_b$ provide the twist of body b in coordinate frame b when multiplied by joint velocities $\dot{\theta}$. This is effectively the Jacobian of a rigid body's position, which is a function of the characters configuration θ. With this, we can then write an expression that relate the joint velocities to linear and angular momentum,

$$ {}^{c}\Phi_\sigma = \left(\sum_{b=1}^{N} {}^{c}_{b}Ad^{-T}\,{}^{b}M\,{}^{b}\Gamma_b \right) \dot{\theta} \tag{1.19} $$

from which the Jacobian is found as

$$ J_m = \sum_{b=1}^{N} {}^{c}_{b}Ad^{-T}\,{}^{b}M\,{}^{b}\Gamma_b. \tag{1.20} $$

The sum in Equation 1.19 interestingly contains the CM position Jacobian. That is, the upper two components of ${}^{c}\Phi_\sigma$ are the linear momentum of the entire character, which is same as the CM position Jacobian multiplied by the joint velocities.

The bottom components of the expressions given in Equation 1.19 and 1.20 provide a means to kinematically approximate and regulate the angular momentum of the character. While we can control the angular momentum by changing any of the joint angles, it is more practical to do this at the root so as to manipulate the angular velocity of the character as a whole, while letting the other constraints dictate what happens with all the other parts of the body. This way we can ensure the authored motion style is preserved by not changing the original keyframe animation of each body part when improving the physical plausibility of the character animation. As a result, we suggest

an alternative way of regulating the angular momentum through rotating the root. This method might lack the dynamical body motion nuances automatically emerging from a momentum control approach, but it allows for an easy way of improving physical realism with minimum changes to the character posture.

From Equation 1.17 we can obtain an expression that relates the character generalized rotational inertia and angular velocity to the angular momentum

$$^{c}H = {}^{c}I\,{}^{c}\omega \tag{1.21}$$

where the rotational inertia I is calculated from Equation 1.10, and the total angular velocity ${}^{c}\omega$ is

$$^{c}\omega = \sum_{b=1}^{N} {}^{c}_{b}R\,{}^{b}\omega, \tag{1.22}$$

where ${}^{b}\omega$ is computed by finite difference method, and ${}^{c}_{b}R$ is the rotation matrix that transforms the angular velocity of body b to the center of mass coordinate frame. In two dimensions the angular velocity becomes a scalar. Therefore, we can drop the rotation matrix from Equation 1.22, and likewise use the Equation 1.12 to calculate the total rotational inertia. This leads to an expression for the angular momentum where all quantities are scalars. In order to enforce a conservation of angular momentum during a motion of interest, it becomes straightforward to set the desired values for the angular momentum as well as the inertia, and compute the desired total angular velocity via $\omega = H/I$. As we are already dealing with quantities at the velocity level, the Jacobian expression J_{ω} will therefore contain zeros and ones for the joints we would like to exclude from or include in the solution. Excluding all joints except the root, the Jacobian matrix will have columns of zeros except for the one column that corresponds to the root rotation. A method that ensures conservation of the angular momentum in 3D will be similar to what we described here.

1.3 Solving the inverse kinematics problem

In this section we describe how to formulate and solve the inverse kinematics problem with a mixture of high and low priority constraints. While our solver is global in the sense that it uses full body configuration space to compute the solution, we also propose a method that allows for local edits to the character.

1.3.1 Physics-based solution

In order to solve for the joint angles that satisfy the physics-related control parameters in 2D we form a 4×1 vector with the desired handle quantities

$$g_{des}(\theta) = \begin{bmatrix} I_{des} \\ c_{des} \\ \omega_{des} \end{bmatrix}. \tag{1.23}$$

We also combine the three Jacobians that we computed so far into a weighted Jacobian matrix

$$J_w = W \begin{bmatrix} J_r \\ J_c \\ J_\omega \end{bmatrix} \tag{1.24}$$

where W is a 4×4 diagonal user-specified weighting matrix in the form of

$$W = \begin{bmatrix} [w_r]_{1\times 1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & [w_c]_{2\times 2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & [w_\omega]_{1\times 1} \end{bmatrix}. \tag{1.25}$$

The weighting matrix allows for additional control over the contribution of each momentum handle to the solution. Since the Jacobian matrix is not square, we use a pseudo-inverse method to compute the solution

$$\Delta\theta = J_w^\dagger \, g_{des}(\theta) \tag{1.26}$$

where $J_w^\dagger$ is the pseudo-inverse of J_w and is computed by

$$J_w^\dagger = {J_w}^T \left(J_w {J_w}^T\right)^{-1}. \tag{1.27}$$

Equation 1.26 provides a least square solution with minimal norm that remains robust as long as singular configurations are avoided. We use an iterative method because our characters typically have many degrees of freedom (20 to 25) and the target configuration is defined by a few control quantities which makes it impossible to apply a closed-form IK solver.

The desired physical quantities are set via an automatic system where the handle values are computed through processing the keyframes of each handle track in the interface panel. Equivalently, the desired parameters can be set by the animator through a set of interactive tools.

1.3.2 Natural motion with prioritized solver

Although solving Equation 1.26 ensures physical plausibility of our character motion, it does not guarantee for the character to maintain a natural posture. In our solver we include three types of constraints to achieve a natural motion:

- pinning constraint that control the velocity of links;
- desired joint angles;
- out of motion-range joint angle corrections.

Dealing with large collections of constraints is of interest in designing complex motions, but it can also become a problem when constraints are infeasible or conflict with one another. One solution is to give physics-based constraints higher priority than the postural constraints, as they are of greater importance and can not be violated.

We use a prioritized solver that takes advantage of the nullspace of $J_w^\dagger$ to allow for secondary

constraints. For any arbitrary vector γ we can expand Equation 1.26

$$\Delta\theta = J_w^{\dagger} g_{des}(\theta) + (\mathbf{I} - J_w^{\dagger} J_w)\gamma \tag{1.28}$$

where $(\mathbf{I} - J_w^{\dagger} J_w)$ performs a projection onto the nullspace of $J_w^{\dagger}$. By suitably choosing γ we can achieve secondary effects such as satisfying the postural (soft) constraints, while $J_w^{\dagger} g_{des}(\theta)$ provides us with minimum error solutions to the physics-based (hard) constraints [Buss (2004)]. We directly follow the work of Yamane and Nakamura (2003) for the computation of γ. Our work, however, differs from their approach as we treat the pose tracker as lower priority constraint and compute it in γ, whereas they treat it as a primary constraint.

The problem of having additional control quantities is that the resulting equations may not have exact solution due to conflict. Even though solving the IK system with a pseudoinverse yields a least square solution, infeasible solutions might be produced when the configuration is singular. Therefore singularity robust methods, such as the one proposed by Deo and Walker (1995) need to be applied.

A singularity robust method (SR), also known as damped pseudoinverse, uses a regularization parameter to relax the solution by letting the norm of the solution have some error

$$A^* = A^T (AA^T + k\mathbf{I})^{-1} \tag{1.29}$$

where A^* is the SR inverse of A, matrix $\mathbf{I}$ is the identity, and k is the damping parameter weighting between the error and the norm of the solution. For small values of k we get smaller errors, but we might as well encounter larger solutions around singular points. Using the SR inverse along with proper tuning of damping k helps us ease the singularity problem by allowing errors near singular points. We, however, want to use regularization only for the constraints that are secondary in terms of importance and use a typical pseudo-inverse method for the momentum related quantities. As a result, we use an undamped pseudo-inverse method for the hard constraints, and a damped pseudo-inverse technique for the soft constraints to achieve the best mixture in our prioritized solver. Algorithm 1-1 provides an overview of the steps involved in our IK solver.

1.3.3 Limited changes

Our system provides both the ability to adjust the character pose globally as well as making changes to a limited set of user specified joints. While making global edits helps fast sketching a pose, giving the animator the ability to apply limited changes to the body is also necessary. This is because an artist is typically interested in making subtle adjustments to a part of the character without affecting the whole body.

In order to isolate a group of joints, a local root R' is selected and the dragging of joint D determines the set of body limbs that are taken into account by the IK solver. Figure 1-2 shows how dragging a joint partitions the character into an active joint set (red nodes) and a passive set (blue nodes). Algorithm 1-2 shows how we mark all the joints as *included* or *excluded* depending on which subtree each joint belongs to.

Procedure 1-1 PHYSIK algorithm

while running **do** // Processing the keyframes
  $\theta \leftarrow$ GET CURRENT JOINTS STATE
  $C \leftarrow$ COMPUTE CM(θ)
  $I \leftarrow$ COMPUTE INERTIA(θ)
  $T \leftarrow$ GET CURRENT TIME
  $(Ks_1, Ks_2) \leftarrow$ GET POSE KEYFRAMES(T)
  $(Kp_1, Kp_2) \leftarrow$ GET PIN KEYFRAMES(T)
  $(Kc_1, Kc_2) \leftarrow$ GET CM KEYFRAMES(T)
  $(Ki_1, Ki_2) \leftarrow$ GET INERTIA KEYFRAMES(T)
  $\theta_{\text{des}} \leftarrow$ INTERPOLATE(Ks_1, Ks_2)
  $P_{\text{des}} \leftarrow$ INTERPOLATE(Kp_1, Kp_2)
  $C_{\text{des}} \leftarrow$ TRAJECTORY TEMPLATE(Kc_1, Kc_2)
  $I_{\text{des}} \leftarrow$ INTERPOLATE(Ki_1, Ki_2)
  while not converged **do** // IK solver
    $J_c \leftarrow$ CM JACOBIAN(θ, C) // Eq. 1.5
    $J_r \leftarrow$ ROTATIONAL INERTIA JACOBIAN(θ, I) // Eq. 1.14
    $J_{\text{hard}} \leftarrow$ COMBINE(J_c, J_r) // Eq. 1.24
    $g_{des}(\theta) \leftarrow$ SET DESIRED HANDLES$(C_{\text{des}}, I_{\text{des}})$ // Eq. 1.23
    $\Delta\theta_h \leftarrow$ SOLVE$(g_{des}(\theta), J_{\text{hard}})$ // Eq. 1.26
    $J_{\text{soft}} \leftarrow$ SOFT CONSTRAINT JACOBIAN$(\theta, P_{\text{des}}, \theta_{\text{des}})$
    $\Delta\theta_s \leftarrow$ SOLVE SOFT CONSTRAINTS$(J_{\text{soft}}, J_{\text{hard}})$
    $\Delta\theta \leftarrow \Delta\theta_h + \Delta\theta_s$ // Eq. 1.28
    $\theta \leftarrow$ UPDATE CHARACTER$(\Delta\theta)$
  end while
end while

When solving for the soft constraints, we adjust the corresponding weight of those that are *included* to 1 and set the rest to 0. At the core of our algorithm we find the path P_m by marching from joint D to R'. For any other joint we find the path P_i to R' and mark the joint *included* only if P_i and P_m share at least one joint other than R'. We do not allow for any translational update to the actual root when performing limited changes. Our method ensures no violation of the positional or physical constraints when a pinned joint is *excluded* and is dragged by D.

1.4 Handle trajectory templates and interface

Given the appropriate handles, such as the CM position and inertia, the key to our solution is to present the animator with a tool that allows them to intuitively edit these features of a motion while preserving physical plausibility. In this section, we describe a collection of tools and trajectory templates that provide this functionality.

Figure 1-3 shows part of the temporal controls interface presetned to the animator. Much like in any standard keyframing interface, there are different tracks for different quantities that have keyframes set, such as pose tracker, hand or foot constraints, as well as the CM position. The animator can

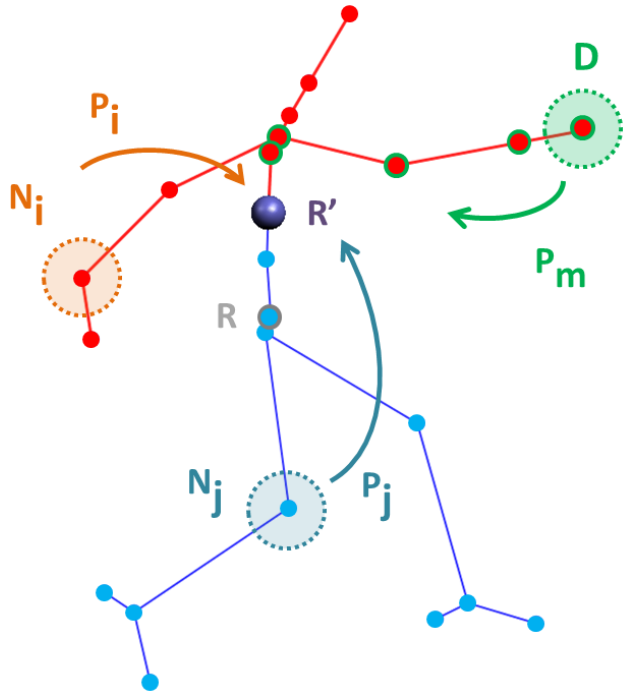

Figure 1-2: *Dragging the joint D separates the body into active (red) and passive (blue) regions. Main path P_m to the local root R' shares at least one node with path P_i, which makes the joint N_i a member of the active set. Path P_j does not share any joints with P_m so its corresponding sample joint N_j is marked excluded during the IK solve.*

insert, delete, copy and paste, and edit keyframes, as well as dragging them to different times. Between a pair of keyframes, we allow the user to adjust different properties, such as setting the interpolation function in the pose track, the positional constraint gains in the pin track, and the template trajectories in the CM track. Figure 1-4 shows a six degrees of freedom space navigator, an optional sensory device to help the animator manipulate the physics-based handles. It can be used to move the CM position, change the inertia or rotate the character by simple actions like push, pull, twist or tilt, while the other hand uses existing mouse to select, drag or edit the handles in the interface.

In order to obtain the character configuration for the current time step, the system first uses the timing pane to select the active pair of keyframes in each track, then computes the desired quantities through processing the selected keyframes, and finally solves for the joint angle updates by inverse kinematics. The animator is provided with optional inclusion or exclusion of keyframes in the pin and the CM tracks. This is a direct feature of separating physics-based quantities and postural constraints in Equation 1.28. Figure 1-5 shows a snapshot of the spatial controls available to the animator during the landing sequence of a jump. The CM position at the keyframes can be dragged, as can be the velocity. Pinning constraints are also an important part of designing the motion. In the figure, note how feet are naturally dragged towards the ground while preparing for landing. The desired touch down points are shown in red on the grass. While the character is still in the air, the pins are set with low gains that increase over time in preparation for the landing phase. The velocity of the CM at the time of landing is likewise used as the initial velocity of the CM Hermite

Procedure 1-2 Limited changes

```
while Local root is selected do
  R ← GET THE ACTUAL ROOT
  R' ← GET THE LOCAL ROOT
  D ← GET DRAGGED NODE(nodes)
  while D is valid do
    MARK ALL NODES EXCLUDED(nodes)
    P_m ← FIND PATH TO LOCAL ROOT(R', D)
    for all nodes do
      N_i ← GET CURRENT NODE
      P_i ← FIND PATH TO R'(N_i, nodes, R')
      if P_i and P_m share any node except R' then
        MARK INCLUDED(N_i)
      end if
    end for
    N_s ← GET A SAMPLE INCLUDED NODE(nodes)
    P_L ← FIND PATH TO R'(N_s, nodes, R')
    P_R ← FIND PATH TO ACTUAL ROOT(N_s, nodes, R)
    if P_L ⊆ P_R then
      MARK INCLUDED(R')
    end if
  end while
end while
```

curve in the landing phase, ensuring that there is a C1 continuous smooth transition between the projectial path and the landing motion. Figure 1-6 demonstrates interactive pin gain controllers available to the animator when authoring a landing scenario. While adjusting the master pin gain controller affects all pinned joints, there also exist individual adjustable spline curves for each joint as an extra means of further fine tuning the dragging effect of each limb. The final gain value of each joint is then the product of the master and the individual gain values.

We have investigated a few different templates for setting physically plausible character motion between CM keyframes.

Hermite. Cubic curves allow position and velocity to be matched connecting the segments between keyframes. In particular, we can ensure smooth trajectories for the CM leading into and exiting the other template trajectories. In order to generate a trajectory for a Hermite interval, CM positions are sampled along the path connecting two Hermite keyframes based on the time duration of the path, and the CM positions and velocities at each end. Although the cubic curves are computed for each Hermite interval separately, the resulting spline when connecting multiple segments will be C1 continuous. Figure 1-7 illustrates an example of generating two Hermite segments for two CM keyframes, one from C_0 to C_1, and one from C_1 to C_0. While our system uses velocity handles to ensure the smoothness of the connected segments, it also allows for creating motions with discontinuity in the trajectory derivative at a desired point, such as C_0, where the

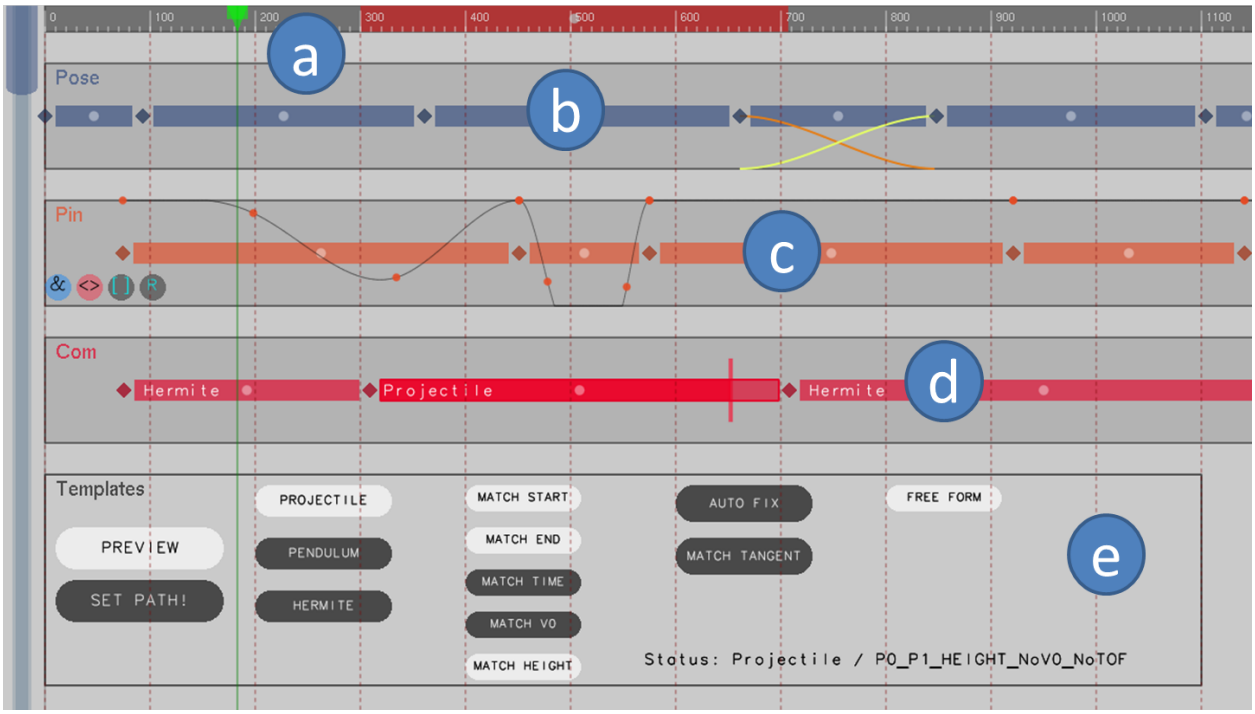

Figure 1-3: *Screen capture of the workspace used to set keyframes and template trajectories. (a) Timing pane with a selected repeat area shown in red and a time scrubber shown in green. (b) Pose keyframing track where the type of interpolation is also set (note the ease-in-ease-out yellow and orange curves). (c) Pin keyframes with a super imposed cubic spline to control pin gains. (d) CM keyframes with a hint for physically plausible timing requirement. (e) Trajectory template control panel.*

entering velocity is different from the exiting velocity. Although authoring motions that require discontinuities in the CM trajectory does not happen too often, inclusion of a feature that does not bound the CM trajectory to be only continuous can help certain motions become more realistic. For instance, we expect a certain level of bounciness when animating a character with non-visceral body limbs, like a skeleton of a robot, in scenarios that involve large contact forces, like falling flat on the ground or being hit by a car.

Projectile. This template has the constraint that the linear momentum of the character is only affected by gravity, and thus the center of mass will undergo a constant vertical acceleration of $-9.8\ \mathrm{ms}^{-2}$, while the horizontal momentum is conserved. The animator can adjust the arc of the jump through a variety of controls such as the maximum height, initial velocity, initial position, end position, and time of flight. Because these parameters are coupled, our tool automatically adjusts the parameters left unconstrained by the animator in order to satisfy the constraints of the template trajectory. In particular, we present three projectile templates, as shown in Table 1.1, each with a specific purpose that makes them intuitive to use by the animator. We label these templates as (I) , (II) and (III).

Projectile (I) presents a straightforward template where the animator will only need to set the start and end CM positions and specify the time of flight, where the height of the arc can be adjusted

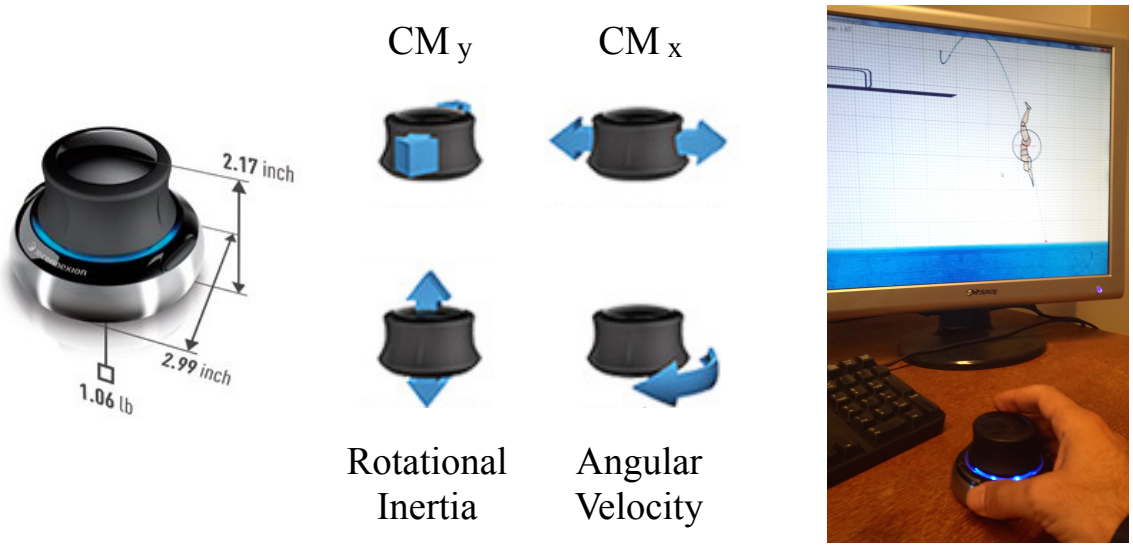

Figure 1-4: *Space navigator. Left: device specifications. Middle: controlling physics-based handles. Right: the device in action.*

through changing the time interval of the keyframes. Note that this template does not use any velocity handles but updates its neighbours velocity handles to ensure smoothness.

Projectile (II) does not require a landing position and instead uses the take-off velocity handle to control the jump arc. In this case we allow the template to automatically compute the landing position and update the corresponding CM keyframe with the new value, $C_e{}'$. The free landing parameter helps the animator to explore different jump scenarios in the sketching phase only by regulating the initial velocity handle without being too much concerned with the landing position.

Projectile (III) uses an auxiliary interactive tool to set the desired maximum height. This is specially useful when we want to author jump motions where take-off and landing, as well as the height of the motion, are more important than the time of flight. For example, performing a slam dunk by a basketball player requires reaching a certain desired height while take-off and landing positions should remain on the floor and inside the playground. Because the time of flight might not match that of the keyframes we provide visual hints both on the ballistic path and in the keyframing panel to help the animator with the process, as shown in Figures 1-8 and 1-9. We also present an "auto-fix" feature to automatically compute a new time of flight $t_e{}'$ that satisfies the jump height and positional constraints. Should the animator decides to use this feature, we move the end CM keyframe such that the new time interval matches $t_e{}'$, as shown in Figure 1-8.

Pendulum. The motion of the CM follows the motion of a pendulum, allowing animation of a character swinging between branches, on monkey bars or spiderman type of swing between buildings. This template assumes a frictionless pivot that results in an undamped motion trajectory of the CM. The only acting forces on the pendulum are the gravitational force and the force that keeps the mass point on the constrained path. The length, initial velocity and pivot position of the pen-

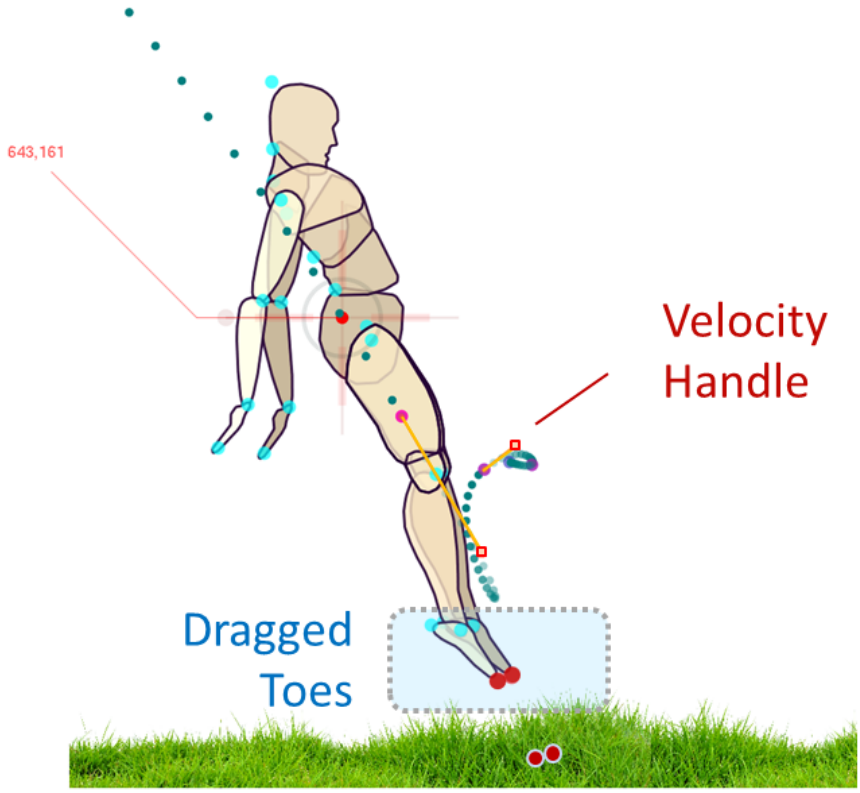

Figure 1-5: *Screen capture of the interface showing handles available for controlling the landing motion of a jump.*

dulum can be adjusted and keyframes can be set to transition at arbitrary times. Just like projectile (II) we do not use the end CM position in generating the pendulum path, rather we update the end keyframe with the position of the CM at the end of the time interval, as shown in Figure 1-10 as C_3'. Using the same template we can also generate path for an inverted pendulum by adjusting the initial velocity handle. An inverted pendulum allows animating a variety of interesting character motions such as walking locomotion or gymnastic front and back aerials. This is inspired by simplified models used to understand animal locomotion, and to control physics based character locomotion.

Table 1.1 summarizes the control variables available to the animator when using each feature trajectory template. We always update the neighbour CM trajectories when editing the path of a template in order to maintain the smoothness of the CM trajectory segments. For instance, in Figure 1-9 we see how the adjacent Hermite templates are updated as we edit the projectile path. Note that we can only update the velocity handles if they are part of the adjustable variables of the trajectory template (V_s and V_e). This means, for example, Hermite curves are always updated through both the initial and end velocity handles, while a pendulum template can only be updated through its initial velocity handle.

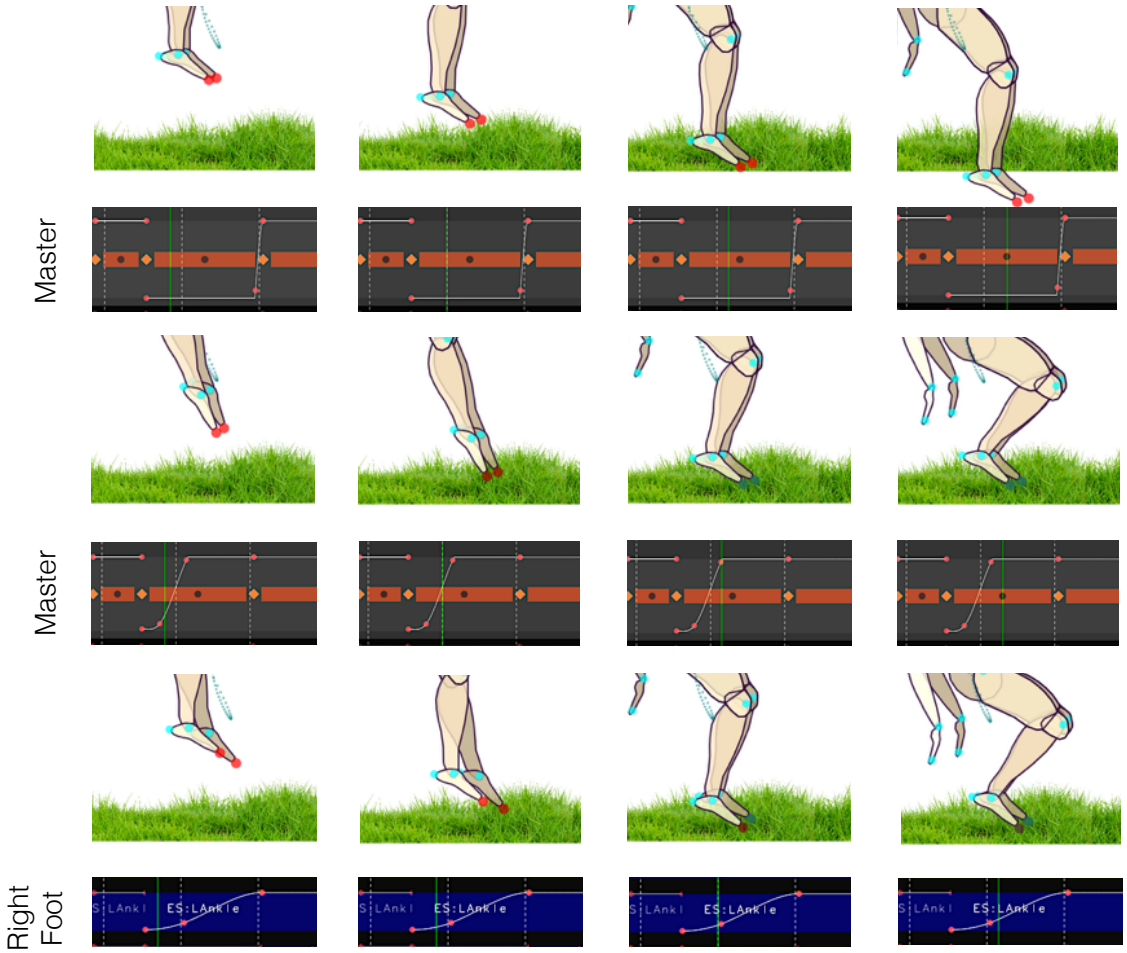

Figure 1-6: *Using overall (master) and individual pin gain controllers to achieve different landing styles. Top: the master gain controller is set to zero for the duration of landing, leading to feet not meeting the desired landing site. Middle: Following the temporal gain set by the cubic spline, feet are stretched to prepare for landing. Note how the pin colour changes from red (maximum tension) to green (completely meeting the desired landing position) as the character approaches the ground. Bottom: Adjusting the right foot pinning gain such that it becomes more relaxed than the left foot for the same motion in the middle row. Note that the right foot is the light brown one.*

1.5 Results and discussion

We present results for a variety of 2D and 3D examples in our framework, which are best seen in the supplementary video [1]. While our system was originally built as a 2D platform, we have also taken a number of steps towards extending it to a 3D system. There remain, however, certain challenges to be addressed in having a mature 3D platform that will be discussed in Chapter 6 as future work. What follows in the rest of this section mostly focuses on planar characters, followed by demonstrations of preliminary results for implementation and testing of our system in 3D.

[1] `http://www.shahinrabbani.ca/publications.html`

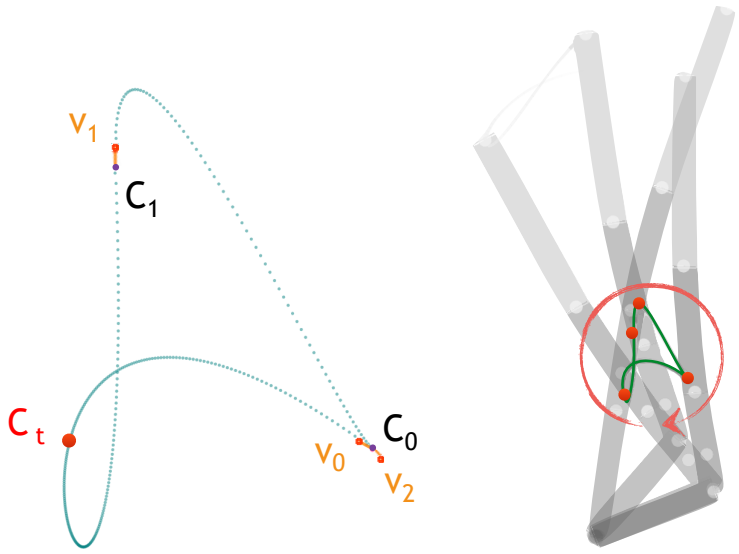

Figure 1-7: *Example Hermite template trajectory. Left: generating a loop trajectory for two CM keyframes with a smooth transition at C_1 and a non-smooth transition at C_0. The CM position at the current time is shown by C_t, while V_0, V_1 and V_2 are the desired exiting, transition and entering velocities of the keyframed CM positions respectively. Right: cyclic motion of a multibody robotic arm by constraining its CM to follow the template trajectory. Note that one end of the model is pinned in order to encourage a time varying change of configuration in the arm motion.*

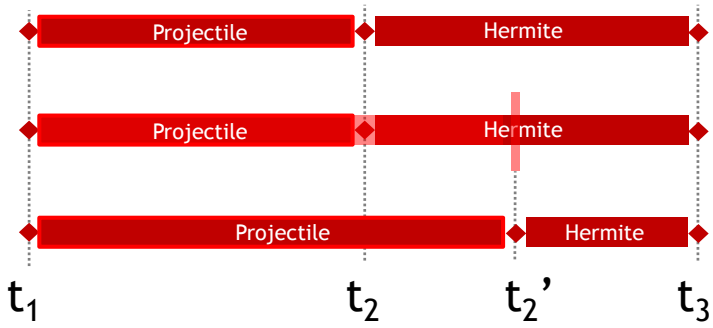

Figure 1-8: *Projectile hint and auto-fix in the CM keyframing track. Top: Initial setup of the keyframes to be edited. Middle: Increasing the desired maximum height of the jump violates the timing of the end CM keyframe. Bottom: Corrected time of flight t_2'.*

1.5.1 2D platform

Our keyframing system allows for rapid authoring of motions by providing visual hints and assistance to the animator whenever a physical constraint is violated. We also show that our method

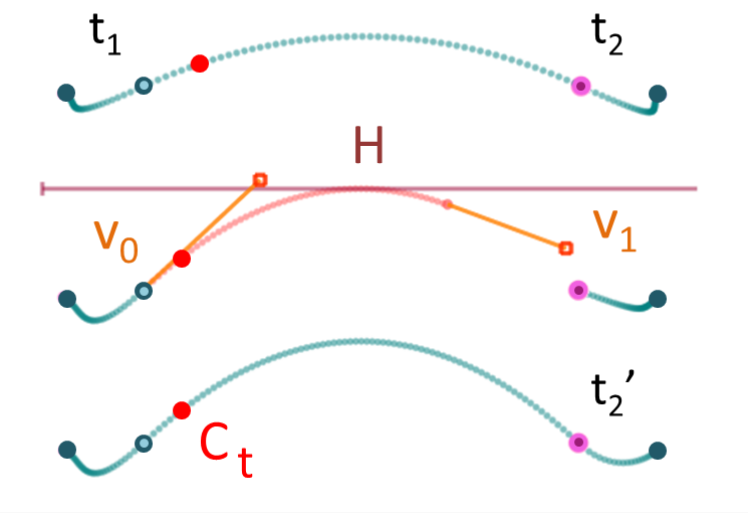

Figure 1-9: *Projectile hint and auto-fix display related to the example shown in Figure 1-8 with the same order of steps. H is the desired maximum height, V_0 and V_1 are the initial and end velocity handles and C_t is the CM position at the current time. The rest of variables are the same as Figure 1-8. Note how the Hermite trajectories before and after the Projectile motion are automatically updated in the third row.*

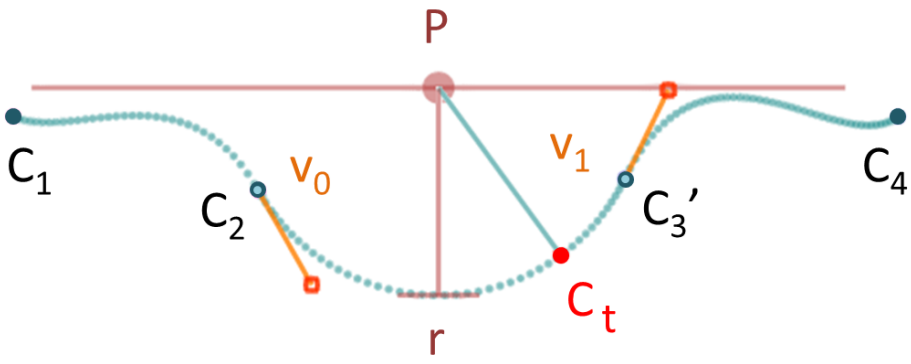

Figure 1-10: *Generating a pendulum trajectory for a pair of CM positions given by C_2 and C_3. Initial velocity handle V_0, the pivot position P and the time interval between C_2 and C_3 are used to generate the trajectory. While C_3' has the same timing as C_3 in the keyframing track, its position is automatically updated by the system to remain on the constrained trajectory. Hermite curves are also set for the keyframe pairs C_1 and C_2 as well as C_3' and C_4. Red dot C_t is the CM position at time t. Note that the end velocity V_1 is used to update the cubic curve on the right-hand side.*

does not make any assumption regarding the morphology of the character. All computations were run on a dual-core 2.4 GHz machine with 12 GB RAM. Each IK step with both hard and soft constraints takes about 250 μs. This helped us achieve real time preview display and user interactions

Template	C_s	C_e	V_s	V_e	ΔT	H/P	C_e'	t_e'
Hermite	x	x	x	x	x			
Projectile I	x	x			x			
Projectile II	x		x		x		x	
Projectile III	x	x			x	x		x
Pendulum	x		x		x	x	x	

Table 1.1: *Template control variables. C_s and C_e are the start and end CM positions, V_s and V_e are the start and end velocities, ΔT is the time duration of the trajectory, H/P means either use of desired height (H) or pivot position (P), C_e' is the computed end CM position and t_e' is the corrected end time.*

Joint	Body	Mass (kg)	Moment ($kg.m^2$)	θ_{min}	θ_{max}
Head	Head	4.2	206	$-\pi/6$	$\pi/6$
Neck	Neck	1.1	18	$-\pi/4$	$\pi/6$
Shoulder	Arm	2	141	$-\pi$	π
Elbow	Forearm	1.4	90	0	$4\pi/5$
Wrist	Hand	0.5	13	$-\pi/2$	$\pi/2$
Chest	Chest	24.9	5000	$-\pi/6$	$\pi/10$
Upper abdomen	Upper abdomen	1.2	140	$-\pi/12$	$\pi/10$
Lower abdomen	Lower abdomen	1.2	140	$-\pi/12$	$\pi/8$
Root	Pelvis	11.8	1116	$-\pi$	π
Hip	Thigh	9.8	1652	$-2\pi/3$	$\pi/3$
Knee	Calf	3.8	606	$-2\pi/3$	0
Ankle	Foot	1	8	$-\pi/8$	$\pi/10$

Table 1.2: *Humanoid physical properties and joint limits.*

in the keyframing interface, which is a crucial factor in animation creation process.

Models. Figure 1-11 demonstrates a humanoid as well as a 4-link abstract model that we use in synthesizing the results for the 2D system. The humanoid character has 20 degrees of freedom (including two dimensional root translation) and has total mass of 80.4 kg, where the size and mass distributions come from a study by Nasa OH (1988). We also use an abstract 4-link arm with top-bottom asymmetric morphology and inertia to highlight the capabilities of our system in generating interesting behaviours for non-human characters. This model has 6 degrees of freedom with a total mass of 36 kg. Tables 1.2 and 1.3 provide the physical properties as well as the joint limits of these characters.

Cyclic motions by physics-based handles. By arbitrarily combining physics-based handles, we can achieve interesting motions even in the absence of interpolated pose keyframes. This is best seen in Figure 1-12 where a mixture of such handles are put at work to create life-like cyclic motions. We use a 4-link abstract model to put emphasis on our system capabilities of generating

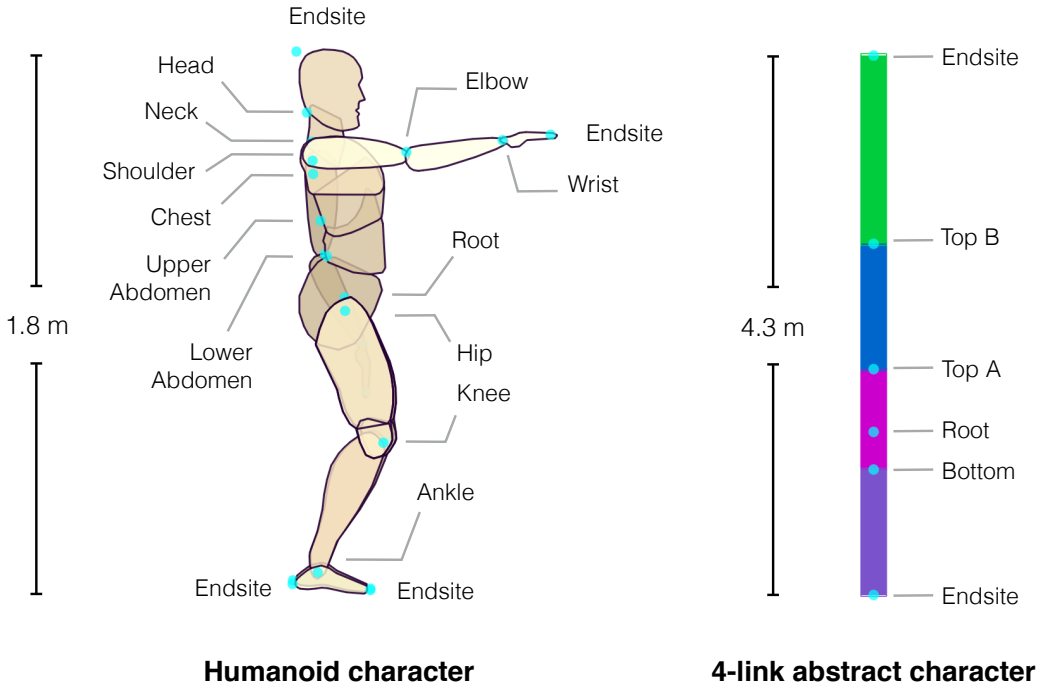

Figure 1-11: *Planar character models. Left: humanoid character based on the morphology and the proportions of an average human body. Right: 4-link abstract robotic arm with top-bottom asymmetrical body segments. Note that the 4-link model is much larger (4.3 m) than the humanoid character (1.8 m).*

Joint & body	Mass (kg)	Moment ($kg.m^2$)	θ_{min}	θ_{max}
Root	11.8	1116	$-\pi$	π
Bottom	9.8	1652	$-3\pi/2$	$\pi/3$
Top A	4.6	206	$\pi/16$	$\pi/2$
Top B	9.8	1652	$\pi/16$	$\pi/2$

Table 1.3: *4-link physical properties and joint limits.*

motions that are familiar and natural, despite the character's morphology. The figure contains four different examples of such motions. The first row applies a pendulum template trajectory to the CM while one end of the arm is pinned to create a swing type motion. The second row demonstrates a motion that resembles what we see in a trotting dog. The inertia handle undergoes a sinusoidal change while the CM handle is not active and one joint is pinned. The third row shows a piston type cyclic motion, which is created by pinning one end of the arm and setting the angular velocity of the model to a constant value. The CM and the inertia handles are not used in this example. Finally, the last row combines an inertia handle with sinusoidal variations with a projectile template to generate a "happy" bouncing arm. More examples are provided in the supplementary video.

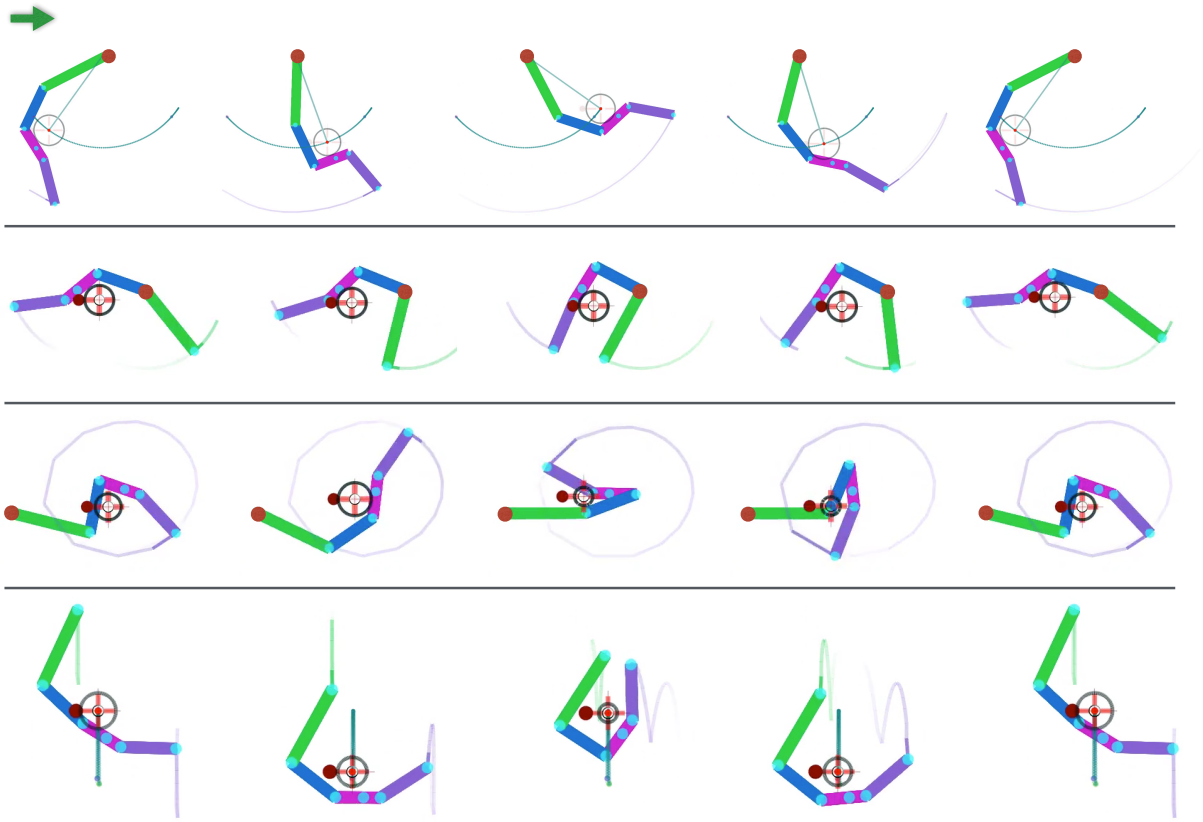

Figure 1-12: *Cyclic motions by physics-based handles. First row: swing motion using a pendulum template. Second row: trotting style motion using sinusoidal changes of the inertia handle. Third row: piston type motion by pinning one end of the arm and applying a constant angular velocity. Last row: "happy" bouncing character using a projectile template and varying inertia.*

Conservation of angular momentum. Figure 1-13 demonstrates how the angular momentum can be constrained to remain conserved while the characters undergo rotations during the flight phase. Setting the desired angular momentum and the rotational inertia quantities, the IK solver adjusts the root rotation such that the overall body angular velocity becomes consistent with the constant angular momentum for the duration of the motion. To demonstrate this, we use a time varying inertia quantity based on a sinusoidal function, as illustrated in the figure. While the plot shows five full periods of the sinusoidal inertia function, the motion stills in the top part of the figure only show the 4-link arm following half of the sinusoidal period. The complete animation is best seen in the supplementary video. Note from the coloured tracker lines how the rotational speed decreases as the character spreads its body from left to right. The same experiment can be performed on a humanoid character with many degrees of freedom to achieve the same motion style. The only adjustment required for this adaptation is to rescale the amplitude of the inertia sinusoid to match the maximum and minimum feasible inertia values of the humanoid character.

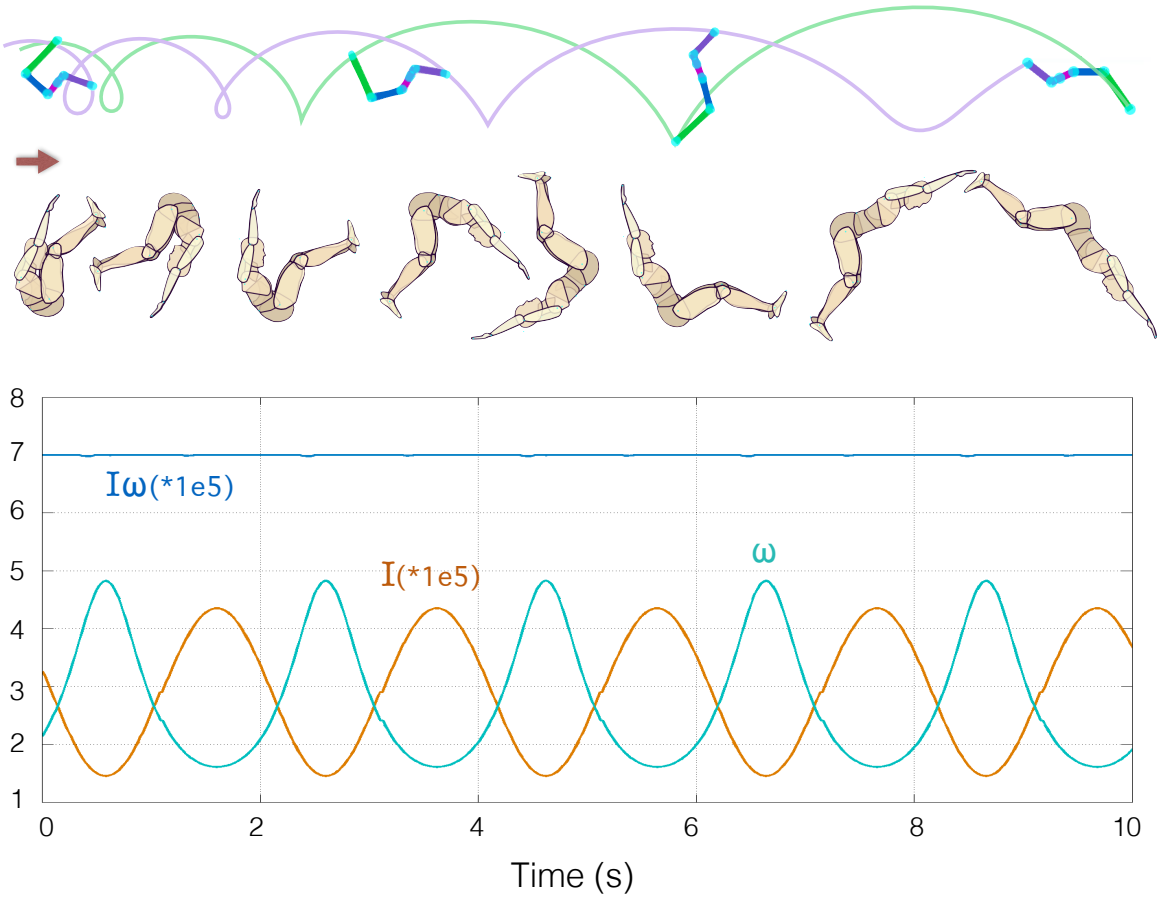

Figure 1-13: *Conservation of angular momentum. Top: rotational inertia values of the 4-link and the humanoid characters follow a time varying sinusoidal function while their angular velocities are adjusted to match a desired constant angular momentum set by the animator. Bottom: the product of the rotational and the angular velocity solution remains constant for the duration of the motion.*

Limited changes. Figure 1-14 shows reproducing results similar to those demonstrated by Yamane and Nakamura (2003), as well as how our limited change algorithm provides more control over editing body posture. In this figure, we compare postural changes of the character with and without a user selected local root, where edits are only applied to a user selected region. The limbs coloured in green are temporarily excluded from the edit process. Our algorithm ensures dragging an active limb results in a new posture that also meets the positional constraints of the passive pins. This can be seen from parts (d) and (f) in the figure where the pinned feet remain in their positions while the the right hand is dragged.

Dive motion. Figure 1-15 shows a fast way of creating the poses that are common in a dive motion simply by regulating the inertia of the character. In this figure we maximize the body inertia to achieve full stretch in the beginning and at the end, and decrease it at the top of the ballistic curve. We use the inertia handle along with the projectile template to quickly generate

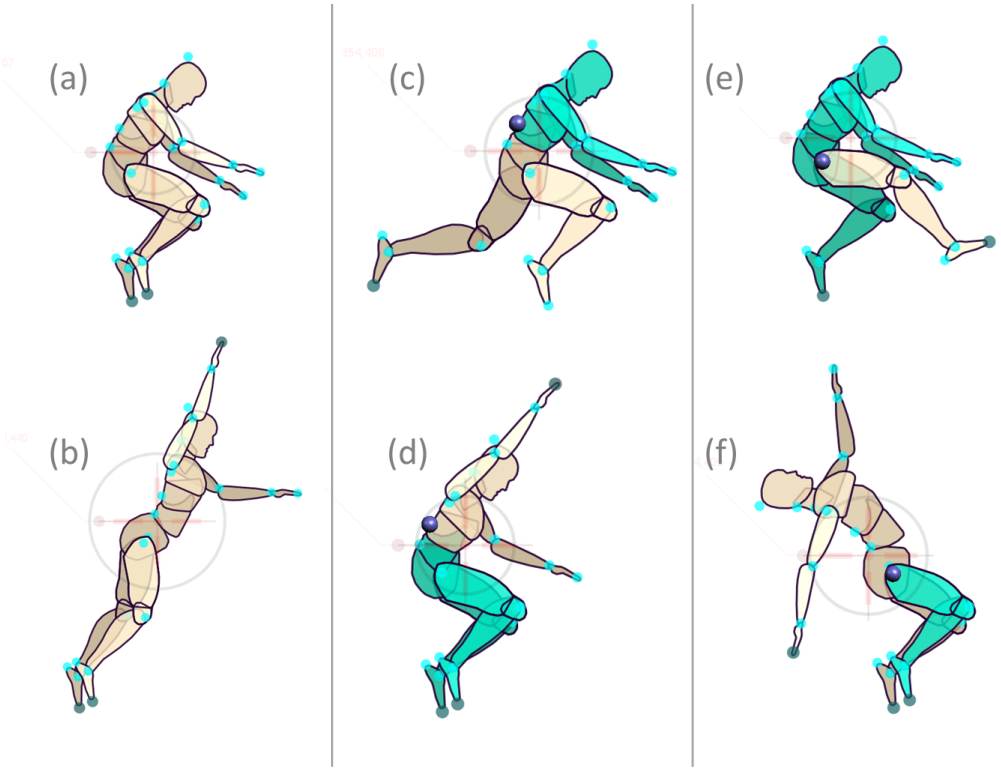

Figure 1-14: *Limited changes. (a) Default pose with feet pinned. (b) Dragging right hand results in full body change. (c) Using local root to only adjust the lower part of the body. (d) Using the same local root to adjust the upper part. (e) Isolating the right leg through a local root at the right hip. (f) Dragging the right hand changes the body posture but has no effect on the excluded limbs. Note that the positional constraint of the right foot is also satisfied.*

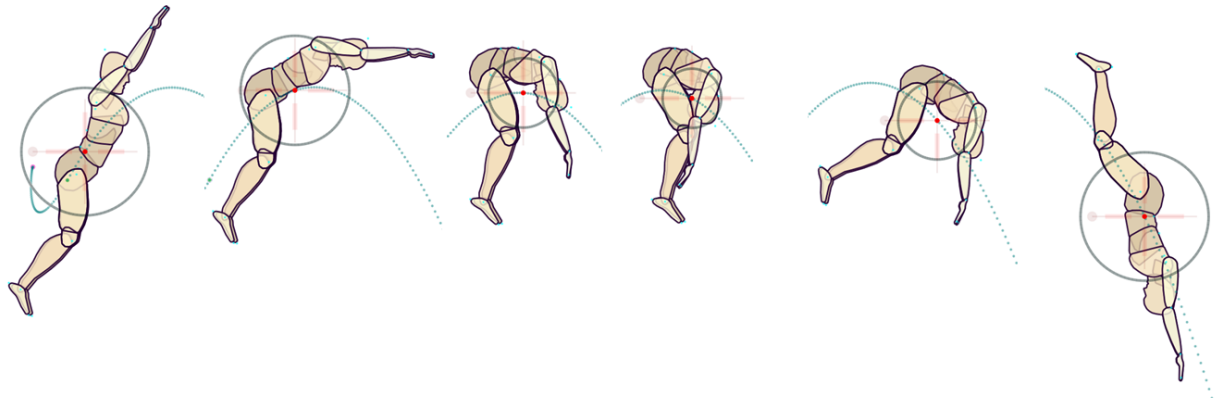

Figure 1-15: *Dive motion using the projectile template and interpolation of rotational inertia. The inertia handle is shown as a grey circle. Note the change of inertia from left to right.*

results shown in this figure.

Work flow. In the video we present our work flow and how an artist is provided with visual hints to correct the physics of a jump motion. We also show an easy autofix button to automatically correct the timing of the keyframes when, for example, the timing difference between two keyframes does not allow the character reach its target landing position.

Face hugger. In addition to the humanoid and the 4-link models, we created an alien character with many body limbs to specifically highlight how the inertia handle can facilitate editing the pose in such cases. This example also shows how the IK handles are not dependant on the morphology of the character. Figure 1-16 illustrates the results of this example. While creating a pose could be challenging due to many joints of the face hugger, we show changing the inertia can provide a fast way of moving the limbs towards a desired pose. Symmetrical postures are achieved by merely using the inertia handle. We also demonstrate that an asymmetrical predator pose emerges naturally by pinning two legs on the ground and moving the tail sideways while lowering the inertia quantity.

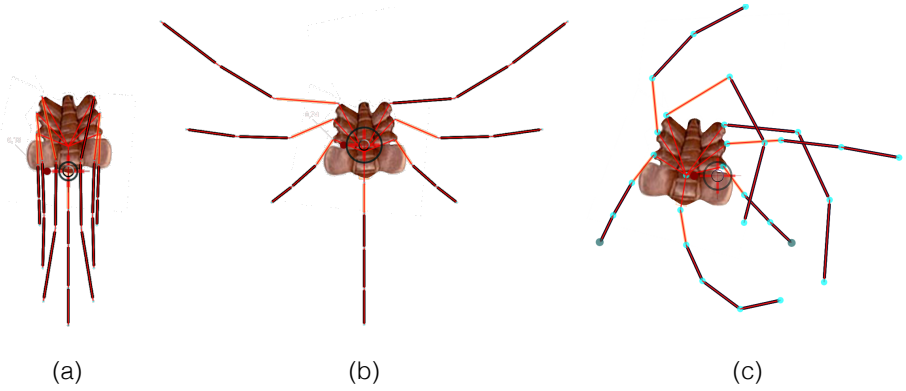

Figure 1-16: *Animating the face hugger by only using the inertia handle to change the legs configuration. Increasing the inertia results in a natural transition from (a) to (b). Note that the long tail in the middle is pinned to enforce symmetry. Moving the tail sideways along with decreasing the character inertia leads to a natural predator pose shown in (c).*

Performance. Figure 1-17 shows an example convergence plot for a typical case of editing the character with varying number of pins. For the sake of performance test we set the desired pose of the character to a fixed rest pose before each call to the IK solver. Note that while this could make it more challenging for the solver to converge, in practice we typically set the desired pose from the solution at the previous step of animation to get faster convergence rates. In our experiments we found that iteration numbers between 30 to 50 generally produce convincing motions while allowing for real time interactions with the character.

Monkey bar. As shown in Figure 1-1 as well as Figure 1-18 and in the supplementary video, all three types of trajectory templates are put together to create an animation where the character performs sportive actions in a highly dynamics manner. Successive swings consist of seven pendulum templates that are connected by Hermite cubic curves to achieve smooth CM trajectories in

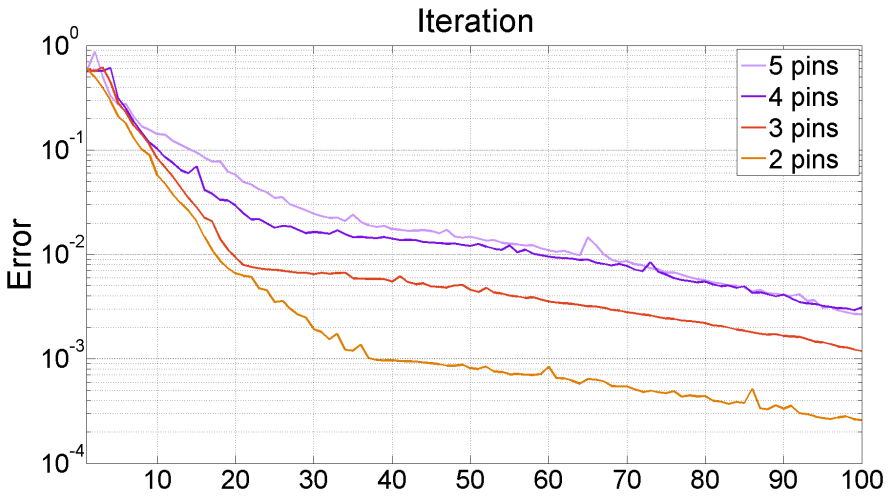

Figure 1-17: *Example convergence plot of an IK solve with different pin numbers. All soft and hard constraints are included in this example. The error is computed by* $||A^{-1}(b - Ax_0)||$ *for the solution* x_0 *to the system* $Ax = b$.

transitions. A projectile template is used for a plausible landing motion and a Hermite template steers the character to its rest pose. Pin gains are also scheduled to achieve natural arms and feet motion in making and breaking contacts with the monkey bar, as well as in landing.

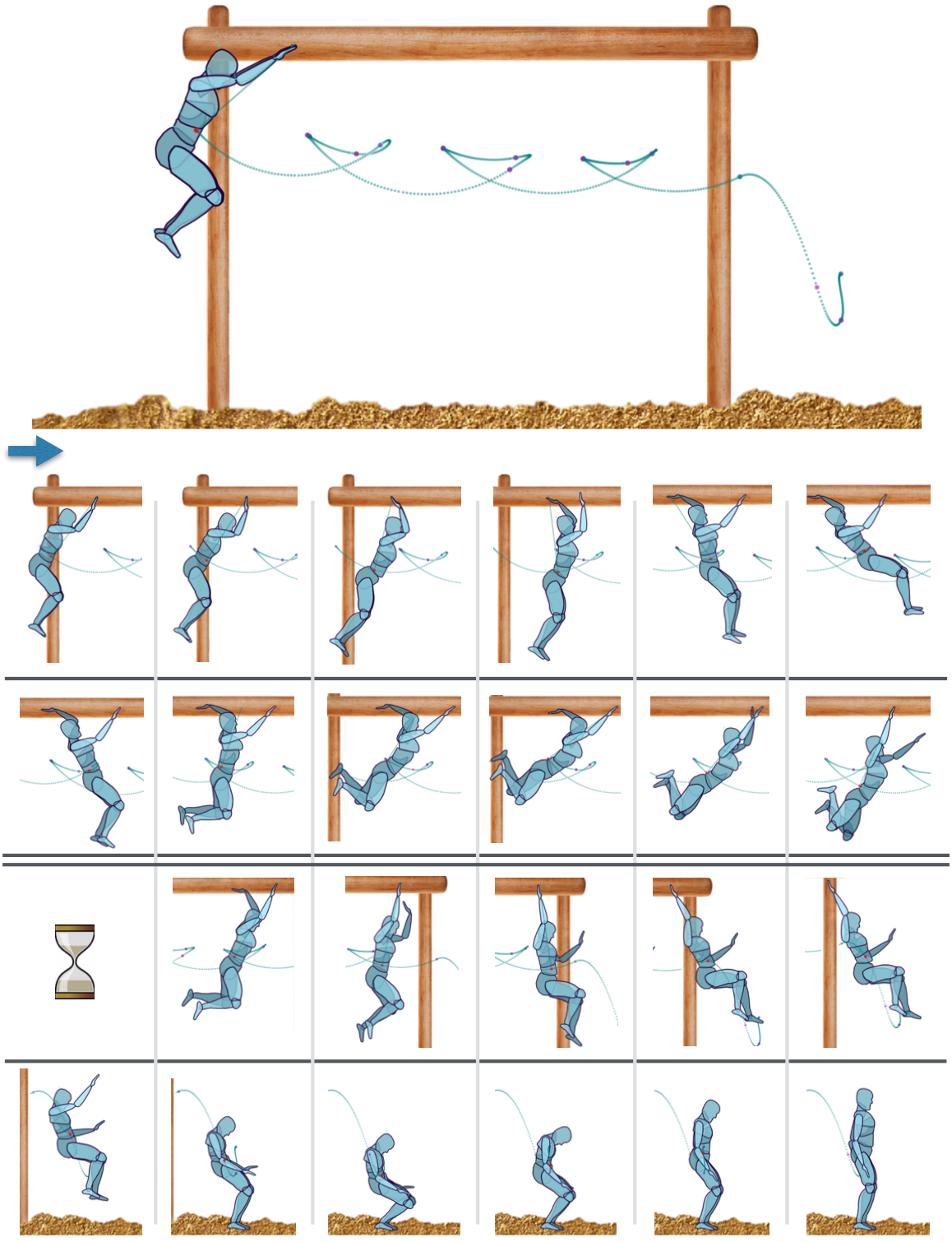

Figure 1-18: *Performing consecutive swing motions on a monkey bar using a pendulum template followed by a projectile template for landing. We dropped the long sequence in between to only show the start and the end of the animation.*

Jump motion. Figures 1-19 and 1-20 (also seen in the supplementary video) demonstrate how a jump motion is authored using the CM handle as well as the projectile and the Hermite trajectory templates. Figure 1-19 shows an example of changing the pose of the character without any necessity to regulate the physics-based handles. In Figure 1-20, we present a long jump example that well demonstrates the application of IK handle in our keyframing system. The character center of mass follows a projectile motion created by constraining a desired height, take off and landing positions. Not only Hermite template handles at take off and landing help creating a smooth curve for the center of mass, but also based on the created ballistic path they are automatically updated to match velocity when interpolating keyframes. Plausible take off and landing feet postures are achieved through pinning the toes and heels to the ground and regulate their gain using the interface.

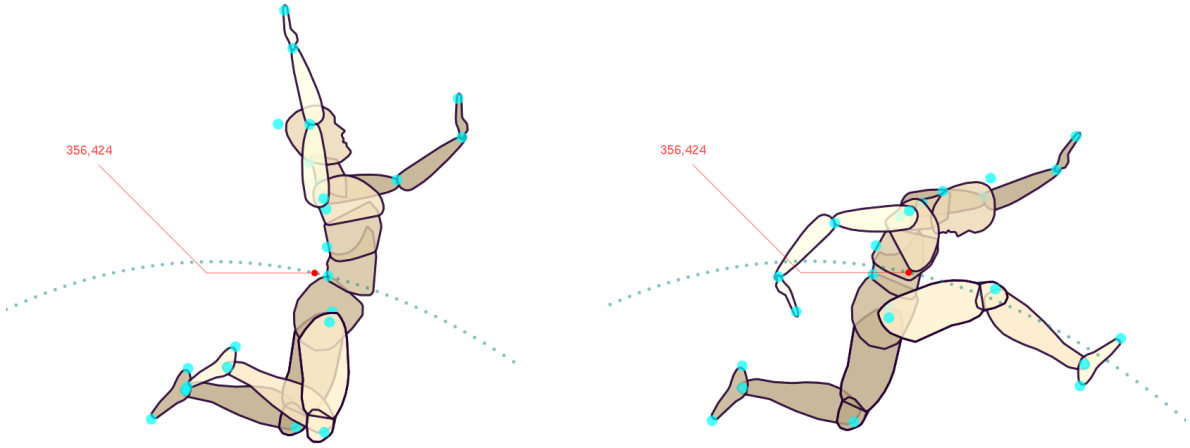

Figure 1-19: *Editing a jumping character. Left: Initial pose. Right: New pose. Note in both cases the center of mass position on the ballistic curve remains fixed.*

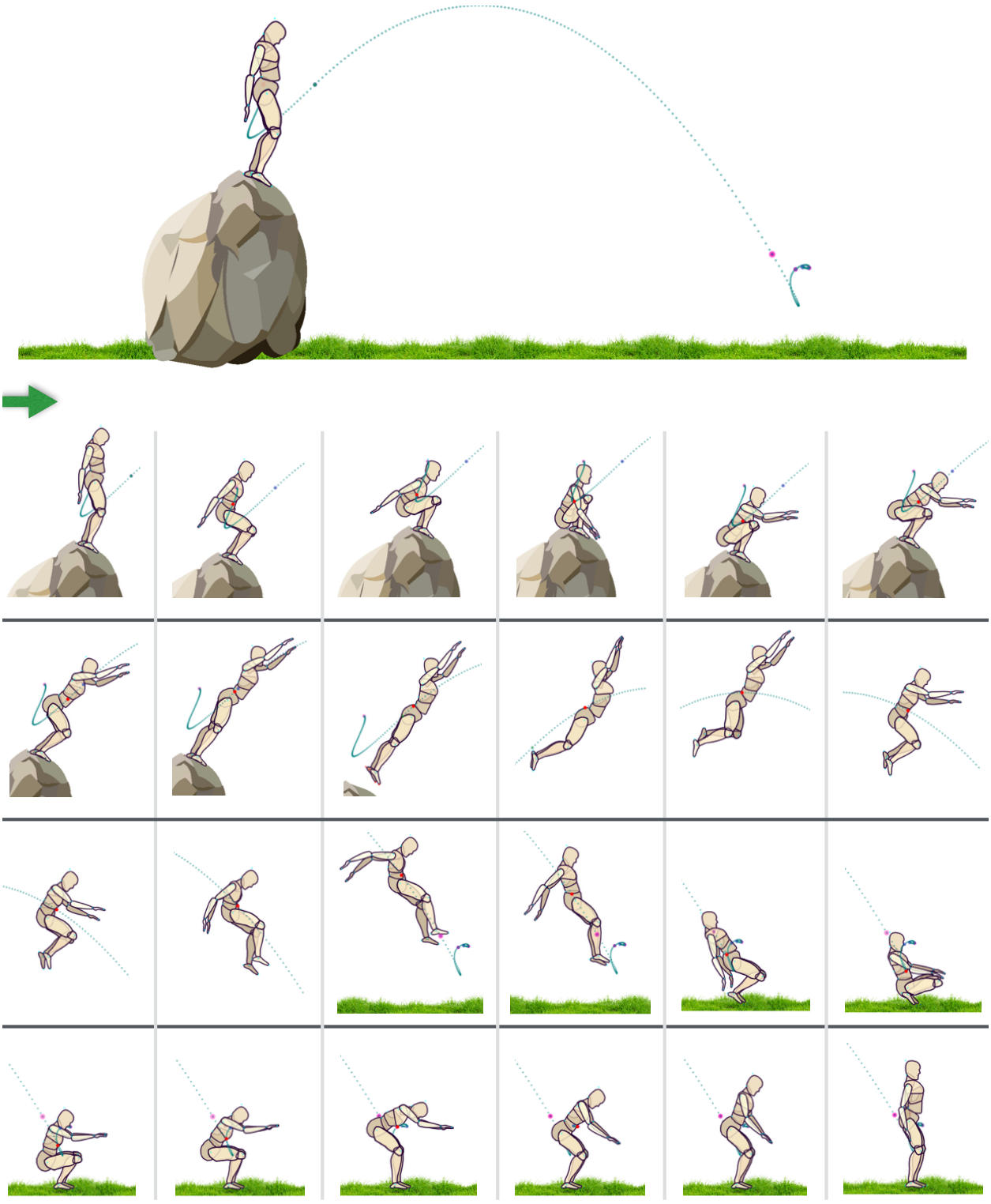

Figure 1-20: *Long jump using projectile and Hermite templates.*

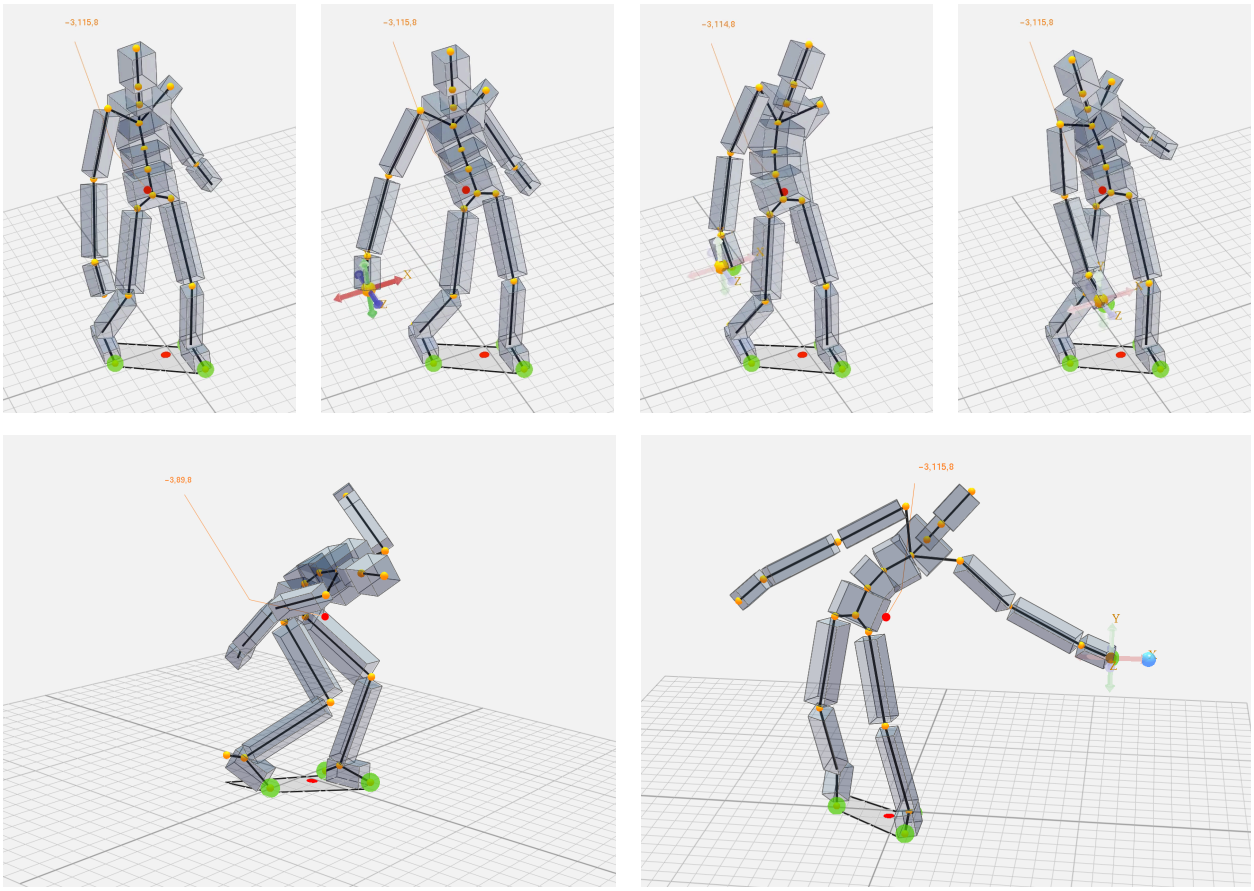

Figure 1-21: *Balance aware postures. Top: motion sequence where the CM of the character stays within the foot support area while the right hand is dragged back and forth. Bottom: vertically relaxed CM handle allows for more flexibility in editing the posture without allowing the CM to leave the foot support.*

1.5.2 Preliminary results for the 3D platform

Extending our keyframing system from 2D to 3D was a straightforward process for the most part, while the remaining challenges are discussed as future work in Chapter 6. Some necessary pieces for a 3D version were already available in our work. The keyframing interface is an abstract platform that does not make any assumptions about the dimensionality of the character, making it easy to adapt to a 3D framework. Also, the derivations of center of mass and inertia handles are the same in 3D. The soft constraints related to natural posture creation are also based on a 3D system inspired by work proposed by Yamane and Nakamura (2003). Furthermore, the re-rooting algorithm discussed in Section 1.3 and the physics-based template trajectories introduced in Section 1.4 are likewise independent of the dimensionality of the character. The IK solver average run time slightly increases in 3D (370 μs per step) due to the added degrees of freedom, but it well remains within interactive frame rates. What follows are examples that show how physics-based handles and template trajectories generalize to a full 3D workflow.

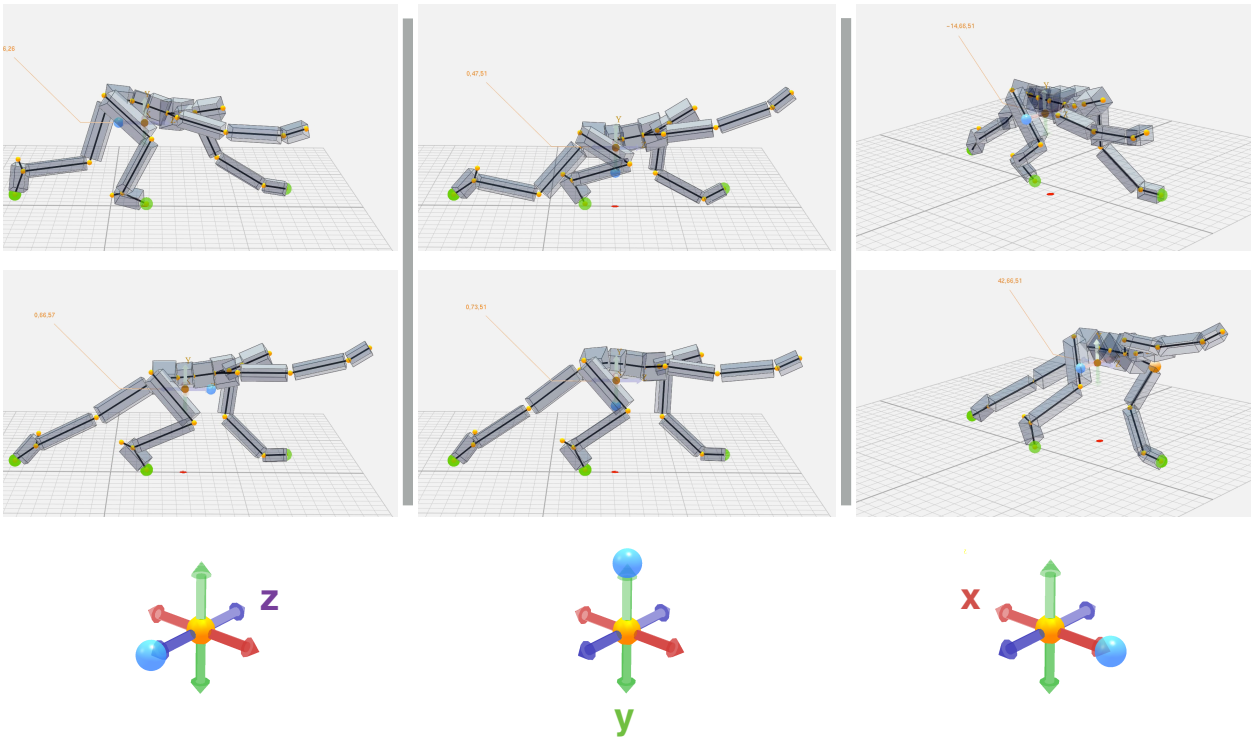

Figure 1-22: *Adjusting 3D CM handle along each axis provides a high level tool to easily edit the character posture. Note the projection of the CM on the ground as a visual reference of how the CM location changes in each case.*

Balance aware postures. We combine the postural editing process with balance considerations in the examples shown in Figure 1-21 (also seen in the supplementary video). The top row demonstrates the effect of dragging the right hand of the character while its CM is bound to stay within the foot support polygon. Such adjustments directly result in balance aware natural postures as shown. Our system also allows for vertically relaxed CM handle, meaning the CM position is free to move vertically but is not allowed to leave the support area. The video stills on the bottom row of Figure 1-21 present two of such cases. In the left part the character is pushed towards a crouch position by dragging down its chest, and on the right it spreads its body in an attempt to reach a target. Note that in all depicted examples the projection of the CM onto the support area stays at its desired position.

CM handle adjustment. Instead of constraining the CM to be in a fixed position or follow a template trajectory, our system enables the artist to intuitively apply full body changes through directly regulating the CM handle, as shown in Figure 1-22. This is particularly useful when multiple parts of the body are pinned and we would like to have a means of globally adjusting the virtual character posture. For instance, consider a sprint runner starter pose. While hands and feet should remain in a suitable location on the ground, the runner can lift up or lower its CM to achieve the best posture in the preparation phase. In the example illustrated in Figure 1-22 we show how the CM can be adjusted along each axis in order to attain different body poses similar to what we

described here for the sprint runner example.

Inertia shaping. While the rotational inertia in two dimensions can be effectively viewed as a disk with varying radius, we can visualize it as an ellipsoid in three dimensions. It then becomes less intuitive to implement a 3D inertia handle for two reasons: first, changing one of the inertia values alters multiple axes of the equivalent ellipsoid; and second, the ellipsoid can itself change orientation as well. Inertia shaping of a three-dimensional virtual character is specially difficult as it is highly under-constrained, meaning there might exist many body configurations that can satisfy a desired inertia shape. While we leave the challenge of creating a user-friendly 3D inertia handle for future work, we have implemented a simple inertia shaping method based on the same principles used in creating the 2D inertia handle. Ignoring the control of the orientation of the inertia ellipsoid, we can consider three perpendicular disks, each with an adjustable radius. It is then up to the animator to use one, two or all three axes in the inertia shaping process. Figure 1-23 demonstrates results of changing inertia along x, z, and x and z axes simultaneously. We must, however, emphasize that we do not provide a mechanism to allow the user to control the orientation of the inertia ellipsoid. Furthermore, avoiding singular configurations is left to the animator's experience in the current interface.

Jump and slide motion. Figure 1-24 shows a jump motion followed by a sliding sequence on a "glassy" surface. This example demonstrates how template trajectories and physics-based handles can be combined to create a dynamic motion in a 3D environment. Similar to the planar example, this jump motion takes advantage of setting foot pins at take off and landing with varying gains to achieve the best postural results. Positional constraints set by the pinning process can also be interpolated between the keyframes using a linear or an ease-in ease-out function. We used the tilted glassy surface to highlight this aspect of our system. Pin positions are set at landing as well as when the character reaches the end of the surface, where the motion generated by the interpolated pin positions conveys a sense of sliding on a slippery object.

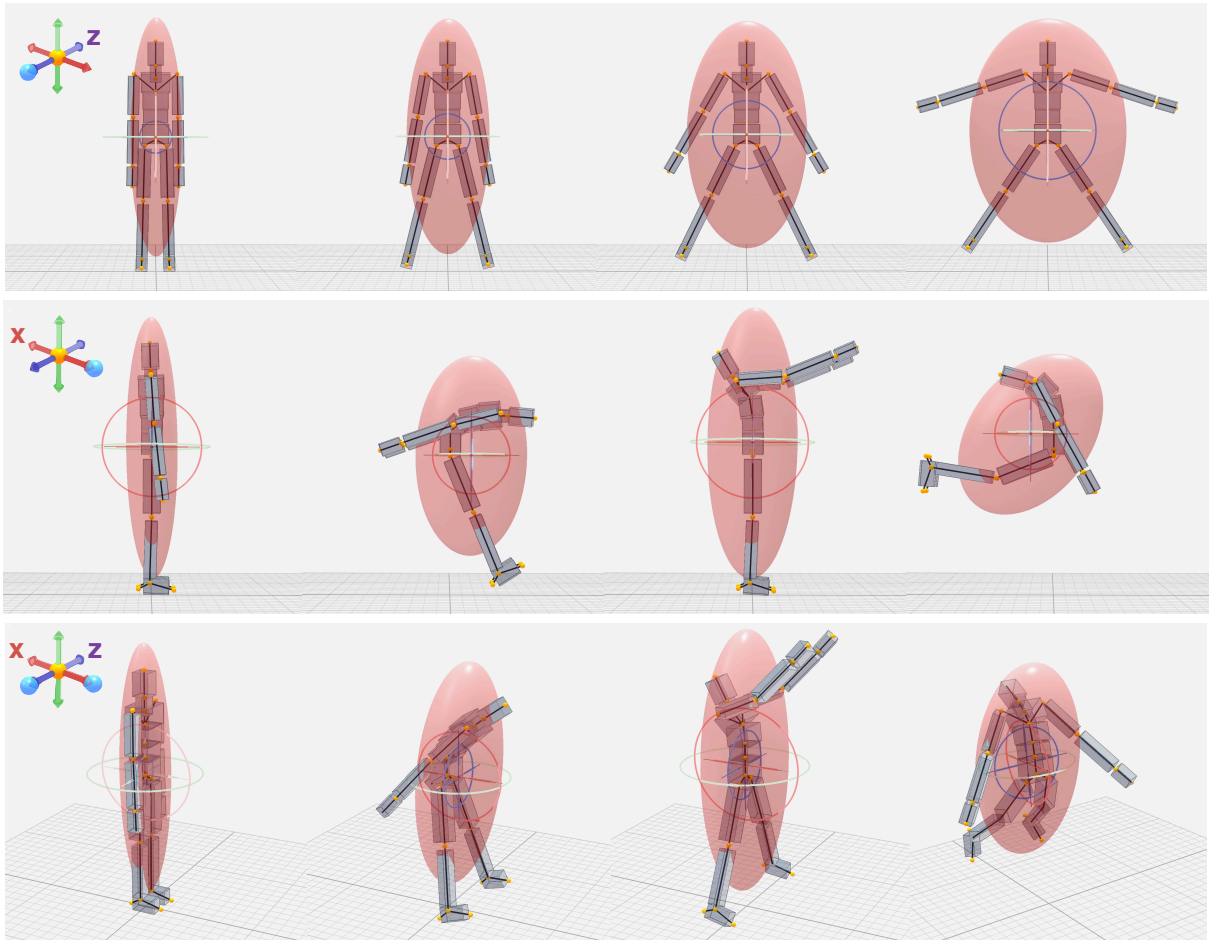

Figure 1-23: *3D Inertia handle. Top: increasing inertia along the z axis. Middle: first decreasing the inertia along the x axis, then increasing it to a near singular configuration, and finally decreasing it. Bottom: combing the actions of the first two rows to achieve the desired poses.*

Figure 1-24: *3D jump. Scene setup is shown on top where the character jumps onto a slippery surface (transparent blue plane) before it lands on the ground. The sequence shows the results of integrating 3D templates and IK handles to create a dynamic motion. The foot pins are interpolated on the slippery surface to create the effect of foot sliding.*

1.6 Summary

We derive expressions for the center of mass, momentum, and the magnitude of the inertia tensor, and we use these as constraints within an inverse kinematics solver to allow animators to easily animate physically plausible motion. Furthermore, physically based template trajectories help an animator produce trajectories that respect physics during dynamic activities. The solver includes soft constraints on the pose of all joints to encourage natural poses, while producing postures that meet reaching objectives and foot placements with constraints of adjustable weight. Re-rooting likewise allows an animator to easily specify local pose changes. We present results for both planar and three-dimensional characters with a variety of different morphologies. Creating physically plausible character motion through keyframing is challenging, and we expect our results to be relevant to character animators of all levels of experience.

Bibliography

Buss, S. R. Introduction to inverse kinematics with Jacobian transpose, pseudoinverse and damped least squares methods, 2004. unpublished survey.

Deo, A. and Walker, I. Overview of damped least-squares methods for inverse kinematics of robot manipulators. *Journal of Intelligent and Robotic Systems*, 14(1):43–68, 1995.

Murray, R. M., Li, Z., and Sastry, S. S. *A Mathematical Introduction to Robotic Manipulation*. CRC, March 1994. ISBN 0849379814.

OH, H. G. A. A. M. R. L. W.-P. A. Anthropometry and mass distribution for human analogues. volume 1. military male aviators. Final report, Anthropology Research Project, 1988.

Yamane, K. and Nakamura, Y. Natural motion animation through constraining and deconstraining at will. *IEEE Transactions on Visualization and Computer Graphics*, 9:352–360, 2003.