Direct Control of Simulated Non-human Characters

Junggon Kim and Nancy S. Pollard
Carnegie Mellon University

Abstract—We present an intuitive technique to control simulated self-propelled characters. In our system, a user can control the motions of a character by directly guiding the character to a desired position with mouse drags while a physics simulation determines the character's motion. Based on the user input, the system automatically computes the actuator command that causes the character to follow the user's intention as closely as possible while respecting the underlying physics. This direct control can be more intuitive to use than existing methods, such as controlling character joints to track a given joint trajectory or set of keyframes, especially when physically plausible dynamic motions are desired. We show the power of our method by creating realistic motions of various kinds of characters in our interactive user interface system, including rigid characters, characters with deformable bodies and rigid skeletons, and self locomoting characters whose bodies form closed loops.

Index Terms—Physically-based character animation, interactive direct control, physics simulation.

1 Introduction

Traditional animators devote much time, practice, and study to making characters appear to move in appealing and natural ways. While portraying a character's personality and style are perhaps of greatest importance, it is also important that a character's motions appear physically plausible and motivated. For example, a character should appear to use its own muscles to perform dynamic maneuvers while interacting with the (changeable) environment (Figure 1). For this reason, creating a character animation through physics simulation is promising, as the underlying physical laws constraining a character's motion can be handled automatically and "under the hood," while the animator is free to focus on the character's performance.

Achieving this goal is quite difficult, however, because finding a good actuator command resulting in a desired motion is not in general an easy problem. Though there are many effective controllers specialized for specific tasks such as walking for a human like character, it is still currently out of reach to obtain a general controller which can be effectively applied to various kinds of characters and tasks.

Designing such a mighty controller is not our goal. Instead, we make use of the human user as a high-level controller by borrowing their intuition about dynamic motions such as the manner of movement and the timing. We believe that people do have a good sense of how a character should move to achieve a desired, physically realistic motion. We also believe that this hypothesis is still valid to a large extent even for non-human like characters such as the ones used in our experiments.

One approach, which is also followed in this paper, is to provide users with an interactive control of a running physically based character simulation. The user directs the animation by, for example, specifying a desired keyframe as in the work of [1], and the system—in real-time or at an animator's desired speed—produces physically correct motion with only legitimate internal actuating forces, as if the motion were being controlled by the character's muscles.

However, even in such an interactive system, it is still difficult to find a good user input such as the keyframe, because it is difficult to exactly predict the effect of the keyframe on the resulting global character motion due to the underactuated nature of the simulated characters—the global position and orientation of the character is unknown and is determined by solving the equations of motion.

We present a technique that allows a user to directly control the motion of a simulated character intuitively in Cartesian space. For example, in Figure 2 the user created a jumping motion of a starfish character by simply dragging the mouse cursor upward quickly to lead the character's center along a desired trajectory, and our system instantly transformed the user input into appropriate actuator commands for physics simulation of the character model. Using our system, the user can control the character's bones and the center of gravity directly in Cartesian space while the simulation is running behind the user interface system.

This approach may appear similar to prevailing IK(inverse kinematics)-based methods in the sense that the user controls the character directly in Cartesian space instead of the joint space (See, e.g., Yamane and Nakamura [2]). However, unlike IK-based approaches, we must consider the underactuated nature of the character, which cannot be handled in kinemat-

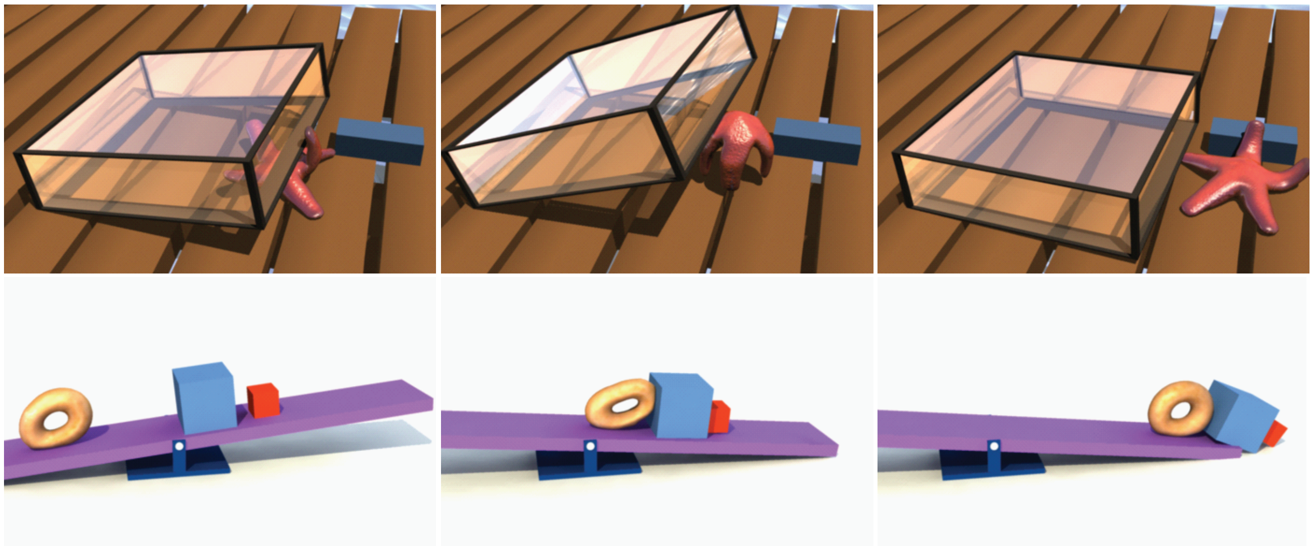

Fig. 1. **Examples of physically simulated character motions using our direct control:** The motions of a starfish and a donut character were created through physics simulations controlled by interactive user inputs such as mouse drags. Our system transforms the user inputs, e.g., the given target position of the body, into an actuator command, such as joint torque or acceleration, for the simulation running behind the interface system. Each character consists of a soft body and a skeleton, and the character motion is generated by using only the internal forces from the actuators installed in the skeleton. There are no external forces acting on the characters except the gravity and the frictional contact forces to handle the complicated interaction between the characters and the changeable environments. As a result, the characters should appear to be using their own muscles to generate the motions. **(Top)** A starfish is trying to escape from a glass case by attempting a few trials, e.g., lifting up the obstacle's edge using its arm (the first frame) or hitting the obstacle with a jump (the second frame). **(Bottom)** A selfish donut is pushing other objects outside of the seesaw.

ics.

We transform the user control into an appropriate joint command to make the character follow the user's intention as closely as possible while still respecting the laws of physics. Therefore, if the user wants to make the character do a flip, he or she must learn how to move the character's body so that the character pushes off with the appropriate momentum to achieve the flip. This learning process is simpler than it may seem due to the real-time trial and error nature of our interface system, and also results in a faithful physical depiction of the motion that incorporates the user's own stylistic considerations.

Though our direct control is applicable to any kind of character model, we have found that the system can be quite effective for creating dynamic motions of non-human like characters such as the ones shown in Figure 1, because usually no high quality motion data (e.g., motion captured data) exists. For such cases, alternative techniques may require large amounts of time and effort to generate physically plausible motions.

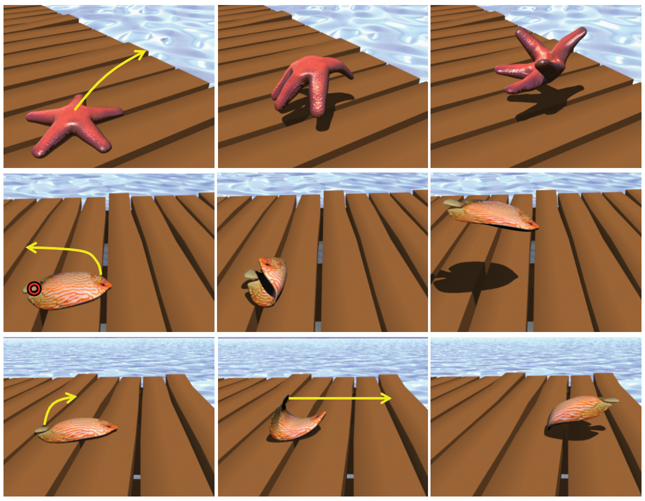

Fig. 2. **Motions of a starfish and a fish characters:** **(Top)** The starfish jump was driven by clicking the center part of the character and then dragging the mouse cursor (the yellow arrow line) upward quickly. **(Middle)** The fish jump motion was initially guided by the mouse drag while maintaining a fixed orientation of the tail (red circle). Then we switched to the keyframe control to get a stretching motion for jump. **(Bottom)** The turnover was guided by the mouse drag.

2 Related Work

Physically based approaches have been explored by many researchers to generate physically plausible character motions while considering environmental interaction. Hodgins et al. [3] presented an algorithm for animating dynamic athletic behaviors using control algorithms, and control based approaches have been pursued by a number of other researchers. In particular, in the computer graphics community, many effective controllers have been developed to generate bipedal character motions, e.g., by using a finite state machine (Yin et al. [4]), but their applications are mostly limited to walking or running of human-like characters. Liu et al. [5] presented a sampling based technique to reconstruct contact-rich captured motions such as rolling. However, their approach cannot be directly applied to non-human or inanimate characters because it is difficult to obtain reference motions with good quality for such characters.

Laszlo et al. [1] introduced an interactive simulation control technique. In their approach predefined poses are mapped to user inputs such as keystrokes or mouse drags, and a PD controller generates joint torques to compensate for differences from the desired joint angles. Such an interface, however, may not be easy to use when the number of motion primitives is large. It is also difficult to select the right key poses with the right timing because the effect of the user input on the simulated character motion is hard to predict when the character is underactuated.

Formulating control in terms of high-level tasks can give more intuitiveness in building controllers for balancing and locomotion (Macchietto et al. [6], de Lasa et al. [7]). Sentis and Khatib [8] introduced a task-space control algorithm and applied it to the simulation of humanoid robots. Using their whole body control framework, they generated various kinds of humanoid motions by combining multiple tasks defined in both Cartesian and joint spaces. Our user interface system also provides similar user inputs to control the physics simulation of characters, but with more emphasis on interactivity and ease of use. Moreover, our formulation can handle a broader range of characters, including characters with deformable bodies and self locomoting closed loops.

3 Our User Interface

The most interesting feature of our user interface system is that the user can create dynamic motions of a character by directly guiding the character's bones and the center of gravity with mouse drags. The character motion is generated through a fully 3D physics simulation running behind the interface system. The feedback rate is fast enough for users to control the motion interactively, and if needed, it can be set to a desired speed by users (e.g., to slow the simulation down to more carefully control complex motions). The user must use his or her own intuition about the physics of the character in designing a motion and catching the proper timing. Then our interface translates the user's command into joint commands such as torque or acceleration for the simulation. The following is a short description of each control provided by the system, and their mathematical formulation will be presented in Section 5.

- **Bones**: The user can set up a desired position and orientation of a bone in the skeleton by using a graphical interface or a script. The mouse cursor can be regarded as the desired position of a selected bone so that the user can change it continuously by dragging the mouse during the simulation. For example, guiding the position of the lamp character's head to move downward and then rapidly upward with a mouse drag will make a jump motion (Figure 3, frame 1 $\sim$ frame 3). The user can make it jump higher and farther by simply dragging the mouse cursor upward more quickly. If the user set up a desired orientation of the character's foot, its orientation with respect to the inertial frame will be controlled continuously even when the character is in mid-air (Figure 3, frame 4).
- **Center of gravity**: The center of gravity (CoG) of the character's skeleton can be controlled by a mouse drag. The user may use this feature, for example, to guide a self locomoting character whose body forms a closed loop (Figure 1, bottom) or to maintain balance after the high speed landing by keeping the CoG over the foot (Figure 3, frame 6).
- **Joints**: The user can specify a desired joint angle for each joint. In the lamp character demo in Figure 3, the user set up desired angles for some of the joints to prepare the landing (frame 5). The foot orientation is still maintained as much as possible while taking the new joint controls into consideration because the user set up the orientation control to be valid until the foot touches down to the ground. Joint limits and torque limits for the character can also be considered automatically when the user guides the character's motion.

A preview function is provided in order to guide the simulation more conveniently (Figure 4). Since the user input, such as the desired position of a body, and the resulting simulation output are seen in the same Cartesian space, showing a glimpse of a short future motion corresponding to a trial input can give the user good intuition about the effect of the input on the output motion. By using the preview function, effective fine tuning of the direct control can be made to create, e.g. a jump followed by a landing on a specific place as shown in Figure 4. To make a prediction of the simulated motions, we assume the user command remains constant throughout the time of the preview. Though we have not tested this in our system, multiple previews with input sampling could

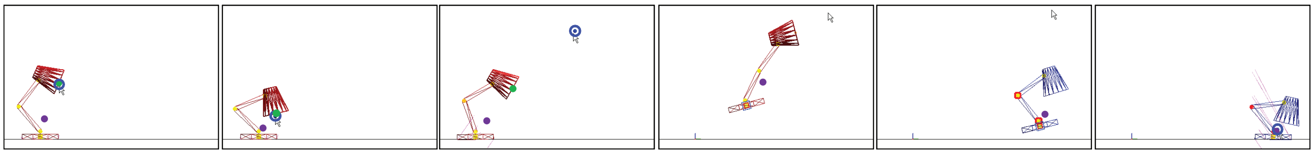

Fig. 3. **A jump motion of a lamp character (screen captured while driving the physics simulation with user inputs):** Using a mouse drag, we guided the character's head (green dot) to make it crouch, then stretched rapidly to make the jump (frame 1 $\sim$ frame 3). The mouse cursor with a blue circle denotes the desired position of a controlled point, here the green dot. When the lamp's foot started to leave the ground, we set up a constraint on the foot to maintain the inclined foot orientation until it touched the ground (frame 4). When the character began falling, we set up desired joint angles to prepare the landing (frame 5). To prevent the character from tumbling over, we controlled the position of the center of gravity (magenta dot) using the mouse cursor (frame 6). In our implementation, physics simulation starts with pressing the mouse button, and stops when the button is released. The simulation runs in the background while the user guides the character using the constraints. Invoking and revoking the constraints can be done through predefined combinations of a mouse click and keyboard inputs at any time when the simulation is paused. In this example, however, we preprogrammed the controls applied to the foot and the joints so that they can be automatically set up during the jump for user's convenience.

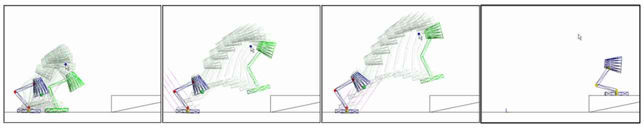

Fig. 4. **Previewing** a short future motion can help the user guide the simulation more easily. In this example the user was able to create a jump on a box in three trials. The user can get an intuition for the relationship between the input (the target position of the head part shown as the mouse cursor with a blue circle) and the output (the trajectory in green) through the preview mode.

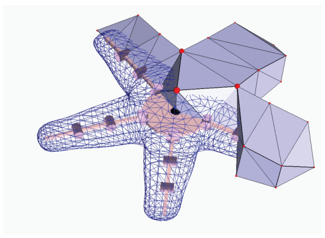

Fig. 5. **Simulation model for the starfish character** This figure shows an example of the simulation models used in our experiments. The starfish model has a skeleton, consisting of rigid bones and joints, and a soft body attached to the skeleton. A virtual 6-DOF passive joint (root joint) connects the center bone to the ground to represent the global position and orientation of the character. The wire frame represents the fine surface of the character's deformable body, and the solid mesh enclosing the surface is for finite element analysis in our simulation system. Some of the volume mesh nodes are fixed to the bones to attach the deformable body to the skeleton, and the others are free to move while being constrained by visco-elastic forces and frictional contact forces.

provide users with more convenience in choosing an appropriate input as done in Laszlo et al. [9].

Keyframing-based simulation is also provided in our system. The user can set up desired key poses by dragging a link with the mouse through an inverse kinematics approach, and then physics simulation runs to obtain the resulting character motion from a reference pose trajectory obtained by interpolating the keyframes with a B-spline. Note that the reference pose trajectory is obtained for active joints only, the joints at which the character could plausibly exert internal muscle forces. Though the keyframe control of simulation in practice will be found unsuitable for generating a long sequence of motion due to the difficulty of predicting the resulting global motion of the character, it can be effectively used to generate a short motion segment such as the stretching phase in jumping (Figure 2, middle).

4 Physics Simulation

We assume that every character has a skeleton. The skeleton can be any set of rigid bodies connected by various types of joints, where one of the bones is connected to the ground by a virtual joint – we call this virtual joint the *root joint*. In order to make the character appear to be self-actuated, the root joint is assumed to be passive so that no force is exerted on the character through the joint. The simulation is driven by joint commands (e.g., torque or acceleration) on the active joints only.

We also attach soft bodies to the skeleton to capture the secondary motions of the character body such as deformation and jiggling. In this case, we use a coarse mesh with an assumption of lumped mass to represent the behavior of the deformable body, and consider the character's fine surface in obtaining the mechanical properties of the body and handling collisions between the character and the environment (Figure 5). Our simulation system is fast enough to be applied to an interactive animation control system such as the one shown in this paper. Details of our simulation system, which combines active skeleton simulation with a passive finite element simulation of deformable flesh, are given in [10]. However, our formulation of constraints for direct control, which is the focus of this paper, is general and can be used with any simulation system, including cartoon or alternative physics simulation systems, as long as linear relationships can be obtained between user-controlled parameters or their derivatives and simulation variables such as acceleration or torque.

5 Direct Control

In our system, all user inputs (e.g., the desired position of a bone indicated by the mouse cursor) are represented as linear equality and inequality constraints,

$$Au = b \quad \text{and} \quad Cu \le d \tag{1}$$

where u denotes the active joint command for the simulation running behind the interface system. We obtain the necessary joint command by solving the constraints in a robust manner.

We prefer using acceleration as the joint command in our direct control of simulated characters, but our formulation is not restricted to any particular choice of joint command. We will discuss the effect of a different choice in Section 6. In case of using acceleration ($\ddot{q}_a$) as the active joint command, we obtain the acceleration for the passive root joint and the torque needed to actuate the active joints by solving the equations of motion. Finally, we advance the physics simulation to the next time step by integrating the acceleration for both active and passive joints.

5.1 Building Constraints

Global position and orientation of bones

The first type of user inputs we consider are the desired position (x_d) and/or orientation (R_d) of a bone. We will build up linear constraints on the active joint acceleration ($\ddot{q}_a$) from the given user inputs (x_d, R_d) and the current system state.

Let $\ddot{X} \in \Re^6$ denote the linear and angular accelerations of a bone with respect to the inertial frame. Because bone accelerations have a linear relationship with the joint input command, here $\ddot{q}_a$, they can be written as

$$\ddot{X} = \frac{\partial \ddot{X}}{\partial \ddot{q}_a}\ddot{q}_a + \ddot{X}_0 \tag{2}$$

where $\frac{\partial \ddot{X}}{\partial \ddot{q}_a}$ denotes the derivative of the bone acceleration with respect to the joint command, and $\ddot{X}_0$ is the bone acceleration when $\ddot{q}_a = 0$. In our implementation we obtain $\frac{\partial \ddot{X}}{\partial \ddot{q}_a}$ rapidly by analytically differentiating a recursive dynamics algorithm, presented in detail in [10] but outside the scope of this article. Note that, in the case of skeleton driven deformable body systems, we only need to differentiate the dynamics of the skeleton because the joint command does not instantly affect the acceleration of passive nodes representing the deformable tissues. When we calculate $\ddot{X}_0$, however, we must use the equations of motion for the whole system to consider the effects of the viscoelastic forces arising inside the deformable body and the frictional contact forces acting on the soft body surface from the environment.

Now we can set up the following constraint equations for the given desired position and orientation of a bone

$$\frac{\partial \ddot{X}}{\partial \ddot{q}_a}\ddot{q}_a = \ddot{X}_d - \ddot{X}_0 \tag{3}$$

where $\ddot{X}_d = (\dot{w}_d, \ddot{x}_d)$ is a desired bone acceleration which is obtained as

$$\begin{aligned} \ddot{x}_d &= k_p(x_d - x) - k_v\dot{x} \\ \dot{w}_d &= k'_p\, R\, \log((R)^T R_d) - k'_v w \end{aligned} \tag{4}$$

where x_d and R_d are the given desired position and orientation of the bone, x, $\dot{x}$, R, and w are the current position, linear velocity, orientation and angular velocity of the bone respectively, and k_p, k_v, k'_p, and k'_v are the gain parameters defined by the user to control how aggressively the character moves toward a desired position.

Center of gravity

Here the user will give a desired center of gravity x^c_d of the skeleton as the user constraint. Similarly to the desired bone acceleration above, the desired acceleration of the center of gravity is obtained by

$$\ddot{x}^c_d = k_p(x^c_d - x^c) - k_v\dot{x}^c \tag{5}$$

where x^c and $\dot{x}^c$ are the current position and velocity of the center of gravity.

The acceleration of the center of gravity can be written as

$$\ddot{x}^c = \frac{\partial \ddot{x}^c}{\partial \ddot{q}_a}\ddot{q}_a + \ddot{x}^c_0 \tag{6}$$

where the derivative of the center of gravity can be obtained by $\frac{\partial \ddot{x}^c}{\partial \ddot{q}_a} = \frac{1}{\sum_i m^i}\sum_i m^i \frac{\partial \ddot{x}^i}{\partial \ddot{q}_a}$, and here x^i and m^i denote the center of gravity position of the i-th bone and its mass respectively, and $\frac{\partial \ddot{x}^i}{\partial \ddot{q}_a}$ can be easily deduced from $\frac{\partial \ddot{X}}{\partial \ddot{q}_a}$ used in (3). Finally we obtain the following constraint equations on $\ddot{q}_a$ for the user constraints on the position of the center of gravity.

$$\frac{\partial \ddot{x}^c}{\partial \ddot{q}_a}\ddot{q}_a = \ddot{x}^c_d - \ddot{x}^c_0 \tag{7}$$

Note that we can control the center of gravity only when the character is in contact with the environment such as the ground. To control the center of gravity in this way, we need to include a constraint-based contact model between the skeleton and the ground in the equations of motion when we build the constraints in (7), though we in fact use a more realistic penalty-based method for the contacts in the simulation. We will discuss this point further in Section 5.4.

Joints

It is straightforward to consider a desired joint displacement in our formulation. Let q_d^i be a given desired displacement of the i-th active coordinate. Then we can build the following constraint equation

$$A^i \ddot{q}_a = \tilde{k}_p(q_d^i - q^i) - \tilde{k}_v \dot{q}^i \tag{8}$$

where $A^i = [0, \cdots, 1, \cdots, 0]$, and q^i and $\dot{q}^i$ are the current displacement and velocity of the coordinate respectively.

To handle the limits of joint displacement, we present a simple and effective method to respect the joint limit rigorously, but in a damped manner to avoid any sudden impulse on the character. For each i-th joint coordinate, we define short ranges of the displacement, $[q_i^L, q_i^L + \Delta_i]$ and $[q_i^U - \Delta_i, q_i^U]$ where our joint limit constraints on the lower and upper limit q_i^L and q_i^U can be activated. When the joint enters one of the areas with a velocity toward the corresponding limit, we must control the joint to stop before reaching the limit by restricting its acceleration range. We determine the acceleration limit by assuming a parabolic curve for the joint trajectory whose maximum or minimum reaches the upper or lower limit and passes through the current displacement with the current velocity. If we restrict the joint acceleration to less than the acceleration of the parabola, then the joint trajectory will always remain within the boundary. Therefore, we set up the following inequality constraints for the joint limit

$$\begin{aligned} \ddot{q}_i &\le \ddot{q}_i^U \quad \text{if } q_i \in [q_i^U - \Delta_i, q_i^U] \text{ and } \dot{q}_i > 0 \\ \ddot{q}_i &\ge \ddot{q}_i^L \quad \text{if } q_i \in [q_i^L, q_i^L + \Delta_i] \text{ and } \dot{q}_i < 0 \end{aligned} \tag{9}$$

where $\ddot{q}_i^U$ and $\ddot{q}_i^L$ are the upper and lower limits of the acceleration of the i-th joint coordinate obtained from the parabola. The acceleration limit depends on the current displacement and velocity as well as the location of the joint limit, and is obtained by $\ddot{q}_i^U = -\frac{\dot{q}_i^2}{2(q_i^U - q_i)+\beta}$ and $\ddot{q}_i^L = \frac{\dot{q}_i^2}{2(q_i - q_i^L)+\beta}$ where β is a small positive value to prevent dividing by zero.

Joint torque limits can also be incorporated into our formulation. We use the following inequality constraints for the torque limits

$$\begin{aligned} \frac{\partial \tau_a}{\partial \ddot{q}_a} \ddot{q}_a &\le \tau_a^U - \tau_{a,0} \\ -\frac{\partial \tau_a}{\partial \ddot{q}_a} \ddot{q}_a &\le \tau_{a,0} - \tau_a^L \end{aligned} \tag{10}$$

where τ_a^L and τ_a^U are the lower and upper limits of the active joint torques, and $\tau_{a,0}$ denotes the torques when $\ddot{q}_a = 0$. Since $\frac{\partial \tau_a}{\partial \ddot{q}_a}$ is a byproduct of the differentiation of the recursive dynamics algorithm, which will also be called to build the bone constraints, we can set up the torque limit constraints efficiently.

5.2 Constraint Solving

All the constraints in Section 5.1 can be merged into a set of linear constraints on the active joint accelerations in the form of

$$A\ddot{q}_a = b \quad \text{and} \quad C\ddot{q}_a \le d \tag{11}$$

where A, b, C and d are nonlinear functions of the current system state $(q, \dot{q})$. The equality constraint equations represent the user's intention for guiding the physics simulation. The inequality constraints form a convex space for the joint acceleration where the joint and torque limits are satisfied.

Though the constraint equations look simple, we need to be careful when we solve the equations in order to make the user interface system robust. Indeed, in many situations A is likely to be ill-conditioned because of singularity. For example, the fish lying on the dock in a straight shape in Figure 2 is a good example of a singular pose where the head and tail of the character cannot be accelerated horizontally at the moment. In this case, if the user gives a wrong input (e.g., a desired position of the head or tail requiring such acceleration) then a naïve solution of the constraint equations will become very large in an attempt to meet the constraints exactly. Such a situation is very likely to cause the simulation to diverge. The user interface system must be robust to such user inputs, which may happen regularly during direct user control of characters.

To avoid problems due to singularities, we build the following quadratic program (QP) by changing the equality constraints into an optimization

$$\min_{\ddot{q}_a} ||A\ddot{q}_a - b||^2 + \alpha||\ddot{q}_a||^2 \quad \text{s.t.} \quad C\ddot{q}_a \le d \tag{12}$$

where $\alpha > 0$ denotes the weight of the second term of the objective function. Changing the equality constraints into such an optimization form is a well-known technique to obtain a stable solution for inverse kinematics [11]. The minimum point of the quadratic function can be obtained by $\ddot{q}_a^* = V(S^\ddagger)^T U^T b$ where $A = USV^T$, $S^\ddagger = \text{diag}(\sigma_i/(\sigma_i^2+\alpha))$, σ_i denotes the singular values of A, i.e., $S = \text{diag}(\sigma_i)$, and α is a parameter to damp out the effect of small singular values.

Different priorities can be imposed on the user inputs. For example, in our experiments, mouse drags were set to be secondary while other constraints on links, such as maintaining orientations of the lamp character's foot and the fish tail in Figure 3 and 2(middle) respectively, were set to be primary. We handle the different priorities by solving the constraints sequentially as done in the literature ([2], [8], [7]). Let $A_p x = b_p$ and $A_s x = b_s$ be the primary and secondary equality constraints from the user inputs. We first choose a particular solution of the primary QP problem ($A_p x = b_p,\ \ Cx \leq d$). Then we set up a linear subspace $x = x_p + N_p y$ with the null space of A_p and a new variable y for exploring the subspace. Note that all the points in the subspace satisfy the primary constraints at the same level, and the particular solution has a minimal norm among the solution space. We will find a solution which meets the secondary constraints within the subspace. The secondary QP problem is formulated in terms of y by substituting $x_p + N_p y$ for x in $A_s x = b_s$ and $Cx \leq d$, and we obtain the final solution by $x = x_p + N_p y_s$ where y_s is the solution of the secondary QP problem.

5.3 Closed Loops

Our acceleration-based formulation can also handle closed loops in the skeleton (Figure 1, bottom). We assume that all the joints in the closed loops are active so that their trajectory can be controlled actively.

We build a virtual tree-topological system by cutting each closed loop at a single joint or link. Keep in mind that we should control the active joints in the open loop system in a coordinated way in order to keep the original closed loops. We do this by constructing constraint equations on the joints in the closed loops and finding a joint subspace which satisfies the closed loop condition.

From kinematics, the closed loop constraints can be written as

$$f(q_a) = I \tag{13}$$

where f represents the forward kinematics of the joint loop which is a nonlinear function of the active joint coordinates q_a, and I is the identity matrix. By differentiating the equation with respect to time, we can obtain a linearized constraint on $\ddot{q}_a$

$$J\ddot{q}_a = -\dot{J}\dot{q}_a \tag{14}$$

where $J = \frac{\partial f}{\partial q_a} \in \Re^{6 \times n_a}$ denotes the Jacobian of the closed loop constraints and $\dot{J}$ is its time derivative. The closed loop constraints are treated with the highest priority in our implementation, and all other constraints are solved within the solution subspace which can be expressed with the null space of J and a particular solution obtained by the Moore-Penrose pseudoinverse.

Though the joint command $\ddot{q}_a$ corresponding to the user inputs satisfies the linearized constraints in (14) at every time step, the system pose at the next time step may slightly deviate from the original nonlinear closed loop constraints in (13). As the simulation proceeds, the tiny error at each time step accumulates and the closed loops can break down. We compensate for this error by projecting the system pose onto the curved subspace using the Newton-Raphson method. At most a few iterations per time step suffice in our experiments.

5.4 Contact Modeling

We use a penalty-based contact model in the physics simulation, which ensures plausible physics for both rigid and deformable bodies. However, in some circumstances it may be advantageous to temporarily assume that contacts are hard constraints in order to compute character actuations that make proper use of the character's ability to push off the ground. In particular, if we do not use this approach, it is not possible for the user to control the character's center of gravity, because under the penalty-based contact model the character appears as an isolated system where acceleration of the center of gravity cannot be affected through internal actuation. (In such a case, the term $\frac{\partial \ddot{x}^c}{\partial \ddot{q}_a}$ in (7) would be the zero matrix.)

When we build the constraints for the computation of the joint command, we temporarily create a joint to express the contact between a bone and the ground. For example, in the case of the lamp character, we replace the original 6-DOF passive root joint with a 3-DOF ball joint, 1-DOF revolute joint, or 0-DOF welding joint when the number of contact points is 1, 2, and 3 or more respectively. If there is no contact, the 6-DOF free joint will remain as the root joint. After building the constraints, the temporary joint for contacts is replaced with the original root joint and the penalty-based contact model is applied for the simulation. This treatment of ground contact can produce interesting effects. For example, if the user attempts to have the lamp character accelerate forward quickly on an icy surface, our system will compute joint accelerations that attempt to accomplish this motion, and the simulated lamp, not finding sufficient friction for the attempted motion, will slip on the surface and possibly fall.

6 Results and Discussion

We tested our algorithms for creating dynamic motions on various kinds of characters such as a lamp, worm, fish, starfish, and a donut-shaped character. All the characters tested, except for the lamp character, have elastic soft bodies attached to their skeletons. For the passive root joint, which connects one of the bones in the skeleton to the ground, we used 6-DOF joints for the lamp and starfish, and 3-DOF planar joints for the worm, fish, and donut characters, although the character models and some of the motions were fully 3D (e.g., see Figure 1, top). The control loop can be executed at a lower rate (e.g., 50Hz $\sim$ 1kHz) than the simulation loop, and the simulation speed can be adjusted by the user. Though the user is able to control most of the tested characters in real time,

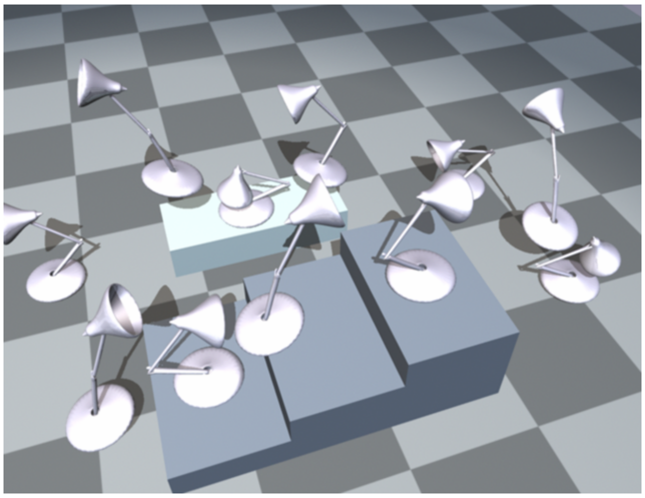

Fig. 6. **A lamp motion:** A 3D lamp motion including jumping up the stairs and sliding on a frictionless box was created in our interface system.

in such a simulation setting, it can be difficult for the user to react quickly and correctly enough to guide the character in a desired way as pointed out in [1]. In our experiments, we preferred controlling the characters at a slow simulation speed, e.g., $3 \sim 8$ times slower than real time.

The elapsed time for creating a physically simulated motion in our system depends on the type of the character and the required controls. For example, even a novice was able to control the selfish donut character in Figure 1 (bottom) to push the other objects outside of the seesaw successfully within a few minutes, and this is because the required control is very simple — only guiding the position of the center of gravity with the mouse suffices for the task. In the case of the starfish escape in Figure 1 (top) in which the user is usually required to manipulate different body parts successively using the mouse, the users had to be trained for some time to become familiar to the interface. Once the users become accustomed to the system, however, they were able to create the motions of the escaping starfish within a few minutes. For the 3D lamp example shown in Figure 6, it took about 30 minutes for a proficient user to create the long sequence of the motion consisting of consecutive jumps and balances.

We formulated all the constraint equations for our direct control in terms of the active joint acceleration. Under the acceleration-based formulation, the constraint solver chooses a minimal acceleration among possible solutions. Therefore, the current momentum of the character is likely to be preserved as much as possible in the next time step, and this leads to a smooth dynamic motion starting from a stationary state. Though the momentum-preserving nature of the acceleration-based formulation is good for starting dynamic motions, it is not proper for generating less dynamic motions (Figure 2, bottom) and stopping motions such as the landing phase of the lamp character (Figure 3, rightmost). We handle situations such as these by modifying the original formulation in (11) in terms of the active joint velocity as

$$A\dot{q}_a^d = hb - A\dot{q}_a \quad \text{and} \quad C\dot{q}_a^d \le hd - C\dot{q}_a \qquad (15)$$

with an assumption of the explicit Euler integration on the velocity, $\dot{q}_a^d = \dot{q}_a + h\ddot{q}_a$ where $\dot{q}_a$ and $\dot{q}_a^d$ represent the velocity at the current and next time step respectively and h denotes the step size for integration. We solve the constraint equations by using the method described in Section 5.2 to obtain a feasible minimal joint velocity at the next time step. Finally we set up the active joint acceleration, the joint command for the simulation, as $\ddot{q}_a = \frac{1}{h}(\dot{q}_a^d - \dot{q}_a)$ and then perform the dynamics simulation to get the system state at the next time step. The constraints can also be formulated in terms of joint torques, and in this case, the resulting motions appear to use less torque than the other formulations.

The physics simulation driven by joint acceleration does not produce compliant joint motions that are responsive to unexpected external forces. This problem, however, can be effectively solved by placing an additional forward dynamics process after the current physics solver which outputs the acceleration of the passive joints and the torque for the active joints. In this case, the active joint torque is used as the input command of the forward dynamics simulation. Note that, for the user to observe the effect of compliance using this technique, the unexpected external forces must be used in the final forward dynamics simulation only.

7 Conclusion

We presented an acceleration-based formulation for direct Cartesian control of underactuated characters with deformable bodies and closed loops, and showed the effectiveness of the control by creating dynamic motions of various kinds of characters in our user interface system. In the system the user can interactively guide the character simulation using his or her own physical intuition through direct control in Cartesian space using mouse drags, for example, for guiding bones and the center of gravity. Realistic and physically plausible character motions can be created without using physically correct reference motion data which is usually unavailable for non-human-like characters such as the ones tested in our experiments.

In our current implementation, the mouse is the only device used to control the characters interactively and continuously and this may limit the user from fully realizing his or her intentions when guiding the simulated characters. Using an input device that would transmit its 3D position and orientation at the same time would enhance the performance of our interface significantly. A real-time motion capturing system could be used for even more sophisticated control as in [12].

Though an artist may want to bend physics in some cases to obtain a desired effect, our current implementation can only produce physically plausible motions. Allowing intentional external force on the root joint could be one solution for this problem, and it would be quite challenging future work to find an optimal trade-off between the realism of the motion and the desired task in the artist's mind.

Acknowledgments

This research was partially supported by NSF award CCF-0702443. The authors gratefully acknowledge Moshe Mahler for help with Maya modeling and rendering. We also thank the graphics group at Carnegie Mellon for their helpful comments.

References

[1] J. Laszlo, M. van de Panne, and E. Fiume, "Interactive control for physically-based animation," in *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000, pp. 201–208.

[2] K. Yamane and Y. Nakamura, "Natural motion animation through constraining and deconstraining at will," *IEEE Transactions on Visualization and Computer Graphics*, vol. 09, no. 3, pp. 352–360, 2003.

[3] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien, "Animating human athletics," in *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, Aug. 1995, pp. 71–78.

[4] K. Yin, K. Loken, and M. van de Panne, "Simbicon: simple biped locomotion control," *ACM Transactions on Graphics*, vol. 26, no. 3, 2007.

[5] L. Liu, K. Yin, M. van de Panne, T. Shao, and W. Xu, "Sampling-based contact-rich motion control," *ACM Transactions on Graphics*, vol. 29, no. 4, 2010.

[6] A. Macchietto, V. Zordan, and C. R. Shelton, "Momentum control for balance," *ACM Transactions on Graphics*, vol. 28, no. 3, 2009.

[7] M. de Lasa, I. Mordatch, and A. Hertzmann, "Feature-based locomotion controllers," *ACM Transactions on Graphics*, vol. 29, no. 4, 2010.

[8] L. Sentis and O. Khatib, "A whole-body control framework for humanoids operating in human environments," in *Proceedings 2006 IEEE International Conference on Robotics and Automation*, May 2006.

[9] J. Laszlo, M. Neff, and K. Singh, "Predictive feedback for interactive control of physics-based characters," in *Proceedings of Eurographics*, 2005.

[10] J. Kim and N. S. Pollard, "Fast simulation of skeleton-driven deformable body characters," *ACM Transactions on Graphics* (in press).

[11] Y. Nakamura and H. Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," *Journal of Dynamic Systems, Meas., and Control*, vol. 108, pp. 163–171, 1986.

[12] M. Dontcheva, G. Yngve, and Z. Popovic, "Layered acting for character animation," *ACM Transactions on Graphics*, vol. 22, no. 3, 2003.

Junggon Kim is currently a Project Scientist in the Robotics Institute at Carnegie Mellon University. He received a B.S. and M.S. in Mechanical Engineering from Seoul National University in 1996 and 1998, and a Ph.D. from the Robotics Laboratory at Seoul National University in 2007. His research interests include robotics, motion planning, and computer graphics and animation. Contact him at junggon@cs.cmu.edu.

Nancy S. Pollard is an Associate Professor in the Robotics Institute and the Computer Science Department at Carnegie Mellon University. She received her PhD in Electrical Engineering and Computer Science from the MIT Artificial Intelligence Laboratory in 1994, where she performed research on grasp planning for articulated robot hands. Before joining CMU, Nancy was an Assistant Professor and part of the Computer Graphics Group at Brown University. She received the NSF CAREER award in 2001 for research on 'Quantifying Humanlike Enveloping Grasps' and the Okawa Research Grant in 2006 for "Studies of Dexterity for Computer Graphics and Robotics. Contact her at nsp@cs.cmu.edu.