Interactive Physically-Based Control of Skeleton-Driven Deformable Characters

Junggon Kim Nancy Pollard

CMU-RI-TR-08-11

June 2008

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

We present an intuitive way to create dynamic motions for physically simulated, skeleton-driven deformable characters. The motion of the character is guided by the user interactively through Cartesian space control and a combination of setpoint selection. To make direct manipulation of a physically simulated character intuitive, we present a novel approach to Cartesian space control, in which the user may directly drive the accelerations or velocities of bones in the character using mouse drags and scripted constraints.

To facilitate direct user control, we must solve the fully nonlinear equations of motion at speeds close to real-time. To achieve this fast computation speed, we model the deformable body as a coarse mesh having an embedded fine surface and apply a geometric recursive dynamics algorithm for fast computation and analytic differentiation. These techniques are not sufficient for good performance, however. We present an improved diagonalization method to efficiently handle mesh elements with large deformation and a number of other optimizations that allow us to achieve simulation speeds that approach real-time.

Contents

1 Introduction **1**

2 Related Work **2**

3 Motion Generation **3**
3.1 Direct Control in Cartesian Space 5
3.2 Variants of the Cartesian Control 7
3.3 Prescribed Control in Joint Space 8

4 Formulation **9**
4.1 Skeleton-driven Deformable Body 9
4.2 Equations of Motion 10
4.3 Elastic Force Computation 11
4.4 Selective Diagonalization 13
4.5 Damping Forces 14

5 Results **17**

6 Conclusion **21**

A Appendix **25**
A.1 Recursive Hybrid Dynamics 25
A.2 Recursive Dynamics for Mass Particles 27
A.3 Derivatives of Link Acceleration 28

1 Introduction

Creating realistic character motion based on physics has received a great deal of attention within the computer graphics community. However, despite great advances in the development of controllers and optimization techniques, this problem is still quite difficult.

The root of the difficulty is that it is not so easy to express how the character should actuate its available joints in order to create a desired motion. Imagine that you are floating in the space. You may not be able to move your body in a desired way because, whenever you actuate your arms or legs with your muscles, the orientation of your body may change in an unfamiliar way and the resulting motion will be different from what you expected. In such a situation you may need special training in order to move confidently and efficiently. The situation can be much better on the ground with gravity. However, though you can move at will by using contact and friction, it is not because the physics becomes easy to solve but because the gifted controller inside your body is well-trained. In fact there is no difference in the basic equations of motion between the first example and the second. The only thing that is new on the ground is the added external forces from contact and friction. Therefore, it is not surprising that it is tremendously difficult to create desired dynamic motions of artificial characters which satisfy physics exactly without the help of a skilled controller such as that found in living creatures.

Designing a good controller is not our goal. Instead, we hope to make use of the user as a controller, borrowing their intuition about dynamic movements, because we believe that people do have a good sense of how a character should move to achieve a desired, physically realistic motion. In this paper we present two approaches to realize the user's intention in creating dynamic motions of skeleton-driven deformable body systems: direct control and keyframe control. In Section 3.1 we introduce a novel technique for direct Cartesian control of motions of skeleton-driven deformable characters. Our transformation converts a low-dimensional control input in Cartesian space into the command inputs on the actuators required to track the user's desired motion. In this mode of interaction, the character uses its actuators to follow the user guidance as closely as possible, while respecting joint displacement or torque limits, and there is no external force acting on the system except contact and friction forces. Note that the user interactively guides the character while the physics based simulation is running, which makes our implementation a form of real time underconstrained inverse dynamics, where a solution must be found that respects the contact between the deformable body and the environment. Because the problem is underconstrained, the motion can have different styles, and in Section 3.2 we present a few variants of the Cartesian control approach which result in different styles of motions. In Section 3.3, we describe prescribed (setpoint) control of the actuators using hybrid dynamics. Here, the trajectory of the active joints of the skeleton is fully determined by a set of user-specified keyframes, and the simulation determines the motion of the passive root joint and the passive nodes that make up the deformable body surrounding the skeleton.

In order to make direct control of the character intuitive, and also to facilitate a "trial and error" approach to animation, we must solve the fully nonlinear equations of motion at interactive rates. We describe our implementation in Section 4. Our

particular contribution in this section is a novel technique to efficiently handle mesh elements with large deformations (Section 4.4). In addition, we present a complete description of our implementation, including a variety of other optimizations required to obtain simulation speeds approaching real-time for our examples.

2 Related Work

Inverse kinematics based motion editing techniques have long been studied. Zhao and Badler [40] introduced an algorithm for adjusting joint angles subject to joint limits while end-effectors attempt to achieve their goals. Lee and Shin [22] presented a hierarchical motion editing system for human figures. Yamane and Nakamura [38] introduced an intuitive pin-and-drag interface for generating motions of articulated rigid body systems. Recently an intuitive pose editing technique for reduced deformable models was presented by Der et al. [9]. Though the motion editing techniques in these various inverse kinematics approaches provide an intuitive interface that is easy to use, their results sometimes look unnatural especially for dynamic motions, because they do not consider the laws of physics and physical constraints on character capabilities.

Realistic character motion has long been recognized as a vital part of animation, and physics based motion generation is receiving more and more attention. Armstrong and Green [1] introduced physical simulation techniques for human models to the graphics community. Hodgins et al. [18] presented an algorithm for animating dynamic athletic behaviors using control algorithms and control based approaches have been pursued by a number of other researchers. Controllers have been developed which track prespecified trajectories (e.g., [20, 42]) or which are designed to closely follow motion capture data (e.g., [39, 33]). However, the ability to interactively influence the behavior of such systems has been limited.

Laszlo et al. [21] introduced an interactive control scheme using PD controllers and motion primitives. In their approach predefined poses or joint commands 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. However such an interface may not be easy to use when the number of motion primitives is large. Even when mouse drags are mapped directly to actuator commands for simple systems, for e.g., a planar Luxo with two actuators in the work of [21], control may not be especially intuitive as the direction of mouse motion will not in general match the direction of character motion. Bergerman and Xu [4] presented a control algorithm for under-actuated manipulators following trajectories on Cartesian space. Their algorithm is similar to one of our Cartesian controls, more specifically, the torque-minimizing Cartesian control described in 3.2. However their formulation does not handle under-constrained and overconstrained situations, which are common in our examples.

An alternative to interactive motion control is motion optimization. Popović et al. [28] for example interactively optimize the motion of rigid bodies. However this approach has not been shown to be practical for animated characters of more complexity due to the time consuming nature of optimization.

Capturing the elastic motions of deformable bodies is one of the most time consuming processes in dynamics simulation. Therefore, many researchers in the computer

graphics community have focused on developing an efficient way to calculate the elastic forces quickly while maintaining good resulting motions. St. Venant-Kirchhoff materials are widely used in the community because they can handle large displacements with relatively simple formulae [25, 7, 2]. A finite volume method was suggested by Teran et al. [35] and they showed that in certain conditions it is equivalent to the finite element method with less computational cost. Though the simple material can handle large displacements, it has trouble with large deformations. Irving et al. [19] presented an algorithm to overcome this problem by applying a special diagonalization that facilitates proper handling of stiffness properties in the case of large deformations. Unlike the work of Irving et al., which diagonalizes the deformation gradient on every element at every time step, we present an algorithm to diagonalize elements with large deformations only. To do this, we present a novel algorithm to identify such elements very quickly (Section 4.4).

Controlling the motion of the deformable body with a relatively small number of control points can lead to a significant speedup in the simulation of deformable characters. Free-form deformation (FFD) introduced by Sederberg and Parry [31] manipulates deformable bodies through an embedding lattice of control points. Faloutsos et al. [10] extended FFD to consider the equations of motion for the animation of deformable characters. Capell et al. [7] used a coarse volumetric mesh embedding the surface of a deformable character and applied the finite element approach to obtain elastic forces within the mesh. They employed a skeleton to control the movement of some mesh nodes. In their approach the bones must lie along the edges of the mesh so that the flesh can rotate around the bones without deforming unnaturally, and they applied additional twist constraints to alleviate this problem. Our skeleton-driven deformable model was inspired primarily by the work of Capell et al. [7], but the skeleton in our model can be any articulated rigid body system with a tree topology and there is no restriction in attaching the deformable body to the skeleton. Galoppo et al. [15] presented a fast algorithm to capture skin deformation of soft articulated characters by decoupling skeleton and skin computations with approximations of Schur complements. We found our formulation convenient for supporting direct user guidance, but it may be possible to incorporate such user guidance into their formulation as well.

There has been much research on the efficient formulation of the equations of motion. Featherstone [14] expressed the dynamics of articulated rigid body systems with spatial notation which leads to an $O(n)$ algorithm for forward dynamics. Rodriguez et al. [30] obtained the spatial operator algebra formulation by identifying structural similarities in the dynamics equations and the equations for discrete-time Kalman filtering. Park et al. [26] introduced a geometric version of the recursive dynamics algorithm based on the theory of Lie groups, and it was extended to obtain recursive forward and hybrid dynamics by Ploen and Park [27] and Sohl and Bobrow [32] respectively.

3 Motion Generation

We describe the equations of motion of a system with an independent reduced coordinates set $q \in \Re^n$. The coordinates of a skeleton-driven deformable body come from skeletal joints including a root joint which connects the skeleton with the ground, and

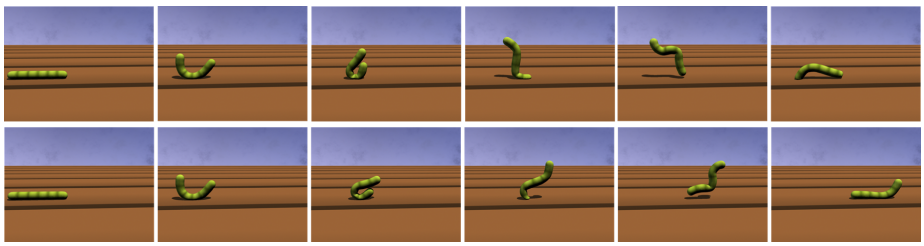

Figure 1: **A jump failure by a bad timing :** The failing motion**(upper row)** was created with the same actuators trajectory as the original jump**(lower row)**, but with slightly different timing. We warped the timing by adding a sine curve, i.e., the new timing t' was generated by $t' = t + \frac{0.05}{t_f}\sin(\frac{2\pi}{t_f}t)$ where t and t_f denotes the original timing and the simulation time respectively.

the nodes representing deformable tissues surrounding the skeleton (Figure 3). The equations of motion of the systems can be written as

$$M\ddot{q} + b = \tau \tag{1}$$

where $M(q)$ is the inertia matrix, $b(q, \dot{q})$ represents the Coriolis, centrifugal and gravity forces, and τ is the forces or torques acting on q.

We assume that the motions of the system are driven by the active skeletal joints, or the actuators only, and the trajectories of passive coordinates representing the root joint and the deformable body are determined by integrating their acceleration which can be obtained from the equations of motion. Suppose that $q = (q_a, q_p)$ and $\tau = (\tau_a, \tau_p)$ where the subscripts 'a' and 'p' denote the active and passive coordinates respectively. Solving $\ddot{q}$ with given set of $(q, \dot{q}, \tau)$ is called *forward dynamics*. In this case the motion of the system can be obtained by integrating $\ddot{q}$, and this is a typical scenario in most physics simulations. In contrast, obtaining $(\tau_a, \ddot{q}_p)$ from given $(q, \dot{q}, \ddot{q}_a, \tau_p)$ is called *hybrid dynamics* or generalized inverse dynamics, and by integrating $\ddot{q}_p$ one can get the complete description of the motion. Note that the traditional inverse dynamics, which calculates τ from given set of $(q, \dot{q}, \ddot{q})$ cannot be solved in our system because there is no way to predefine the trajectory of the passive coordinates without simulation.

In general it is not easy to find an input for the actuating joints, $\tau_a(t)$ or $q_a(t)$ to obtain a desired motion in a situation where there are a mix of active and passive degrees of freedom. Consider trying to create the jump motion shown in Figure 1(lower). If the actuator commands are too small, too large, or have the wrong timing, the worm will most likely not make a jump at all. As an example of this, in Figure 1(upper), we show a motion created with the same joint command trajectory as the original jump motion, but with slightly warped timing. In this case, the worm fails to jump. Fortunately a human user can use their physical intuition quickly to discover the proper timing. In this paper we present two approaches to realize the user's intention in creating dynamic motions of the skeleton-driven deformable body systems. In Section 3.1 we introduce a transformation which converts a low-dimensional Cartesian control input into the command inputs on the actuators. We also present a few variants of the transformation in Section 3.2 which generate different styles of motions. Prescribed

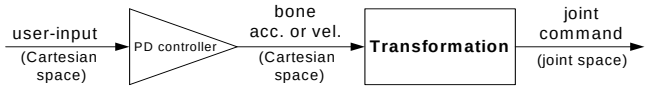

Figure 2: **The direct control in Cartesian space** must transform the desired bone velocity or acceleration given by users into the joint space commands for dynamics simulation. In our implementation users can give a desired target position of the bone interactively during the dynamics simulation by dragging the mouse cursor, and the user input will be converted to the target velocity or acceleration of the bone with a simple PD controller.

control of the actuators with hybrid dynamics simulation will be described in Section 3.3. The combination of these techniques make it straightforward to generate a variety of behaviors interactively, such as the jump and wave motions shown in Figure 7.

3.1 Direct Control in Cartesian Space

Similar to the inverse kinematics based pose editing techniques [22, 38], Cartesian space can give users more intuition into creating dynamic motions of characters. However, in contrast to inverse kinematics, in which the user manipulates positions or velocities directly, a dynamic simulation must be controlled through manipulating torques or accelerations on the actuating joints.

In our approach, at every simulation time step, a target acceleration on a selected bone in the skeleton is generated in the Cartesian space through a simple PD controller from user input such as mouse dragging, and the acceleration must be transformed into the joint space commands which are in the form of active joint torques or accelerations (Figure 2). The link acceleration in Cartesian space can be regarded as a kinematic function of the displacements, velocities, and accelerations of the active skeletal joints and the root joint. However, since the commands on all active joints affect the accelerations of the passive root joint via the equations of motion, we cannot rely on kinematics only for the transformation and need to additionally manipulate the dynamics equations to consider such a secondary effect. In this section we temporarily assume that the joint commands are the active joint accelerations in our formulation to avoid unnecessary complication in describing the approach, but the active joint torques can also be used as the commands in our approach with very similar formulae to that of this section (Section 3.2).

Let $\mathcal{A} \in \Re^6$ denote the linear and angular accelerations of a bone with respect to the global frame. By describing the link acceleration kinematically in terms of the displacement, velocity, and acceleration of the active and passive coordinates, and then employing the dynamics equation (1) to relate the passive and active acceleration, one can rewrite the link acceleration as

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

where $\ddot{q}_a$ denotes the acceleration of the active coordinates, $\frac{\partial \mathcal{A}}{\partial \ddot{q}_a}$ denotes the derivative of the link acceleration with respect to the accelerations of the active coordinates, and

$\mathcal{A}_0$ is the link acceleration when $\ddot{q}_a = 0$. We should mention that there is no approximation or simplification in Equation (2), i.e., the link acceleration is intrinsically linear with respect to $\ddot{q}_a$ while its coefficients $\frac{\partial \mathcal{A}}{\partial \ddot{q}_a}$ and $\mathcal{A}_0$ are nonlinear functions of system state.

When we calculate $\mathcal{A}_0$ in the constraint equations (2) we must use the equations of motion for the whole deformable body system. In particular, the passive nodes that are directly connected to the skeleton act as a pathway for external forces acting on the skeleton in the form of elastic forces, damping forces and contact forces. All of these forces affect the bone acceleration. They are determined by the current state $(q, \dot{q})$ only, and their effect appears in the term $\mathcal{A}_0$ in Equation (2).

The derivative of the link acceleration $\frac{\partial \mathcal{A}}{\partial \ddot{q}_a}$ can be obtained by differentiating the equations of motion of the skeleton system including the root joint. Since the skeleton can be an arbitrary articulated rigid body system, calculating $\frac{\partial \mathcal{A}}{\partial \ddot{q}_a}$ is not a simple problem. We employ a geometric formulation of the dynamics algorithms by Park et al. [26] in order to obtain the analytical derivatives of the equations of motions efficiently and elegantly. We present the explicit formulas for the derivatives in appendix A.3.

With a given desired acceleration for the bone $\mathcal{A}_d$, the constraints on the command inputs can be written as linear equations $\frac{\partial \mathcal{A}}{\partial \ddot{q}_a}\ddot{q}_a = \mathcal{A}_d - \mathcal{A}_0$. In general one can set up constraint equations for every bone in the same way and build up

$$Cx = d \tag{3}$$

where C and d are the row-wise collections of $\frac{\partial \mathcal{A}}{\partial \ddot{q}_a}$ and $\mathcal{A}_d - \mathcal{A}_0$ on each link respectively and $x = \ddot{q}_a$.

Besides the constraints on the link accelerations, the boundary limits of the active joint displacement and a desired target pose of the skeleton can also be incorporated to the constraint equations (3). The boundary limits can be considered with a penalty-based soft constraints. For example, in the case of exceeding the upper limit, i.e., $q_a^i > q_{a,\max}^i$, one can use $\ddot{q}_a^i = -k_p(q_a^i - q_{a,\max}^i) - k_c\dot{q}_a^i$ where $q_{a,\max}^i$ denotes the upper limit of the displacement of the i-th active coordinate, and k_p and k_c are positive scalar gains. One can also impose a desired target pose $q_{a,\mathrm{ref}}$ by setting up the constraints $\ddot{q}_a = -\gamma(t)\left(K_p(q_a - q_{a,\mathrm{ref}}) + K_c\dot{q}_a\right)$ where $q_{a,\mathrm{ref}}$ represents the given target pose, K_p and K_c are positive definite gain matrices, and $\gamma(t)$ denotes a scaling function on time increasing from 0 to 1 gradually.

Though the linear constraint equations (3) look simple, we need to be very careful when we solve the equations. In many situations C is likely to be ill-conditioned and the exact solution tends to be very large which may cause divergence in simulation. To overcome such a problem, we employed the singularity-robust inverse(SR-inverse) which has been used for inverse kinematics [38, 24]. By using the singular value decomposition and a constant $\alpha > 0$, the SR-inverse of C can be computed as $C^\ddagger = V(S^\ddagger)^T U^T$ where $C = USV^T$, $S^\ddagger = \mathrm{diag}(\sigma_i/(\sigma_i^2 + \alpha))$, and σ_i denotes the singular values of C, i.e., $S = \mathrm{diag}(\sigma_i)$. Note that $C^\ddagger$ becomes identical to the Moore-Penrose pseudoinverse when $\alpha = 0$.

A hierarchy on the constraints can be imposed by dividing them into primary and secondary constraints [38]. Let $C_p x = d_p$ and $C_s x = d_s$ be the primary and secondary constraints on the command respectively. If the number of primary constraints is less

than the number of actuators, there are an infinite number of solutions for the primary constraints and in this case, one can choose a solution that minimizes the secondary constraints error. The solution space of the primary constraints can be expressed as $x = x_0 + N_p y$ where x_0 denotes a solution for the primary constraints, N_p represents the null space of C_p, and y is a new variable for exploring the subspace. By substituting $x_0 + N_p y$ for x in the secondary constraint equations and solving the equations in terms of y in a least square sense, one can get $y = (C_s N_p)^{\ddagger}(d_s - C_s x_0)$. In spite of sacrificing the accuracy, we used the singularity-robust inverse even in selecting x_0, i.e., $x_0 = C_p^{\ddagger} d_p$ to prevent extraordinarily large commands which may cause divergence in simulation.

Once the accelerations of the active joints are determined by solving Equation (3), the accelerations of the passive coordinates including the passive root joint are calculated by hybrid dynamics and the system state at the next time step can be obtained by integrating the accelerations of the whole coordinates.

In our implementation the user can control the system by dragging a link with the mouse, and if needed, simple constraints on the other links, such as zero acceleration or zero velocity, can be added. A simple PD controller generates target accelerations for both the dragged link and the other constrained links. The default setting is that mouse drags are secondary constraints while other constraints on links such as maintaining orientation are primary, and the joint constraints such as the boundary limits and the desired target pose of the skeleton are secondary. However, this can easily be adjusted by the user, for example to make mouse drags the primary constraint or to place all constraints at the same level.

In many situations of the motion examples we have tested, the number of constraints was less than the number of actuators. In this case, an infinite number of solutions can exist theoretically, and the solution chosen in this section will be minimal actuator accelerations which satisfy the constraints or minimize the error on them. This is because we formulated the linear constraint equations (3) with respect to the actuator accelerations $\ddot{q}_a$. We will introduce a few variants of the formulation in Section 3.2, which would result in different style of motions, by reformulating the constraints with other meaningful variables such as actuator torques or even the joint velocities.

3.2 Variants of the Cartesian Control

We can choose the joint torques τ_a as the actuator command instead of the acceleration in Section 3.1, and in this case, minimal actuator torques will be obtained. The link acceleration can be rewritten as

$$\mathcal{A} = \frac{\partial \mathcal{A}}{\partial \tau_a} \tau_a + \mathcal{A}_0^a \tag{4}$$

where $\mathcal{A}_0^a$ denotes the link acceleration when $\tau_a = 0$ and the derivative of the link acceleration with respect to the actuator torques can be obtained by the formulae in Appendix A.3. The boundary limits on the torques can also be considered with a penalty-based constraints similarly to the displacement limits in Section 3.1. Once the actuator torques are determined by solving the constraint equations, the accelerations of the whole system coordinates, $\ddot{q}$, can be obtained by solving forward dynamics.

The skeletal coordinates $q_s = (q_a, q_r)$, including the coordinates on the root joint q_r, can also be chosen as the variables in the formulation of the link acceleration. The kinematic relation between the link acceleration and the accelerations of the skeletal coordinates can be written as

$$\mathcal{A} = \frac{\partial \mathcal{A}}{\partial \ddot{q}_s} \ddot{q}_s + \mathcal{A}_0^s, \tag{5}$$

and this is equivalent to the well-known relation $\mathcal{A} = \mathcal{J}\ddot{q}_s + \dot{\mathcal{J}}\dot{q}_s$ where $\mathcal{J}$ denotes the Jacobian of the link. In the case of passive root joint, since $\ddot{q}_r$ cannot be arbitrarily specified, i.e., it is determined passively from $\ddot{q}_a$ and current system state by the dynamics of the system, we need additional constraints to consider the dynamics,

$$\tau_r = \frac{\partial \tau_r}{\partial \ddot{q}_s} \ddot{q}_s + \tau_{r,0} = 0 \tag{6}$$

which means that the torque acting on the passive root joint should be zero.

Assuming an explicit single step integrator, Equations (5) and (6) can be reformulated with the skeletal joint velocities, and this results in an interesting minimum velocity solution. With the explicit Euler integration on velocity, $\ddot{q}_s = \frac{1}{h}(\dot{q}_s^d - \dot{q}_s)$ where $\dot{q}_s$ and $\dot{q}_s^d$ denote the joint velocities at the current and next time step respectively and h represents the step size, one can obtain

$$\mathcal{V} = \mathcal{J}\dot{q}_s^d \tag{7}$$

and

$$\tau_r = \frac{1}{h}\frac{\partial \tau_r}{\partial \ddot{q}_s} \dot{q}_s^d + \tau_{r,1} \tag{8}$$

where $\mathcal{V}$ denotes a given desired link velocity at the next time step and $\tau_{r,1}$ represents the torque acting on the root joint when $\ddot{q}_s = -\frac{1}{h}\dot{q}_s$. Similarly to the link acceleration, the link velocity is determined from user input such as mouse dragging through a simple controller. In our implementation, we set up the torque constraint on the root joint as the primary constraints and use the Moore-Penrose pseudoinverse to obtain the correct solution space of the constraints. Then, the other constraints such as Equation (7) or boundary limits of the joint displacements are treated as secondary or tertiary constraints in a similar manner to that described in Section 3.1. Once the solution of the equations, i.e., the minimal skeletal velocities at the next time step which satisfies the dynamics equations (8) and minimizes the errors on (7), is obtained, then the accelerations of the passive coordinates for deformable tissues are calculated by hybrid dynamics. The resulting motions generated in this way appear as well-controlled slow motions which can often be seen in apparatus gymnastics.

3.3 Prescribed Control in Joint Space

Even though the direct guidance in Cartesian space by dragging the mouse is intuitive and simple to use, imposing the path trajectory for the active joints and solving for the resulting motion through hybrid dynamics simulation is a powerful way to generate a

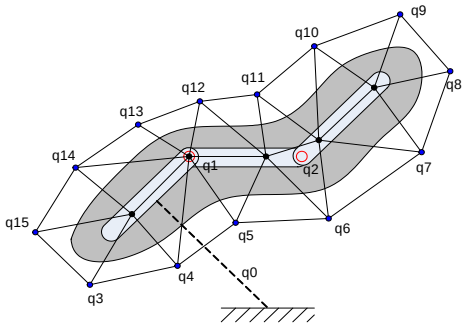

Figure 3: **A 2D illustration for skeleton-driven deformable body systems:** The shape of the deformable body is defined by the fine surface, but the elastic motions of it are represented by the coarse volumetric mesh. The deformable body is attached to the skeleton by fixing some of the mesh nodes to the bones, and the actuators in the skeleton generate the motions of the whole system. In this illustration, the independent reduced coordinates set of the system is $q = \{q_0, q_1, q_2, \cdots, q_{15}\}$ where q_0 denotes the coordinates of the passive root joint, (q_1, q_2) represents the active coordinates of the actuating joints, and the others are for the passive DOFs of the deformable body.

typical motion segment such as stretching phase in jumping. In contrast to the Cartesian controls where the global motion of selected links is guided, this approach only concerns the relative joint motions of the skeleton. Therefore the user must estimate the resulting global motion when he or she designs the path trajectories.

In our implementation the user can create keyframes by dragging a link with the mouse through an inverse kinematics approach, and this is inspired primarily by [38]. Then, the key poses are interpolated by a B-spline to generate a continuous active joint trajectory. At each time step, the displacements, velocities, and accelerations of the active joints are set up as input, and accelerations of the passive coordinates describing the root joint and the deformable tissues are calculated by hybrid dynamics. It should be noted that, in this approach, there is no reason to rely on the popular forward dynamics simulation which requires a carefully tuned servo controller. The $O(n)$ hybrid dynamics algorithm employed here produces the very motions which would be obtained through the forward dynamics simulation with an ideal perfect servo controller (Section 4.2).

4 Formulation

4.1 Skeleton-driven Deformable Body

Our skeleton-driven deformable model was inspired by the work of Capell et al. [7]. It consists of a coarse volumetric mesh with an embedded fine surface to represent a deformable body and a properly designed skeleton to derive the motion of the whole system. Unlike the work of Capell et al. [7] where they restricted the bones to lie along

edges in the volumetric mesh, we attach the deformable body to the skeleton by fixing some of the nodes to the bones. The skeleton can be an arbitrary articulated rigid body system with a tree topology and there is no restriction on the types of joints and the mass properties of the bones in the skeleton (Figure 3).

The positions of the vertices on the embedded fine surface are determined from the positions of nodes in the coarse volumetric mesh. We assume that the surface is linearly related to the mesh, i.e.,

$$y = \sum_i \phi_i x_i \tag{9}$$

where $y \in \Re^3$ and denotes the global position of a surface vertex, $x_i \in \Re^3$ and $\phi_i \in \Re$ represent the global positions of the related neighboring nodes in the volumetric mesh and their corresponding constant coefficients respectively. Even though there is no restriction on selecting the nodes, we choose the four nodes of the tetrahedron enclosing the surface vertex in the case of tetrahedral mesh. In this case, ϕ_i becomes the barycentric coordinates with respect to the four element nodes.

The continuous mass distribution of the deformable body can be approximated with a set of lumped mass particles located at each node of the volumetric mesh. As the material is filled only inside of the embedded surface, we need to calculate the mass contributed to the filled region of a volumetric mesh element by each one of its vertices. In the case of tetrahedral elements with linear shape functions, the mass contribution of one of the nodes of the element is calculated by integrating $\int_{\mathrm{V}} \rho(x)\varphi(x)d\mathrm{V}$ where $\rho(x) \in \Re$ and $\varphi(x) \in \Re$ denote the material density and the barycentric coordinate at an interior point $x \in \Re^3$. The volume integration is a time-consuming process, but one can precompute this as the mass of each particle does not change during simulation.

The elastic forces in the deformable body are obtained by a finite element approach, and they are regarded as external forces acting on the element nodes. The explicit formula for calculating the elastic forces will be discussed in section 4.3. For fast simulation, we do not consider the viscosity of the elastic material within the finite element framework. Instead, we employ an external damper to attenuate the vibration of the deformable tissues and this will be addressed in section 4.5.

We apply penalty-based contact normal forces and Coulomb friction forces to the surface vertices that collide with the ground as in the work of Faloutsos et al. [10]. Specifically, the contact force f^s on a surface vertex can be transformed to the equivalent forces f_i^v acting on the neighboring nodes in the volumetric mesh by

$$f_i^v = \phi_i f^s \tag{10}$$

where ϕ_i denotes the barycentric coordinates of the surface vertex defined in Equation (9).

4.2 Equations of Motion

The mass particles, or the nodes in the volumetric mesh of the deformable body, can be classified into fixed particles and free particles. The fixed particles denote the nodes attached on the bones and the others are the free particles which represent the motions

of the deformable body with respect to a reference rigid body which can be either a bone or the ground.

To describe the motions of the mass particles representing the deformable tissue, we employed a relative coordinate system using joints. For fixed particles attached to the bones we simply merged their mass properties into the bones, and for the other particles we connect 3-DOF translational joints to their reference rigid bodies. Though there is no restriction in selecting the reference rigid bodies in our formulation, we prefer choosing the nearest bone from each mass particle rather than the ground as we can apply damping forces directly in the translational joint coordinates (Section 4.5).

As we assume a tree-topology skeleton, the whole system including the skeleton and the mass particles is also an articulated rigid body system with tree topology. Therefore, standard $O(n)$ recursive dynamics algorithms originated from [14] can be applied to simulate the system. In the case of forward and hybrid dynamics (see section 3 for the definition of them), they consists of three recursions:

1. Forward recursion: Calculate kinematic information such as position and velocity of each link.

2. Backward recursion: Calculate the articulated body inertia and bias force of each link.

3. Forward recursion: Calculate the acceleration or torque of each joint as well as the acceleration of each link.

We follow a geometric formulation of the recursive dynamics algorithms by Park et al. [26]. Geometric recursive algorithms for forward and hybrid dynamics of open chain systems were reported in Ploen and Park [27] and Sohl and Bobrow [32] respectively where the formulations were based on 1-DOF joints. To manipulate multi-DOF joints (e.g., the root joint and the 3-DOF translational joints connecting the particles) more efficiently, we present more general form of the algorithms based on multi-DOF joints, and the detail can be found in appendix A.1.

Since the algorithm is formulated for rigid bodies connected by general types of joints, it is not efficient to apply the algorithm directly to the free mass particles because they do not have inertia and are connected by 3-DOF translational joints. In appendix A.2, we present a variant of the algorithm that has been customized for the mass particles and can be used together with the algorithm for rigid bodies presented in appendix A.1.

4.3 Elastic Force Computation

Since the elements in the volumetric mesh may usually undergo large displacement and deformation, we employed an approach based on nonlinear continuum mechanics. We follow O'Brien and Hodgins [25] for the explicit formula on the deformation gradient and Teran et al. [35] for their finite volume method in calculating the elastic forces. We also refer the reader to Bonet and Wood [5] and Basar and Weichert [3] for an introduction to nonlinear continuum mechanics.

To describe deformation of an object we need the deformation gradient tensor F which is defined as

$$F = \frac{\partial x}{\partial X} \tag{11}$$

where x and X denote the positions of a point in the object after and before deformation respectively. In the case of a tetrahedral mesh with a linear shape function, F is a constant 3×3 matrix in each tetrahedron and one can see, e.g., O'Brien and Hodgins [25] for an explicit formula of F.

As a general measure of deformation, the Lagrangian or Green strain tenor E is defined as

$$E = \frac{1}{2}(F^T F - I) \tag{12}$$

which is nonlinear in the deformation. The deformation causes internal stress inside the deformed object and the relationships between the strain and stress, known as constitutive equations, depend on the type of material.

St. Venant-Kirchhoff (or StVK) material has been widely used in the computer graphics community because it is the simplest hyperelastic material which can be applicable to geometrically nonlinear analysis. The constitutive equation for the material is defined by

$$S = \lambda(\mathrm{tr}E)I + 2\mu E \tag{13}$$

where S is the second Piola stress tensor, and λ and μ are Lamé constants. For a detailed mathematics on the StVK model, we refer the reader to Basar and Weichert [3].

The elastic forces acting on the vertices of a mesh element can be obtained from the stress. We follow the formula in Teran et al. [35] which uses the first Piola-Kirchhoff stress tensor defined by $P = FS$. The method is the same as FEM (e.g., [25]) in the case of tetrahedra with linear shape function and constant strain. Once the stress P is obtained, the elastic force on a vertex i of a tetrahedron due to the element can be written as

$$f_i = Pb_i \tag{14}$$

where $b_i = -\frac{1}{3}(A_1N_1 + A_2N_2 + A_3N_3)$, A_j are the areas of the undeformed faces incident to the vertex, and N_j denotes the normal vectors to those undeformed faces. Since A_j and N_j do not change during simulation, b_i can be precomputed. One can also save multiplications additionally by computing $f_0 = -(f_1 + f_2 + f_3)$. The total elastic force acting on a vertex is obtained by summing the forces exerted by all elements incident to the vertex.

We should note that Equation (14) is only valid when the mesh element is fully filled with the elastic material. Since the elastic force is the negative partial derivative of the elastic potential in the element with respect to the nodal position and the elastic potential is proportional to the material volume, the elastic force can be rewritten by

$$f_i = wPb_i \tag{15}$$

where w denotes the ratio of the filled region in the element to its volume, i.e., $w = \int_V \mathrm{sgn}(\rho(x))dV / \int_V dV$. The volume ratio w for each mesh element can be precomputed because it does not vary during simulation.

4.4 Selective Diagonalization

We employ the diagonalization technique of Irving et al. [19] to treat largely deformed or even inverted elements. However, unlike the work of Irving et al., where the time-consuming diagonalization must be done on every element, we diagonalize only elements that we know to have large deformations. We introduce a new technique to efficiently test for large deformations, which makes this selective digonalization possible.

Since the StVK model can survive large displacement in spite of its simple calculation, the model has been widely used in the computer graphics community. However, the material has also a fatal drawback, i.e., the stiffness gets weaker as the compression increases, so it cannot be used when the element undergoes large deformation. If the deformation passes beyond a certain point, then the stress begins to decrease and finally gets to zero when the element becomes flat. Moreover, if the element is inverted, now the elastic force acts to keep the element inverted [19].

To overcome such a problem Irving et al. [19] modified the constitutive model via diagonalization of the deformation gradient F to obtain valid restoring forces when the compression is larger than a certain point or the element is inverted. Suppose that F can be diagonalized as

$$F = U\hat{F}V^T \tag{16}$$

where U and V are pure rotations, i.e., $U^TU = V^TV = I$ and $\det U = \det V = 1$. Then, since the first Piola-Kirchhoff stress tensor P is invariant under rotations of material space as well as under rigid body rotations for isotropic materials, $P = P(F)$ can be rewritten as

$$P = U\hat{P}V^T \tag{17}$$

where $\hat{P} = P(\hat{F})$. Therefore, once $\hat{P}$ is determined from the diagonalized $\hat{F}$, one can rebuild the stress tensor from Equation (17). For example, in the case of the StVK material, one can modify $\hat{P}$ outside of a given strain boundary by linearizing it to obtain valid forces for the elements with large deformations and even for inverted elements.

In order to determine whether an element undergoes large deformation or the element is inverted, we need information on the diagonalized $\hat{F}$ which can be obtained by executing SVD(or solving the eigenproblem $F^TFv = \lambda v$) and then post-processing to make U and V become pure rotations [19]. However, even though F is just a 3×3 matrix, executing SVD or an eigensolver and the post-processing on every element for every simulation time step is computationally expensive.

In our approach we only check the range of solutions of the characteristic equation $\det(F^TF - \lambda I) = 0$ and the sign of $\det F$ to see whether the element undergoes large deformation or not. The characteristic equation results in a cubic polynomial $f(\lambda) = \lambda^3 + a\lambda^2 + b\lambda + c = 0$ and its solutions are the square of the entries of $\hat{F}$. In order to ensure that all three solutions of the cubic polynomial equation are located within a given range $[\lambda_L, \lambda_U]$ where $0 < \lambda_L < \lambda_U$, the polynomial should satisfy

$$f(\lambda_L) < 0, \quad f(\lambda_U) > 0, \quad \lambda_L < \alpha, \beta < \lambda_U \tag{18}$$

where α and β are the solutions of $f'(\lambda) = 3\lambda^2 + 2a\lambda + b = 0$ (See Figure 4). Moreover, if

$$\det F > 0, \tag{19}$$

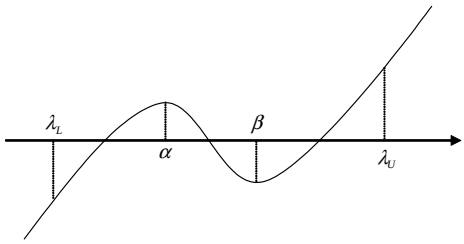

Figure 4: **A cubic polynomial**

which means that the element is not inverted, then all the entries of $\hat{F}$ are positive. Therefore all three entries of $\hat{F}$ are located within the range $[\sqrt{\lambda_L}, \sqrt{\lambda_U}]$ if and only if both Equations (18) and (19) are satisfied. With given F and $C = F^T F$, our determination process requires only 37 multiplications in our implementation.

For every element we execute our determination process, and then we follow the diagonalization technique by Irving et al. [19] for the largely deformed elements only. In the worst case, i.e., when every element is either deformed largely or inverted, our approach could be inefficient for its added determination process. However, in most normal motions of characters, only some part of the entire deformable bodies undergoes large deformation. For example, the average ratio of the number of largely deformed or inverted mesh elements to that of total elements during the jump in Figure 1(lower) is 0.33, a figure that of course depends on various parameters such as (λ_L, λ_U), the skeletal pose, the material properties, and the contact with the ground. The ratio affects the speed of dynamics simulation (Figure 5), and in our experiments, this step alone usually results in a speedup of a factor of up to 2 in the time required for the entire dynamics simulation (Table 1).

4.5 Damping Forces

The masses of the nodes which represent deformable tissues can vary within the same volumetric element, as they are dependent on the embedded surface as well as the shape of mesh elements. If an element includes a small patch of the surface and one of the element nodes is located far outside of the surface, the nodal mass computed from the volume integration will be relatively very small compared to the other nodes whose elements are located fully inside of the surface. Such a difference between the particle masses seems to be quite reasonable in order to capture the original mass distribution of the deformable body. However, if we want to calculate the damping force on the node with a finite element approach to attenuate resulting vibration due to the elastic forces, the small mass can cause a severe simulation problem. Usually we need to set up a very small step size for explicit integrators to satisfy the stringent quadratic step size restriction for the damping force [6].

In our work we ignored the viscosity of the material within the finite element framework. Instead, we add an external damper to each of the free mass particles in the deformable body. As we do not want the gross rigid body motions of the skeleton to

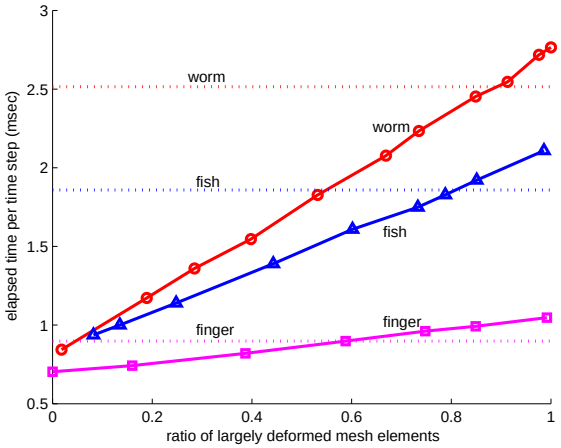

Figure 5: **Elapsed time for hybrid dynamics per time step** with respect to the **ratio of the number of largely deformed mesh elements** to that of total elements is shown here. The solid lines with red circles, blue triangles, and magenta squares denote the elapsed times to calculate hybrid dynamics with our new selective diagonalization method for the worm, fish, and finger models shown in Figure 6 respectively. The dotted lines in red, blue, and magenta show the elapsed times with the previous method by [19] for the same models. They do not depend on the ratio. The figure shows that our selective diagonalization method is more efficient than the previous method when the system has a small ratio. If the ratio exceeds a limit, such as 0.9(worm), 0.8(fish), and 0.6(finger), then our method loses the merit of its speed because of the added checking process. Our method is more effective on systems with large number of mesh elements, and the numbers of elements of the models here are 714(worm), 415(fish), and 119(finger) respectively (Table 2).

affect the damping force, we connect the other end of the damper to the nearest bone. The damping force f_d is defined by

$$f_d = c\dot{x} \tag{20}$$

where c is a damping coefficient which is proportional to the nodal mass and $\dot{x}$ is the relative velocity of the node with respect to the bone. In our implementation the mass particle is connected by a 3-DOF translational joint to its nearest bone, and in this case, the nodal velocity in the bone frame is updated during the recursive dynamics algorithm so that we do not need any additional computation for this damper model. By introducing the external damper on each node, we can achieve effective damping without significantly reducing the time step size.

Motion	Avg. ratio	Speedup	
Worm jump	0.33	1.76	
Worm jump 2	0.29	1.85	
Worm waving	0.18	2.21	
Worm bending	0.18	2.21	Figure 7
Worm limping	0.30	1.83	
Worm rolling	0.47	1.43	
Fish escaping	0.45	1.33	Figure 8
Finger manipulation	0.11	1.23	Figure 9

Table 1: **Average ratios** of the number of largely deformed mesh elements to that of total elements are shown for the motions in Figure 7, 8, and 9. The average ratios were obtained from the entire motion sequences corresponding to the filmstrips. The resulting **speedups** in the dynamics simulation by our selective diagonalization method were estimated using the data in Figure 5. Note that the real speedup in creating the character motions would be lower than this because the motions were generated mostly by using the Cartesian controls which require to calculate the derivative of the skeletal dynamics in addition to the dynamics simulation.

Models		Worm	Fish	Finger
DOF		543	258	82
Skeleton	Bones	7	7	5
	Actuators	6	6	7
Volumetric mesh	Nodes	224	107	52
	Tetrahedra	714	415	119
Surface mesh	Vertices	262	958	157
	Triangles	520	1912	276
Avg. calc. time (msec)	Joint	1.60	1.48	0.629
	Cartesian	2.86	2.44	2.10
step size (msec)		1	1	0.5

Table 2: **The numbers for each models: (Average calculation time per time step)** In order to measure the average calculation time, we created one or two representative motions for each model using Cartesian control. We extracted the joint trajectories of the motions and ran the hybrid simulation with the trajectories to measure the calculation speed of the joint space control. We used a jump and rolling motions for the worm model, a jump motion for the fish model, and a short finger manipulation of an object for the finger model. **(Step size)** For the worm and finger model, we always used 1 msec and 0.5 msec respectively as the step sizes for mixed Euler integration. For the fish model, we used 1 msec for most part of the simulation. However, sometimes we decreased the step size temporarily to prevent the simulation from diverging especially when the character collides with the ground at high speed.

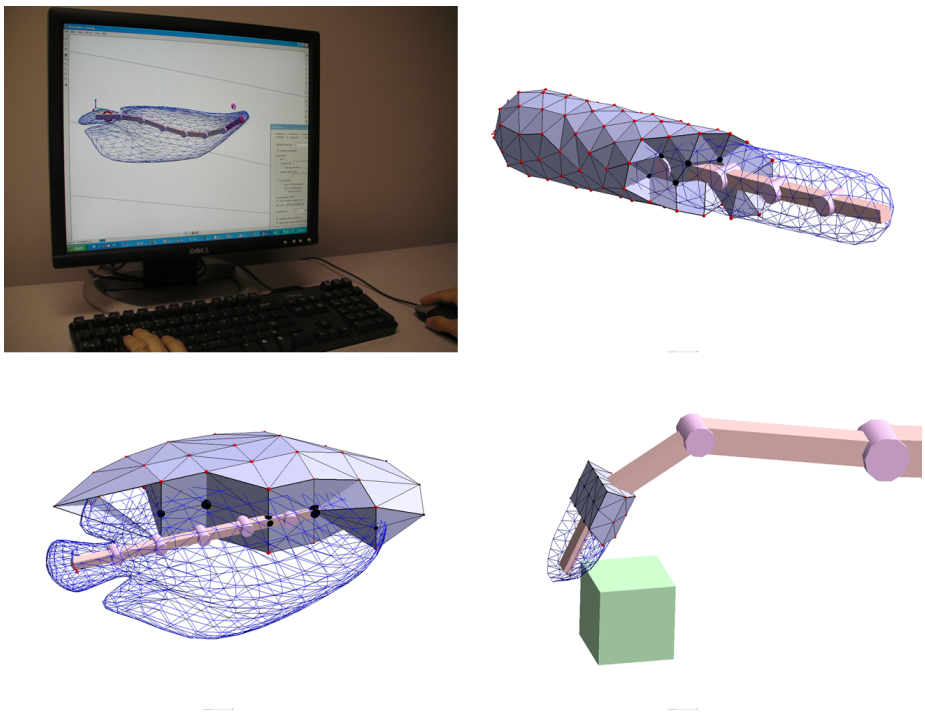

Figure 6: **The user interface(upper left) and test models:** We have tested our algorithms with a worm**(upper right)**, a fish**(lower left)**, and a finger model**(lower right)**. All the models have three dimensional deformable bodies attached to their skeletons. The wire frames are the fine surfaces of the deformable bodies, and the solid meshes represent the enclosing coarse volumetric meshes for FEM. The nodes fixed to bones are depicted as black spheres while the others are red ones, and the node size is proportional to their masses.

5 Results

We have tested our algorithms by creating various motions with the skeleton-driven deformable body systems such as a worm, a fish, and a finger models shown in Figure 6. More information on the models is presented in Table 2. All of the tested models are three dimensional, but we set the root joints of them to allow only planar motions because of the limitation of our two dimensional mouse dragging interface. However, it should be emphasized that we are running the fully three dimensional dynamics algorithms in the simulation and our simulation control algorithms have also been formulated based on fully three dimensional models.

We used the mixed Euler integrator – it is also called the symplectic Euler method [34] – for its simplicity and good stability, and set up the step size for integration with 1 msec for the worm and fish models in most cases, and 0.5 msec for the finger model. For the selective diagonalization we used $\lambda_L = 2/3$ and $\lambda_U = 5/3$ which correspond to $k_0/2$ and $2k_0$ respectively where k_0 denotes the material stiffness when there is no

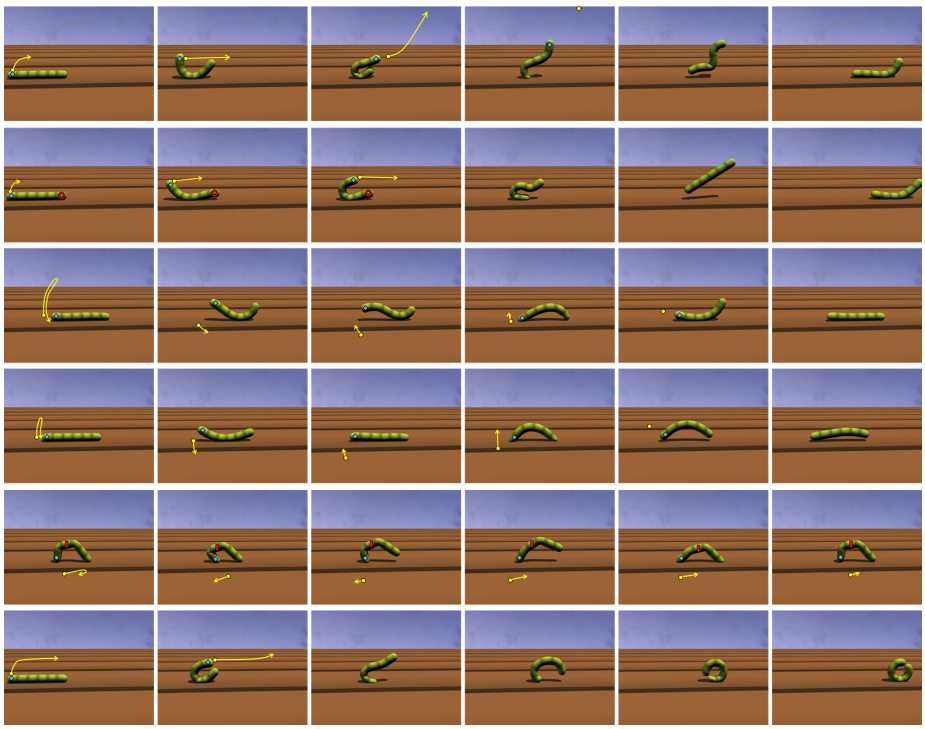

Figure 7: **Motions of the worm character:** All the motions are created by mixing the Cartesian space controls and joint space controls in our user interface system. We added illustrations to show the user inputs during the Cartesian controls on the filmstrips. The yellow circles and arrow lines denote the current position of the mouse cursor and its trajectory from the previous frame to the current frame respectively, and the blue diamonds represent the selected bones to be guided by the mouse drag. The red circles and rectangles shows the constraints imposed on bones to maintain their orientations or positions.

deformation, i.e., $F = I$.

All the example motions and benchmarks were tested on a desktop machine with a 2.66GHz Intel Core 2 processor and 2GB of RAM. We show the average elapsed times per single time step for the simulations through the Cartesian controls and the prescribed joint commands in Table 2. In our experiment the calculation speed of the guided simulation with the Cartesian controls is around two times slower than that of simulation with joint commands, and this is because for the Cartesian controls we need to calculate the derivatives of the equations of motion with respect to the command inputs of the actuators in addition to solving the dynamics equations.

Various kinds of motions of the worm, fish, and finger model were generated by mixing the Cartesian and joint controls of the simulation described in Section 3 and their filmstrips are shown in Figure 7, 8, and 9. Our user interface system (Figure 6) allows the user to trace back to any time in the past during the simulation and resume

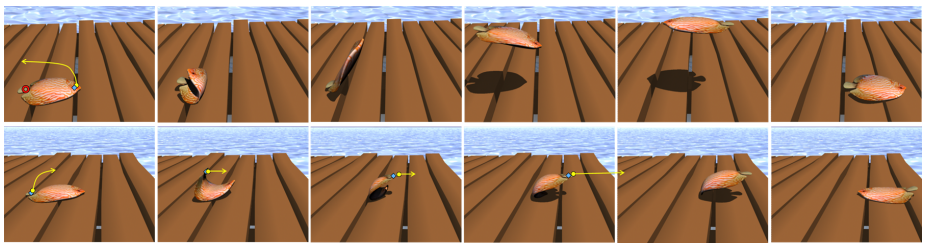

Figure 8: **Motions of the fish character:** The fish character is trying to escape from the dock and these are two short segments of the long sequence of motion. **(Upper)** The jump motion was generated by using the acceleration-minimizing Cartesian control initially and then switching to the prescribed joint control with the stretched skeletal pose as the keyframe to take off from the floor. **(Bottom)** The turnover was guided with the velocity-minimizing Cartesian control, and the resulting motion is relatively slow compared to other dynamic motions.

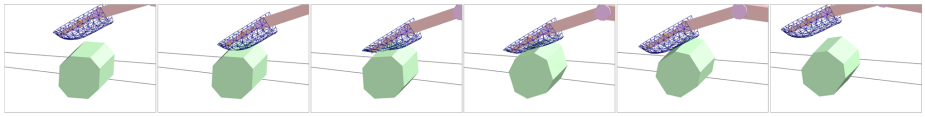

Figure 9: **Finger manipulation of an object:** This is a very short segment of a finger manipulation of the object. The motion was generated by using the velocity-minimizing Cartesian control. The user clicked the bone under the fingertip with the mouse and then simply dragged it at will while observing the resulting motion of the object and the deformation of the soft fingertip. The object was modeled as a rigid body with 6-DOF passive root joint, and unlike to the other models such as the worm and fish characters, the root joint of the finger skeleton was active.

generating motions from that point seamlessly, and this helps the user to create motions effectively and efficiently with a trial and error approach.

Figure 7 shows the worm motions created in our experiments. We added an illustration on them to show the user inputs used in Cartesian controls such as a desired target position(the yellow circles) of the selected bone(the blue diamonds) and the constraints imposed on other bones for maintaining orientation(the red hollow circles) and position(the red rectangles for vertical position). The yellow arrow lines show the trajectory of the target position from the current to the next frame, and this corresponds to the mouse drag trajectory during the simulation guide by user. In our experiment it took less than one or two minute(s) for a well-trained user to create each of the worm motions using our user interface system.

The first row in Figure 7 shows a jump of the worm which was driven by the direct Cartesian control described in Section 3.1 with mouse dragging. The user guided the target position of the very left bone by moving mouse cursor, and the character followed the guidance by actuating its internal active joints and using the contact while respecting the laws of physics. For the landing we switched to forward simulation with no torque on actuators, and in this case, we added a spring-damper on each actuator to

get a soft landing without unnatural vibration.

The second row in Figure 7 shows another jump of the worm which was initially driven by the Cartesian control before the takeoff and then switched to the keyframe control mode in Section 3.3 for stretching. We set up a constraint on the very right bone to maintain its orientation when we guide the very left bone with mouse dragging. As the prescribed joint control with the keyframe only concerns the relative active joint trajectory, the user must estimate the resulting global motion in designing the keyframe and the timing. For the landing we again switched to the forward simulation as in the jump above.

The third row in Figure 7 shows a wave motion of the worm driven by the torque-minimizing variant of the Cartesian control. The motion was created by dragging the mouse cursor on the very left bone up and down repeatedly, and the resulting waving motion looks quite reasonable in the sense that it would minimize the actuator torques while tracing the user input.

For comparison purposes, in the fourth row of Figure 7, we present the motion that would result from acceleration-minimizing Cartesian control with user input similar to that for the wave motion above. In this case the worm tried to follow the user guidance by bending its body up and down, and this style of motion will reduce the accelerations of the active joints compared to the torque minimizing waving motion.

The fifth row in Figure 7 shows a limping motion. This motion was generated with the velocity-minimizing Cartesian control in Section 3.2. We set up a constraint on the middle bone maintaining the vertical position of the bone to help the character not to fall down, and guided the position of the very left bone by dragging the mouse cursor right and left repeatedly. In order to get the asymmetrical pose and motion, we imposed different weightings on the joints in solving the linear constraint equations.

The last row in Figure 7 shows a rolling motion of the worm. It was generated by initially guiding the position of the very left bone with the acceleration-minimizing Cartesian control and then switching to the prescribed joint control using a keyframe. The process was similar to that of the jump motion in the first row of Figure 7, but we needed more patience to find a good switching timing by trial and error.

Figure 8 shows two short segments of a long motion sequence of a fish trying to escape from the dock. The short jump motion in the upper row was generated with a similar process as the worm jump in the first row of Figure 7. We designed very quick takeoff to obtain a high jump, so that we needed to change a few simulation parameters, such as the damping coefficient, step size, and the contact parameters, to treat high speed contacts at the moment of the takeoff and the resulting high speed landing. Though the simulation speed would become slow during the periods of high speed contacts due to the reduced step size, this did not affect the entire time spent to create the jump motion so much because, in most of the simulation time, we could still use the regular step size for our simulation. Moreover, in general, the portion of the pure calculation time in the entire process for creating such an animation is small. In our experience, a significant portion of the time was spent in designing or determining the way of movement and manipulating the user interface for setting up constraints and adjusting various parameters. The slow turnover motion shown in the bottom row was obtained by using the velocity-minimizing Cartesian control. The motion was generated by simply dragging the mouse slowly along the path shown in the figure

after clicking the tail of the fish.

Figure 9 shows a very short segment of a finger manipulation with the deformable fingertip. The object was modeled as a rigid body with six dimensional passive root joint, and the contact between the fingertip and the object was considered with the penalty based method described in Section 4.1. In this example the trajectory of the bone where the fingertip was attached was guided by the velocity-minimizing Cartesian control to manipulate the object interactively. Unlike the other characters, we assume that the finger model has active root joint, so that we do not need to consider the constraints on the root joint torques, Equation (6) in Section 3.2. At every time step, the skeletal joint commands are calculated to follow the target trajectory of the bone, and the motions of the passive elements, such as the tissues in the fingertip and the rigid object, are obtained by solving the equations of motions of the entire system with the hybrid dynamics algorithm described in 4.2.

6 Conclusion

We have introduced a novel technique to control the dynamic simulation of skeleton-driven deformable characters. In our approach users can guide the simulation interactively in Cartesian space which is more familiar and intuitive to use. The target accelerations or velocities of selected bones are generated from user input in Cartesian space, and our transformation converts them into actuator commands. With a few variants of the Cartesian control, one can also guide the simulation in different styles.

In order to make the approach intuitive, and also to facilitate a "trial and error" approach in creating dynamic motions of a character, we solve the fully nonlinear equations of motion at interactive rates by applying an efficient recursive algorithm for calculating the dynamic equations and their derivatives, and introducing the selective diagonalization technique. In our formulation, the computational costs for both the Cartesian space and joint space controls of dynamics simulation are linear in the number of mesh elements of the deformable body with same skeleton.

One limitation of our approach is that there is some learning time required to train a user with the system and build intuition about the physical responses of a model. The interaction speed across different processor speeds also must be preserved to take advantage of the training.

In our current implementation, only the position of the mouse cursor in 2D screen is used as an interactive input for the Cartesian controls, and because of this, sometimes it would be difficult to guide the character motions in a desired way. For example, when we generated the rolling motion of the worm in Figure 7, it was difficult to find a good timing for switching from the Cartesian guiding to the keyframe based joint control to make the character roll on the floor. If the orientation of the selected bone could be guided directly in addition to the positional input, then generating such a motion could be much easier. In general, though our Cartesian controls with low dimensional input provides an intuitive way of creating dynamic motions of characters, sometimes undesired or unfamiliar motions can be generated and these are mainly caused by the redundancy in solving the constraints given by users to obtain corresponding joint commands for dynamics simulation. One possible area of future work would be constraining the

space of joint commands using an example database. Increasing the dimension of user control space by using, e.g., a motion capture system or multi-input touch screen, may be also an interesting topic for future work.

Acknowledgements

References

[1] W. W. Armstrong and M. Green. The dynamics of articulated rigid bodies for purposes of animation. In *Proceedings of Graphics Interface*, pages 407–415, 1985.

[2] Jernej Barbič and Doug L. James. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Transactions on Graphics (SIGGRAPH 2005)*, 24(3):982–990, August 2005.

[3] Yavuz Basar and Dieter Weichert. *Nonlinear Continuum Mechanics of Solids*. Springer, 2000.

[4] Marcel Bergerman and Yangsheng Xu. Robust joint and cartesian control of under-actuated manipulator systems. *ASME Journal of Dynamic Systems, Measurement, and Control*, 118(3):557–565, 1996.

[5] Javier Bonet and Richard D. Wood. *Nonlinear continuum mechanics for finite element analysis*. Cambridge University Press, 1997.

[6] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[7] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive skeleton-driven dynamic deformations. *ACM Trans. Graph.*, 21(3):586–593, 2002.

[8] Michael F. Cohen. Interactive spacetime control for animation. *SIGGRAPH Comput. Graph.*, 26(2):293–302, 1992.

[9] Kevin G. Der, Robert W. Sumner, and Jovan Popović. Inverse kinematics for reduced deformable models. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1174–1179, New York, NY, USA, 2006. ACM.

[10] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Dynamic free-form deformations for animation synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):201–214, 1997.

[11] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 251–260, New York, NY, USA, 2001. ACM.

[12] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. The virtual stuntman: dynamic characters with a repertoire of autonomous motor skills. *Computers & Graphics*, 25(6):933–953, 2001.

[13] Anthony C. Fang and Nancy S. Pollard. Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):417–426, July 2003.

[14] R. Featherstone. The calculation of robot dynamics using articulated-body inertias. *The International Journal of Robotics Research*, 2(1):13–30, 1983.

[15] Nico Galoppo, Miguel A. Otaduy, Serhat Tekin, Markus Gross, and Ming C. Lin. Soft articulated characters with fast contact handling. *Computer Graphics Forum (Proc. of Eurographics 2007)*, Vol.26(3), 2007.

[16] Michael Gleicher. Retargetting motion to new characters. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 33–42, New York, NY, USA, 1998. ACM.

[17] Radek Grzeszczuk and Demetri Terzopoulos. Automated learning of muscle-actuated locomotion through control abstraction. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 63–70, New York, NY, USA, 1995. ACM.

[18] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. In *Proceedings of SIGGRAPH 95*, pages 71–78, August 1995.

[19] G. Irving, J. Teran, and R. Fedkiw. Invertible finite elements for robust simulation of large deformation. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 131–140, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[20] Evangelos Kokkevis, Dimitri Metaxas, and Norman I. Badler. User-controlled physics-based animation for articulated figures. In *CA '96: Proceedings of the Computer Animation*, page 16, Washington, DC, USA, 1996. IEEE Computer Society.

[21] Joseph Laszlo, Michiel van de Panne, and Eugene Fiume. Interactive control for physically-based animation. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 201–208, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[22] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 39–48, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[23] Richard M. Murray, Zexiang Li, , and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[24] Yoshihiko Nakamura and H. Hanafusa. Inverse kinematic solutions with singularity robustness for robot manipulator control. *Journal of Dynamic Systems, Meas., and Control*, 108:163–171, 1986.

[25] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. In *Proceedings of SIGGRAPH 99*, pages 137–146, August 1999.

[26] F. C. Park, J. E. Bobrow, and S. R. Ploen. A lie group formulation of robot dynamics. *Int. J. Rob. Res.*, 14(6):609–618, 1995.

[27] S. R. Ploen and F. C. Park. Coordinate-invariant algorithms for robot dynamics. *IEEE Transactions on Robotics and Automation*, 15(6):1130–1135, 1999.

[28] Jovan Popović, Steven M. Seitz, and Michael Erdmann. Motion sketching for control of rigid-body simulations. *ACM Trans. Graph.*, 22(4):1034–1054, 2003.

[29] Zoran Popović and Andrew Witkin. Physically based motion transformation. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 11–20, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[30] G. Rodriguez, K. Kreutz-Delgado, and A. Jain. A spatial operator algebra for manipulator modeling and control. *Int. J. Rob. Res.*, 10(4):371–381, 1991.

[31] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph.*, 20(4):151–160, 1986.

[32] Garett A. Sohl and James E. Bobrow. A recursive multibody dynamics and sensitivity algorithm for branched kinematic chains. *Journal of Dynamic Systems, Measurement, and Control*, 123(3):391–399, 2001.

[33] Kwang Won Sok, Manmyung Kim, and Jehee Lee. Simulating biped behaviors from human motion data. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 107, New York, NY, USA, 2007. ACM.

[34] Ari Stern and Mathieu Desbrun. Discrete geometric mechanics for variational integrators. *Chapter in ACM SIGGRAPH'06 Course Notes on Discrete Differential Geometry*, 2006.

[35] J. Teran, S. Blemker, V. Ng Thow Hing, and R. Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 68–74, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[36] Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: physics, locomotion, perception, behavior. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 43–50, New York, NY, USA, 1994. ACM.

[37] Michiel van de Panne and Alexis Lamouret. Guided optimization for balanced locomotion. In Dimitri Terzopoulos and Daniel Thalmann, editors, *Computer Animation and Simulation '95*, pages 165–177. Springer-Verlag, 1995.

[38] Katsu Yamane and Yoshihiko Nakamura. Natural motion animation through constraining and deconstraining at will. *IEEE Transactions on Visualization and Computer Graphics*, 09(3):352–360, 2003.

[39] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: simple biped locomotion control. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 105, New York, NY, USA, 2007. ACM.

[40] J. Zhao and N. Badler. Real time inverse kinematics with joint limits and spatial constraints. *ACM Transactions on Graphics*, 1995.

[41] Peng Zhao and Michiel van de Panne. User interfaces for interactive control of physics-based 3d characters. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 87–94, New York, NY, USA, 2005. ACM.

[42] Victor Brian Zordan and Jessica K. Hodgins. Motion capture-driven simulations that hit and react. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 89–96, New York, NY, USA, 2002. ACM.

A Appendix

A.1 Recursive Hybrid Dynamics

We consider articulated rigid body systems with tree-topology, and in this case, each body is connected to its parent body by a joint. The joints in the system can be either active or passive. In the case of passive joints such as the root joint or the translational joints connecting the mass particles, the torques are always given or known and the accelerations need to be obtained by solving the equations of motion. On the other hand, the active joints can be actuated in the simulation either by applying torques or by prescribing accelerations on them. If all the active joints are actuated by applying torques, then forward dynamics algorithm can be applied to solve the accelerations of the active and passive joints. If any of the active joints are derived by prescribed trajectories, then hybrid dynamics algorithm is needed to obtain the torques on the prescribed joints and the accelerations of the other joints. Since forward dynamics can be regarded as a special case of hybrid dynamics when none of the joints are prescribed, only a recursive hybrid dynamics algorithm will be presented in this section.

while (forward recursion) {
$T_{\lambda(i),i} = \text{function of } q_i$
$V_i = \mathrm{Ad}_{T^{-1}_{\lambda(i),i}} V_{\lambda(i)} + S_i \dot{q}_i$
$\eta_i = \mathrm{ad}_{V_i} S_i \dot{q}_i + \dot{S}_i \dot{q}_i$
}
while (backward recursion) {
$\hat{J}_i = J_i + \sum_{k \in \mu(i)} \mathrm{Ad}^*_{T^{-1}_{i,k}} \Pi_k \mathrm{Ad}_{T^{-1}_{i,k}}$
$B_i = -\mathrm{ad}^*_{V_i} J_i V_i - F_i^{\mathrm{ext}} + \sum_{k \in \mu(i)} \mathrm{Ad}^*_{T^{-1}_{i,k}} \beta_k$
if ($i \in \mathcal{P}$) {
$\Pi_i = \hat{J}_i$
$\beta_i = B_i + \hat{J}_i (\eta_i + S_i \ddot{q}_i)$
} else {
$\Psi_i = (S_i^{\mathrm{T}} \hat{J}_i S_i)^{-1}$
$\Pi_i = \hat{J}_i - \hat{J}_i S_i \Psi_i S_i^{\mathrm{T}} \hat{J}_i$
$\beta_i = B_i + \hat{J}_i \left\{ \eta_i + S_i \Psi_i \left(\tau_i - S_i^{\mathrm{T}} \left(\hat{J}_i \eta_i + B_i \right) \right) \right\}$
}
}
while (forward recursion) {
if ($i \in \mathcal{P}$) {
$\dot{V}_i = \mathrm{Ad}_{T^{-1}_{\lambda(i),i}} \dot{V}_{\lambda(i)} + S_i \ddot{q}_i + \eta_i$
$F_i = \hat{J}_i \dot{V}_i + B_i$
$\tau_i = S_i^{\mathrm{T}} F_i$
} else {
$\ddot{q}_i = \Psi_i \{ \tau_i - S_i^{\mathrm{T}} \hat{J}_i (\mathrm{Ad}_{T^{-1}_{\lambda(i),i}} \dot{V}_{\lambda(i)} + \eta_i) - S_i^{\mathrm{T}} B_i \}$
$\dot{V}_i = \mathrm{Ad}_{T^{-1}_{\lambda(i),i}} \dot{V}_{\lambda(i)} + S_i \ddot{q}_i + \eta_i$
$F_i = \hat{J}_i \dot{V}_i + B_i$
}
}

Table 3: **Recursive Hybrid Dynamics**

A geometric formulation of recursive hybrid dynamics for articulated multibody systems with tree topology is shown in Table 3, and the meaning of symbols used in the algorithm is summarized in Table 4. Unlike the 1-DOF joint based formula by Sohl and Bobrow [32], the algorithm presented here, which has been formulated based on multi-DOF joints, can be applied simply and efficiently to the system containing multi-DOF joints without introducing dummy links to fold multiple 1-DOF joints. We also refer the reader to [23] for an introduction to the geometric concepts such as generalized velocities and forces, and their transformation rules. In order to consider the gravity effect, set $\dot{V}_{\mathrm{ground}} = \left(\begin{smallmatrix} 0 \\ -g \end{smallmatrix} \right)$ where $g \in \Re^3$ denotes the magnitude and direction of gravity.

symbol	description	type
$\mathcal{P}$	Index set of prescribed coordinates	
$\lambda(i)$	Index of the parent body of body i	
$\mu(i)$	Index set of the child bodies of body i	
$\{i\}$	Coordinate frame attached to body i	
joint i	Joint connecting body i and its parent body	
q_i	Coordinates of joint i	$\Re^{n_i}$
τ_i	Torque(or force) acting on joint i	$\Re^{n_i}$
$T_{\lambda(i),i}$	Homogeneous transform from $\{\lambda(i)\}$ to $\{i\}$	$SE(3)$
V_i	Generalized velocity of body i, viewed in $\{i\}$	$se(3)$ or $\Re^6$
$\dot{V}_i$	Component-wise time derivative of V_i	$se(3)$ or $\Re^6$
S_i	Jacobian of $T_{\lambda(i),i}$, i.e., $T^{-1}_{\lambda(i),i}\dot{T}_{\lambda(i),i} = S_i\dot{q}_i$	$\Re^{6\times n_i}$
J_i	Generalized inertia of body i, viewed in $\{i\}$	$\Re^{6\times 6}$
F_i	Generalized force transmitted to body i from its parent through the connecting joint i, viewed in $\{i\}$	$dse(3)$ or $\Re^6$
F_i^{ext}	Generalized external force acting on body i from environment, viewed in $\{i\}$	$dse(3)$ or $\Re^6$
$\hat{J}_i$, B_i	Articulated inertia of body i and corresponding bias force	$\Re^{6\times 6}$, $\Re^6$
η_i, β_i	Temporary variables for calculation	$\Re^6$, $\Re^6$
Ad_T	$\text{Ad}_T = \begin{bmatrix} R & 0 \\ [p]R & R \end{bmatrix}$ where $T = (R,p) \in SE(3)$, $R \in SO(3)$, $p \in \Re^3$ and $[\cdot]$ denotes the skew-symmetric matrix representation of a 3-dimensional vector.	$\Re^{6\times 6}$
ad_V	$\text{ad}_V = \begin{bmatrix} [w] & 0 \\ [v] & [w] \end{bmatrix}$ where $V = (w,v) \in se(3)$ or $\Re^6$	$\Re^{6\times 6}$
Ad_T^*	$\text{Ad}_T^* = (\text{Ad}_T)^{\text{T}}$	$\Re^{6\times 6}$
ad_V^*	$\text{ad}_V^* = (\text{ad}_V)^{\text{T}}$	$\Re^{6\times 6}$

Table 4: **Description of symbols**

A.2 Recursive Dynamics for Mass Particles

In this section we present a variant of the recursive algorithm for rigid bodies which is customized for the free mass particles of the deformable tissues. The mass particles are connected to a bone in the skeleton by passive 3-DOF translational joints, and they have no child body. We assume that a local coordinate frame is attached to each particle and its orientation is aligned to the coordinate frame of the reference bone.

The algorithm for recursive dynamics of mass particles is presented in Table 5 where m_i is the mass of particle i, f_i^{ext} represents an external force acting on the particle described in the local frame, $V_i = (w_i, v_i)$ denotes the generalized velocity of the

while (forward recursion) {
$p_i = p_i^0 + q_i$
$V_i = \begin{pmatrix} w_i \\ v_i \end{pmatrix} = \begin{pmatrix} w_{\lambda(i)} \\ v_{\lambda(i)} + w_{\lambda(i)} \times p_i + \dot{q}_i \end{pmatrix}$
}
while (backward recursion) {
$\Pi_i = 0$
$\beta_i = \begin{pmatrix} 0 \\ \tau_i \end{pmatrix}$
}
while (forward recursion) {
$\ddot{q}_i = \frac{1}{m_i}(\tau_i + f_i^{\text{ext}}) - w_i \times (v_i + \dot{q}_i) - \dot{v}_{\lambda(i)} - \dot{w}_{\lambda(i)} \times p_i$
}

Table 5: **Recursive Dynamics for Mass Particles**

particle, $V_{\lambda(i)} = (w_{\lambda(i)}, v_{\lambda(i)})$ and $\dot{V}_{\lambda(i)} = (\dot{w}_{\lambda(i)}, \dot{v}_{\lambda(i)})$ are the generalized velocity of the particle's reference bone and its component-wise time derivative respectively, and I denotes the 3×3 identity matrix. Though the last equation for obtaining $\ddot{q}_i$ in the algorithm seems to be somewhat awkward, it is exactly equivalent to the Newton's second law $f_i^{\text{g}} = m_i a_i^{\text{g}}$ where f_i^{g} denotes the net force acting on the particle described in the global frame and a_i^{g} is the global acceleration of the particle.

A.3 Derivatives of Link Acceleration

The generalized velocity of link i, $V_i = (w_i, v_i) \in$ se(3) or $\Re^6$ in Table 3, means the angular and linear velocity of the link relative to the global frame, but viewed in the body frame $\{i\}$. Therefore the global linear and angular velocity can be written as $(R_i w_i, R_i v_i)$ where $R_i \in$ SO(3) represents the relative orientation of the body frame with respect to the global frame. By differentiating the global velocity in time and using $R_i^T \dot{R}_i = [w_i]$ where $[\cdot]$ denotes the skew-symmetric matrix representation of a 3-dimensional vector, one can get the global linear and angular accelerations of link i as

$$\mathcal{A}_i = (R_i \dot{w}_i, R_i(w_i \times v_i) + R_i \dot{v}_i). \tag{21}$$

Let $z \in \Re$ be either the torque or the acceleration of a coordinate $q_j \in \Re$. Then, since R_i, w_i and v_i are not functions of z, the derivative of the global acceleration with respect to z can be written as

$$\frac{\partial \mathcal{A}_i}{\partial z} = \left(R_i \frac{\partial \dot{w}_i}{\partial z}, R_i \frac{\partial \dot{v}_i}{\partial z} \right) \tag{22}$$

where $\frac{\partial \dot{V}_i}{\partial z} = \left(\frac{\partial \dot{w}_i}{\partial z}, \frac{\partial \dot{v}_i}{\partial z} \right)$ can be obtained by differentiating the recursive dynamics algorithm. The derivatives of the recursive dynamics with respect to the torque(τ_j) and acceleration($\ddot{q}_j$) of the coordinate are shown in Table 6.

In case of $z = \tau_j$:

while (backward recursion) {
  $\frac{\partial B_i}{\partial z} = \sum_{k \in \mu(i)} \mathrm{Ad}^*_{T^{-1}_{i,k}} \frac{\partial \beta_k}{\partial z}$
  if ($i \in \mathcal{P}$) {
    $\frac{\partial \beta_i}{\partial z} = \frac{\partial B_i}{\partial z}$
  } else {
    $\frac{\partial \beta_i}{\partial z} = \frac{\partial B_i}{\partial z} + \hat{J} S_i \Psi_i \quad \frac{\partial \tau_i}{\partial z} - S_i^{\mathrm{T}} \frac{\partial B_i}{\partial z}$
  }
}
while (forward recursion) {
  if ($i \in \mathcal{P}$) {
    $\frac{\partial \dot{V}_i}{\partial z} = \mathrm{Ad}_{T^{-1}_{\lambda(i),i}} \frac{\partial \dot{V}_{\lambda(i)}}{\partial z}$
    $\frac{\partial F_i}{\partial z} = \hat{J}_i \frac{\partial \dot{V}_i}{\partial z} + \frac{\partial B_i}{\partial z}$
    $\frac{\partial \tau_i}{\partial z} = S_i^{\mathrm{T}} \frac{\partial F_i}{\partial z}$
  } else {
    $\frac{\partial \ddot{q}_i}{\partial z} = \Psi_i \quad \frac{\partial \tau_i}{\partial z} - S_i^{\mathrm{T}} \hat{J}_i \mathrm{Ad}_{T^{-1}_{\lambda(i),i}} \frac{\partial \dot{V}_{\lambda(i)}}{\partial z} - S_i^{\mathrm{T}} \frac{\partial B_i}{\partial z}$
    $\frac{\partial \dot{V}_i}{\partial z} = \mathrm{Ad}_{T^{-1}_{\lambda(i),i}} \frac{\partial \dot{V}_{\lambda(i)}}{\partial z}$
    $\frac{\partial F_i}{\partial z} = \hat{J}_i \frac{\partial \dot{V}_i}{\partial z} + \frac{\partial B_i}{\partial z}$
  }
}

In case of $z = \ddot{q}_j$:

while (backward recursion) {
  $\frac{\partial B_i}{\partial z} = \sum_{k \in \mu(i)} \mathrm{Ad}^*_{T^{-1}_{i,k}} \frac{\partial \beta_k}{\partial z}$
  if ($i \in \mathcal{P}$) {
    $\frac{\partial \beta_i}{\partial z} = \frac{\partial B_i}{\partial z} + \hat{J} S_i \frac{\partial \ddot{q}_i}{\partial z}$
  } else {
    $\frac{\partial \beta_i}{\partial z} = \frac{\partial B_i}{\partial z} - \hat{J} S_i \Psi_i S_i^{\mathrm{T}} \frac{\partial B_i}{\partial z}$
  }
}
while (forward recursion) {
  if ($i \in \mathcal{P}$) {
    $\frac{\partial \dot{V}_i}{\partial z} = \mathrm{Ad}_{T^{-1}_{\lambda(i),i}} \frac{\partial \dot{V}_{\lambda(i)}}{\partial z} + S_i \frac{\partial \ddot{q}_i}{\partial z}$
    $\frac{\partial F_i}{\partial z} = \hat{J}_i \frac{\partial \dot{V}_i}{\partial z} + \frac{\partial B_i}{\partial z}$
    $\frac{\partial \tau_i}{\partial z} = S_i^{\mathrm{T}} \frac{\partial F_i}{\partial z}$
  } else {
    $\frac{\partial \ddot{q}_i}{\partial z} = \Psi_i \quad -S_i^{\mathrm{T}} \hat{J}_i \mathrm{Ad}_{T^{-1}_{\lambda(i),i}} \frac{\partial \dot{V}_{\lambda(i)}}{\partial z} - S_i^{\mathrm{T}} \frac{\partial B_i}{\partial z}$
    $\frac{\partial \dot{V}_i}{\partial z} = \mathrm{Ad}_{T^{-1}_{\lambda(i),i}} \frac{\partial \dot{V}_{\lambda(i)}}{\partial z} + S_i \frac{\partial \ddot{q}_i}{\partial z}$
    $\frac{\partial F_i}{\partial z} = \hat{J}_i \frac{\partial \dot{V}_i}{\partial z} + \frac{\partial B_i}{\partial z}$
  }
}

Table 6: **Derivatives of Hybrid Dynamics**