Dynamo: Dynamic Data-driven Character Control with Adjustable Balance (vgsp_0007)

Pawel Wrotek
Brown University

Odest Chadwicke Jenkins*
Brown University

Morgan McGuire
Williams College

Abstract

Dynamo (DYNAmic MOtion capture) is an approach to controlling animating characters in a physically dynamic virtual world. Leveraging existing methods, characters are simultaneously physically simulated and driven to perform kinematic motion (from mocap or other sources). Continuous simulation allows characters to interact more realistically in a more generic fashion than under methods that switch between animations.

One contribution of Dynamo is the stable playback of motion capture in the presence of dynamic interactions and without excessive gain tuning. This stability is achieved by promoting joint target angles from the traditional parent-bone reference frame to the reference frame of the character as a whole. Control in this manner allows a character to set and maintain poses in world space robust to dynamic interactions and produce physically plausible transitions between motions without explicit blending. To provide human-plausible global motion, a weak spring at the character root tempers our world-space model to account for external constraints that should break balance. This root spring provides an adjustable parameter that allows characters to fall when significantly unbalanced or struck with extreme force.

We demonstrate Dynamo through in-game simulations of characters walking, running, jumping, and fighting on uneven terrain while experiencing dynamic external forces. Using standard physics and graphics engines, Dynamo performs control without significant cost to game frame rate.

CR Categories:

Keywords: animation, physical simulation, motion capture, rag doll

1 Introduction

The motion of characters for virtual worlds comes from many sources. These include motion capture, manual keyframes, dynamic simulation (e.g., 'rag doll'), and inverse kinematics. Applications like video games and robotics require creating new motions on the fly by combining motions from these sources. For example, a virtual boxer's attacks are primarily driven by motion capture, however he must recoil or fall according to dynamic simulation when he himself is hit. Creating a new motion in real-time by combining simultaneous motions from two different kinds of sources is a hard problem. Even without dynamic simulation, blending two motion capture animations is often insufficient or overly limited due to physically implausible artifacts in the resulting motion [Safonova and Hodgins 2005].

Figure 1: Screenshot from a video-game scenario: two dynamically simulated Dynamo-controlled boxers fight on an uneven platform. The boxers stagger under each other's blows, appropriately place their feet on the uneven surface, smoothly transition between motions including rising from falls, and can even be bludgeoned to unconsciousness. The physical simulation and control computation run in 0.001s per frame.

Although rigid body physics has been incorporated into several games and engines (e.g., Havoc, Ageia, Dismount), the role of physical entities has often been limited to passive dynamics. In other words, limp entities that exert no motor forces. Creating active physical characters that impose force to achieve some objective involves creating an autonomous controller. Several research efforts [Hodgins et al. 1995; Wooten and Hodgins 2000; Faloutsos et al. 2001] have found that creating and coordinating such active controllers is a non-trivial task due to the generation of desired motion (i.e., trajectory formation) and tuning of motor gains (i.e., proportional scaling of motor force to desired offset in pose).

We posit that these classic active control problems can be addressed in part by revisiting the basic mechanisms for controlling physically simulated characters. Through the proliferation of motion capture, the creation and reuse of human-like motion has experienced increased generalization and customization through recent methods for motion graphs [Kovar and Gleicher 2004; Arikan et al. 2002], biomechanical modeling [Liu et al. 2005] and switching between mocap and simulation [Zordan et al. 2005; Mandel 2004]. However, these methods do not consider the physics of the environment or severely limit physical interaction (e.g., instantaneous reactions). Similar to [Zordan and Hodgins 2002], we believe a viable and undervalued approach to active control combines the dynamics of physical simulation with task-oriented desired motion produced via motion capture or inverse kinematics. Specifically, desired motion is used to produce motor forces to drive physical characters.

However, the problem of active motion control, translating desired motion into motor forces, remains a difficult problem due to difficulty in maintaining global constraints (e.g., balance) and the sensitivity of motor gains. A motion control system suitable for games

*e-mail:cjenkins@cs.brown.edu

should be able to support active characters capable of executing locomotion, object manipulation, and recovery in highly dynamic situations. Despite having infinitely powerful motors, current control methods have achieved this functionality up to the point of non-locomotive behavior requiring heavy tuning of motor gain parameters.

In this paper, we propose an approach to motion control suitable for interactive gaming through the world space expression of desired character pose. We hypothesize that desired motion for proportional-derivative servo (PD-servo), used in most active control animation applications, is better expressed in world space rather than in parent space. A PD-servo generates motor forces that attempt to minimize the angular error between the desired and current configuration of each rotational degree of freedom of a character. Angular configuration of a joint has been traditionally described with respect to the rotational axes based on the inboard (or parent) body attached to a joint. However, this form of proprioception, sensing of one's own body, is a holdover from older mechanical sensing limitations in the form of angular potentiometers and optical encoders. In contrast, our world space PD-servo can be analogized to setting desired character pose based on accelerometers at each rigid body. Motion control in this fashion generates motor forces that minimizes the global orientation of each body. As a result, our world space PD-servo provides stable motion control without excessive tuning to find critical damping.

We propose a world space servo strategy and present results for actuating motion capture data on a physically simulated humanoid. World space servoing allows for characters to balance in dynamic environment without explicit center-of-mass consideration. This servo mechanism is tempered by a root spring mechanism that can be adjusted to allow for characters to avoid "super human" control and fall in desired situations. We consider motions that demonstrate motion control over various combinations of static balance, dynamic balance, bipedal locomotion, object and inter-character interaction. In particular, we focus on active character control for dynamically interactive situations. To this end, we have implemented dynamical humanoids in our game engine at frame rates exceeding real-time. We demonstrate these humanoids in a simple fighting game and performing various motions, such as walking, jumping, boxing, and gymnastics, on uneven terrain and subject to user interaction.

2 State of the Art and Related Work

We summarize the state of the art in producing physically plausible character animation. Our coverage focuses on existing representations and methods for skeletal animation, motion capture, motion generalization, physical simulation, and applications in games and film. Our proposed approach builds on this previous work to yield an integrated means for realizing interactive control of physical characters.

2.1 Skeletal Kinematics

A skeleton is defined by tree of *bones*, where pairs of bones are connected by revolute joints. For a humanoid character, the root is typically the pelvis and the leaves are fingers, toes, and skull. The simulation skeleton (or kinematics) does not have to match the actual anatomy of a human skeleton. Often each simulation bone is a simplification of a number of real bones. For simulation purposes, each bone is parameterized on its length and its orientation at the joint with its parent. For a human model, the shoulder joint is thus modeled as part of the humerus (upper arm) bone.

A frame is the configuration of a skeleton at a specific instant of time. A configuration defines the desired orientation of each bone in the skeleton. A motion is a sequence of frames over time.

During animation the skeleton can be *skinned* by a 3D mesh to make it appear as realistic figure. A simple skinning method is to render each bone as a mesh of approximately the same length with orientation given by that bone. This creates the appearance of an action figure toy, where no matter how lifelike the mesh appears it will still move as a jointed rigid body. A more sophisticated method is called *matrix skinning* (see [Lewis et al. 2000] for discussion). This method drapes a complete mesh over the skeleton and at each vertex assigns a set of coefficients determining how that vertex is influenced by nearby bones. As the bones animate the vertices smoothly deform creating the illusion of a complex musculature. One drawback of this approach is that vertices tend to pinch at joints, especially under twisting. Skinning methods that preserve apparent muscle volume at joints are an active area of research. Note that with any skinning method the skeleton itself is not rendered in this process, and some bones may actually extend outside the visible 3D mesh.

2.2 Animation from Motion Capture

Motion capture is the process of estimating a set of frames that describe the motion of a human performer. This process typically through some inference of human configuration from sensor data, such as video, reflective markers, and electromagnetic systems, at discrete instances of time. Once captured, estimated motion can be played back by interpolating across these discrete frames. Splines are used to achieve smooth (C^2) transitions and velocities through the keyframes. Shoemake's classic notes [1985] describe how to perform this operation on orientations expressed as quaternions.

However, strict playback of motion capture in this fashion ignores the general notions of physics from which is was generated. Similar to a video, a captured motion is a static observation of the physics that occurred over a specific interval of time. Several research efforts have been geared towards generalizing captured motion such that it can be reused for creating new motions. Motion editing through spacetime constraints [Gleicher 1998] is one well studied method. Such optimization procedures can incorporate physical and biomechanical constraints such as in [Popović and Witkin 1999; Liu et al. 2005].

Complementary to optimization, motion graphs [Kovar et al. 2002; Arikan et al. 2002] can be constructed by finding suitable transitions between frames of different motions in a database. The suitability of a transition is determined by a metric that specifies the perceptual quality of blending between motions at two specific frames. This metric could be cast as a spatio-temporal kernel whose transitive relations allow for the estimation of motion clusters [Jenkins and Matarić 2004; Kovar and Gleicher 2004]. Motion clusters can be generalized to form new motions through interpolation [Rose et al. 1998]. While such methods provide good kinematic control, interactive and physically dynamic character control with these methods has remained elusive.

2.3 Active Dynamical Control and Motion Capture

Dynamical simulation involves modeling the dynamics of a physical system such the motion of objects in the system can be predicted. Assuming rigid objects, the physical state of a virtual world can be stated as the position, orientation, and velocities of each object and any constraints between objects, such as joints. If an appropriate model of physics, a dynamical simulation will be able to integrate physics over some specified interval of time to predict

the state of the world at end of the time interval. With the influx of dynamics engines (e.g., the Open Dynamics Engine, Havoc, Massive), the level of physical realism has drastically improved in recent years.

While physics can now be applied to virtual characters, the control of articulated characters that dynamically perform subject to physics remains problematic. Often, physics in video games is limited to rigid objects (Valve's Half-life2), inanimate entities (ragdoll), or blobby characters (Chronic Logic's Gish). Control of articulated and humanoid characters has not quite materialized. Analogous to autonomous robotics, this circumstance can be attributed to the uncertainty induced by physics and need for suitable artificial intelligence to deal with this uncertainty.

In their groundbreaking work, Hodgins et al. [Hodgins et al. 1995; Wooten and Hodgins 2000] have demonstrated that active control of articulated structures can be achieved for a variety of dynamical activities. Their examples spanned several athletic behaviors, including running, cycling, diving, and gymnastics. Such functionality, however, required an costly amount of effort to manually craft controllers for a specific activity and tune controller motor gains. The resulting controllers lacked scalability to new motor activities and kinematic structures and were not overly robust to physical disturbances. Given the difficulty in controller creation, later work [Hodgins and Pollard 1997; Faloutsos et al. 2001] addressed scalability issues of preexisting controllers, but defining and implementing new controllers remained burdensome.

Moving away from manually crafted controllers, Zordan and Hodgins [Zordan and Hodgins 2002] used motion capture to define a more general data-driven control method. In their method, a feedback control system was used drive a character to perform motion capture data. Characters can be controlled to perform any desired motion from a variety of sources, such as motion capture or inverse kinematics, instead of from a programmer's preconceptions. Although this control approach offers greater scalability, the authors reported difficulty selecting appropriate motor gains and limitations to non-locomotive (statically balanced) activities. To achieve locomotive abilities, these authors have more recently proposed methods [Zordan et al. 2005; Mandel 2004] that primarily utilize motion capture and transition to and from dynamic simulation for brief physical interactions.

Inspired by [Zordan and Hodgins 2002], we have aimed to remove the burdens of motor gain tuning and move towards fully simulated interactive characters. In our approach, we augment the basic PD-servo mechanism to use world space desired poses and realize control over motions with locomotive properties to new environments without burdensome gain tuning.

2.4 Games and Film

Games and movies contain sophisticated motions that appear to integrate inverse kinematics, simulation, and motion capture elements. Examples include animation of on-the-fly created characters in Spore (EA 2006), massive battle scenes in The Lord of the Rings film (Weta using Massive), and boxers in Fight Night Round 3 (EA 2006).

However, with regard to general-purpose simulation, games and movies represent the "state of the artist," not the "state of the art." In film, good motion is often the combination of hours of offline simulation and hand tweaking by experienced animators. For games, the algorithms run in real-time but often rely heavily on precomputed animation sequences[1] and are designed to operate in constrained situations that avoid difficult cases. In most games, full details are never publicity disclosed, so we have no way of knowing how reproducible or general these results are. We are motivated to make the kinds of effects we see in high-budget games and film 'real' so that they can be applied in more general situations and inform other disciplines like robotics, where hand-tuning or situation-specific results are insufficient.

[1] Sometimes precomputed exhaustively for all possible interactions!

3 Dynamo

Characters are simulated as articulated rigid bodies with joint limits. As with any other body in a physics simulation, they can collide with themselves and with the environment. They experience external forces like friction and gravity.

We drive these characters by computing motor forces for joint servos. The Dynamo control system comprises four basic ideas detailed in this section:

1. Apply torques to match desired *world-space* pose.
2. Maintain root orientation with a weak spring.
3. Simulate stunning by decreasing motor gains.
4. Motion blending emerges from continual simulation.

3.1 Working with World-Space Ball Joints

It is common practice to apply a proportional-derivative (PD) controller to compute motor torques about each degree of freedom based on the desired and actual angles and angular accelerations:

$$\text{torque} = k_1(\theta_{\text{desired}} - \theta_{\text{actual}}) + k_2(\dot{\theta}_{\text{desired}} - \dot{\theta}_{\text{actual}}) \quad (1)$$

In this equation, k_1 and k_2 are the user-specified gains on the simulated joint motor with respect to position and velocity (we denote all user-specified constants as k-values to distinguish them from derived quantities). Variable θ is only for exposition in this notation section and does not appear again in the paper.

Equation 1 is directly applicable to parent-space hinge joints, which are parameterized by a single joint angle that is easily measured. Like previous work [Zordan and Hodgins 2002], we use PD controllers, but must introduce a slightly richer notation to move between parent and world space and to generalize from hinge joints to ball joints with three degrees of rotational freedom (of course, 1- and 2-axis joints remain expressible within this generalization). For a given simulation step, we denote the known quantities *at each bone* as:

$\mathbf{P}_d$	Desired *parent-space* orientation (3×3 matrix) from mocap
$\mathbf{W}_d$	Desired *world-space* orientation (eq. 3)
$\mathbf{W}_a$	Actual world-space orientation from sim
ω_d	Desired world-space angular velocity (3-vector) (eq. 4)
ω_a	Actual world-space angular velocity from simulation

Our matrix multiplications assume column-vector matrices, where a vertex to be transformed would appear on the right of the matrix.

From these variables, we solve for world-space torque τ. Unless otherwise specified, these variables refer to the current bone b; alternately an additional subscript specifies the parent or root bone.

We operate on 3D orientation matrices instead of 1D angles, so it is necessary to express the differences in equation 1 as a function mapping two matrices to a 3-vector whose direction $\vec{v}$ is the axis of rotation between the reference frames and whose magnitude is the rotation angle θ. We denote this difference

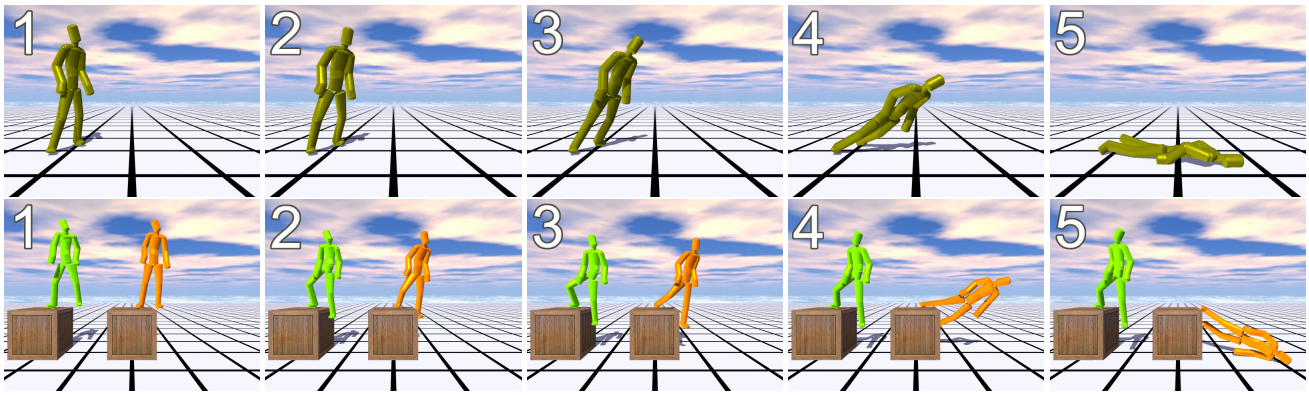

Figure 2: Examples of unbalanced and super-balanced characters. (Top) A character on a flat floor actuating mocap with an unpowered root quickly falls over, even when starting from a balanced position. This occurance is due to the uncertainty induced by physical simulation. (Bottom) The leftmost, green character with a meathook root remains upright, but implausibly balances with one foot on a crate. The rightmost, orange character with a weak spring root yields the desired behavior.

$$\Delta(\mathbf{D},\mathbf{A}) = \theta\vec{v}, \tag{2}$$

where θ and $\vec{v}$ are the axis and angle of the product $\mathbf{D} * \mathbf{A}^{-1}$. Appendix A lists source code implementing this function.

3.2 Torques from Captured Motion

Characters apply torques to their bones to match target motion capture or key frame poses. Motion capture data typically is expressed as a series of key poses. A pose contains a coordinate frame for each bone relative to its parent. Without loss of generality, assume that the target pose is exactly specified in the original data by a key pose; poses between key poses are obtained by interpolation, e.g., spline interpolation of the frames expressed as quaternions [Shoemake 1985].

We first lift each bone's target orientation frame from parent space matrix $\mathbf{P}_d$ to world space matrix $\mathbf{W}_d$ by recursively applying the target frames:

$$\mathbf{W}_d = \begin{cases} \mathbf{P} & b = \text{root} \\ \mathbf{W}_{a,\text{root}} * \mathbf{P}_d & \text{parent} = \text{root} \\ \mathbf{W}_{d,\text{parent}} * \mathbf{P}_d & \text{otherwise} \end{cases} \tag{3}$$

The *actual* orientation of all bones except the root are ignored by this transformation. This is the key to the stability of Dynamo. Previous methods compute a target relative to the current parent reference frame, which propagates error down long linkages.

We define the desired world-space angular velocity ϖ as the angular velocity needed to reach the desired pose at some future time $(t+\delta)$ given the desired pose at the current time t,

$$\omega_d = \frac{1}{\delta}\Delta\big(W_d(t+\delta), W_d(t)\big). \tag{4}$$

In our simulations, we chose δ to be the duration between mocap keyframes. Note that we did not choose the instantaneous derivative of the current desired orientation as desired velocity. Because games integrate state in discrete Euler time steps, it is necessary to look at least one timestep into the future for velocity, since that is when the effects of the applied torque will be observed.

Torque is computed differently at the root and non-root bones. For non-root bones, the world-space torque τ to apply to the bone servo for the next simulation step is given by,

$$\tau_{b\neq\text{root}} = k_1\Delta(\mathbf{W}_d, \mathbf{W}_a) + k_2(\omega_d - \omega_a). \tag{5}$$

This is a straightforward world-space, 3D variation on eq. 1. Gain constants used in our experiment are described in Section 4. Because error is not propagated through the bone hierarchy, the constants need not be tuned precisely. Additional consideration should be given toward whether world space is defined absolute or egocentric coordinates. In absolute coordinates, the servo routine will enforce the character's pose such that the orientation of the root aligns with motion capture. Absolute coordinates leads to issues with super-balancing, described later. In contrast, an egocentric world space (or "person coordinates") leaves the root orientation free to be commanded and adjusted as desired. Thus far we have only described torques on servos for non-root bones. The following section addresses the root, which is treated specially.

3.3 Weak Root Balance Spring

Were we to leave the root unpowered, the character would eventually fall over while playing back captured motion (see Figure 2. Pure motion capture playback does produce balance for three reasons. First, the simulated character differs from the actual human who created the motion. Second, discrete simulation always accumulates error. Thus even a perfect model experiences imperfect simulation. Third, dynamic constraints are not accounted for in the mocap data, so even a slight slope to the floor will topple the character.

Today, most games directly drive the character's root bone by explicitly setting its coordinate frame, which can be observed by the character's feet sliding along the ground when the root velocity does not match the velocity implied by the animation. These explicit controllers also preclude ballistic motion–to jump, the controller must intentionally lift the character.

The lower row of characters in Figure 2 demonstrate the super-balance problem. The characters in the first frame begin in an unbalanced state with one foot on a crate. In this situation, the character should fall to the ground, which the unpowered root simulates correctly. Because the strong root joint tolerates no deviation, it produces balance in a physically implausible situation, as shown in the last frame.

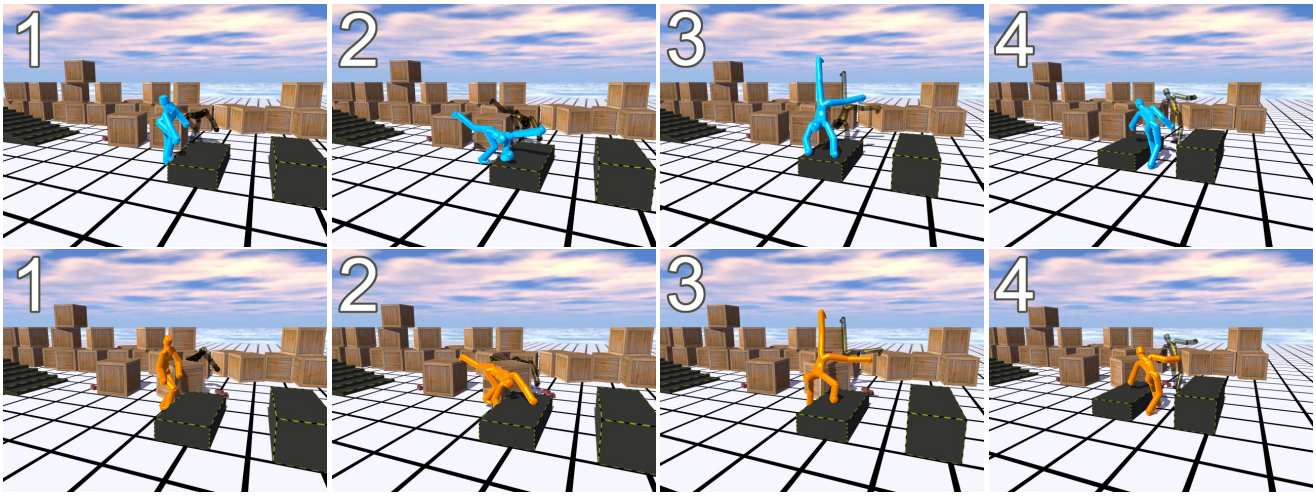

Figure 3: Sequences of parent-space (top, blue) and world-space (bottom, orange) characters adapting a cartwheel animation to an obstacle, with mocap driving the characters behind in black. Both methods produce reasonable results, with better preservation of the mocap by the world space method.

The ideal balance controller distributes over all bones the task of shifting weight to maintain the character's center of mass over its support polygon on the ground. Doing so is a hard problem; several techniques and tools have been demonstrated to produce good results, but none are efficient enough for real-time interactive use [Komura et al. 2004].

Dynamo uses a new technique that we call a *weak root spring*. Although physically incorrect, it produces plausible balance phenomena in many situations and is extremely efficient. The spring is a powered joint connected to the universe, like the meathook. However, unlike the meathook, it avoids super-balance through torque limits and can be broken when seriously imbalanced, producing realistic falls.

Dynamo computes the torque at the root in several stages. We notate the stages with primes indicate intermediate results as we build towards τ_{root}. The first stage applies PD equation 5 to the root,

$$\tau''_{\text{root}} = k_1 \Delta(W_{d,\text{root}}, W_{a,\text{root}}) + k_2(\omega_{d,\text{root}} - \omega_{a,\text{root}}). \quad (6)$$

Because balance should only apply torques perpendicular to gravity, the second stage removes the component parallel to rotation about the gravity force vector: F_G,

$$\tau'_{\text{root}} = \tau''_{\text{root}} - \frac{F_G\left(F_G \cdot \tau''_{\text{root}}\right)}{||F_G||^2}. \quad (7)$$

The third stage enforces two torque limits to prevent super-balance. The lower limit $k_L > 0$ merely clamps the maximum torque to be applied. When the higher limit $k_U > k_L$ is also exceeded, however, the root spring is considered broken an applies no torque. This allows a significantly imbalanced character to fall. Thus the net root spring torque applied is:

$$\tau_{\text{root}} = c \begin{cases} \tau'_{\text{root}} & ||\tau'_{\text{root}}|| < k_L \\ k_L \dfrac{\tau'_{\text{root}}}{||\tau'_{\text{root}}||} & ||\tau'_{\text{root}}|| < k_U \\ 0 & ||\tau'_{\text{root}}|| \geq k_U. \end{cases}, \quad (8)$$

where $c = 1$ when the character is in contact with the environment and therefore able to exert contact forces to balance itself, and $c = k_{air} < 1$ otherwise. We classify a character as in contact with the environment if and only if the rigid body simulation system introduced a contact constraint with another object during the previous time step.

The k_{air} constant is the amount of "air control" the character can exert. Choosing $k_{air} = 0$ completely prevents the character from balancing in the air, which is overly restrictive since it is possible to self-induce angular acceleration without exerting contact forces (falling cats and spinning ice skaters are good examples of this phenomena.)

3.4 Reaction to Impacts and Motion Blending

Similar to [Zordan and Hodgins 2002], we simulate the stunned response of the character to significant impacts by rapidly ramping down the servo gains k_1, k_2 and then slowly raising them to normal levels over several frames of animation. For very large impacts, the gains ramp all the way to zero. This creates an unpowered character–a rag doll–and simulates unconsciousness.

In some situations, powering down gains will imbalance the character. That in turn causes the root spring to break, which makes the character fall over. Note that falling down causes impacts with the ground. Those impacts cause more powering down, so even if the initial impact was light the character may end up unconscious.

The idea of reducing gains to simulate a stunned character is not novel. What is unique about Dynamo is that working in world space makes the simulation sufficiently robust over a large range of servo gains that animation remains stable as we move the gains far from their hand-tuned optima.

As with stunning, Dynamo's stability allows a very simple motion blending strategy to produce effective results. To transition between animations, we simply linearly interpolate the world space desired poses. The control system then produces plausible transitions from motor torques that incrementally attract towards the new pose. Assuming reasonable gains, the character will not snap instantly to the new pose due to constraints imposed by continual physical simulation. Parent-space methods can achieve similar results if precisely appropriate gains are found. Otherwise, the resulting motion is subject to lagging and wobbly oscillations.

4 Results

We implemented a test framework for the Dynamo control system that simulates a video game environment. Rigid body physics are provided by the Open Dynamics Engine (ODE; http://ode.org). The G3D library (http://g3d-cpp.sf.net) is used for rendering and general 3D support code. Tests were run on a single-core 3.5 GHz Intel Pentium 4 processor under Windows XP. Humanoid kinematics and motion were provided through mocap data obtained from the CMU Motion Capture Database, Credo Interactive's MegaMocap V2 package, and a custom animated "stand up" motion.

The result figures in this section are exerpts of our comprehensive results video. Most of our result figures show simple scenes to make the experiments clear; we can simulate significantly more complex cases.

4.1 Performance

All results were computed in real-time. We clocked simulation at 120 Hz because the underlying ODE library uses static penetration-based collision detection and misses fast-moving objects at larger timesteps. A high simulation rate also allows finer differential control on the servos.

The entire simulation process is very efficient. ODE is highly optimized and Dynamo adds little overhead. This speed is important for games where simulation is only allocated a fraction of the CPU since it competes with audio, AI, graphics, and network code. A typical game scene involving five articulated characters with 36 bones each, 50 other rigid bodies, and a 5000-polygon environment. Simulation requires approximately 0.001 seconds per frame (3% of the CPU) to process all rigid body collision detection, computation of desired torques, and the full constrained solver inside ODE.

4.2 Gain Insensitivity

Our video results for cartwheel and obstacle navigation tasks illustrate gain insensitivity for world space servoing. Shown left-to-right in Figure 3, characters are playing straight motion capture with no simulation (black), parent space servoing (blue), and world space control using Dynamo (orange). Significant time was spent finding the best servo gain constants for critical parent-space damping. The world space method is sufficiently insensitive that we experimented with both custom gains as well as direct usage of the parent-space gains. Both sets of gains produced similar results for world space servoing. Figure 3 compares parent-space and world-space characters adapting to an obstacle. If our character is located in a flat, featureless environment, Dynamo drives it to perform the cartwheels just as the original motion indicates. However, if there is an obstacle in the way our character now reacts accordingly and realistically vaults over the obstacle. Parent space can reasonably adapt in many cases, but produces a weaker motion that loses much of the style from the mocap.

In the obstacle adapation task (Figure 4.3), however, world space servoing demonstrated a clear improvement to parent space. The character for this task was required to walk up a ramp, down stairs, through a hanging crate, and finish with cartwheels over large blocks in the walking path. During its navigation of the course, the world-space character remains true to the motion capture animation, but the parent-space character has less stability and appears to quiver and jiggle.

Dynamo's robustness to gain variations is a strong result. Hand-tuning gains for a character is a time consuming process, which is substantially reduced by moving to world-space where the exact constants are not so critical.

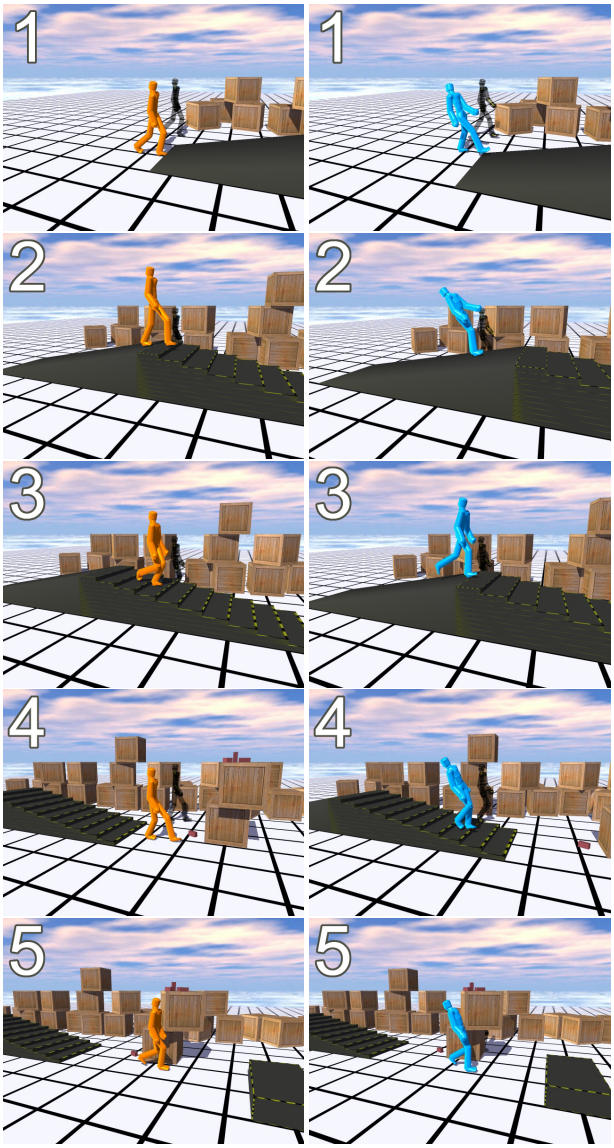

Figure 4: A pair of characters walking an obstacle course with torques computed in world space (left, orange) and parent space (right, blue). Our best attempts at critical damping parent space incurred wobble in the torso. World space servoing produced more faithful motion across various damping parameters.

4.3 Balance and Adaptation

Because we use motion capture only as a guideline for actively controlling our character, the character can dynamically interact with its environment in plausible ways. A basic form of dynamic interaction is static balance. As illustrated earlier in Figure 2, a character actuating mocap for idly standing should remain balanced on flat ground and fall in unbalanced states, such as with one foot on a crate. Our weak root spring controller generates plausible control in both cases.

Static balance and plausible falling were tested subject to interactive applications of force. Figure 5 shows a Dynamo character re-

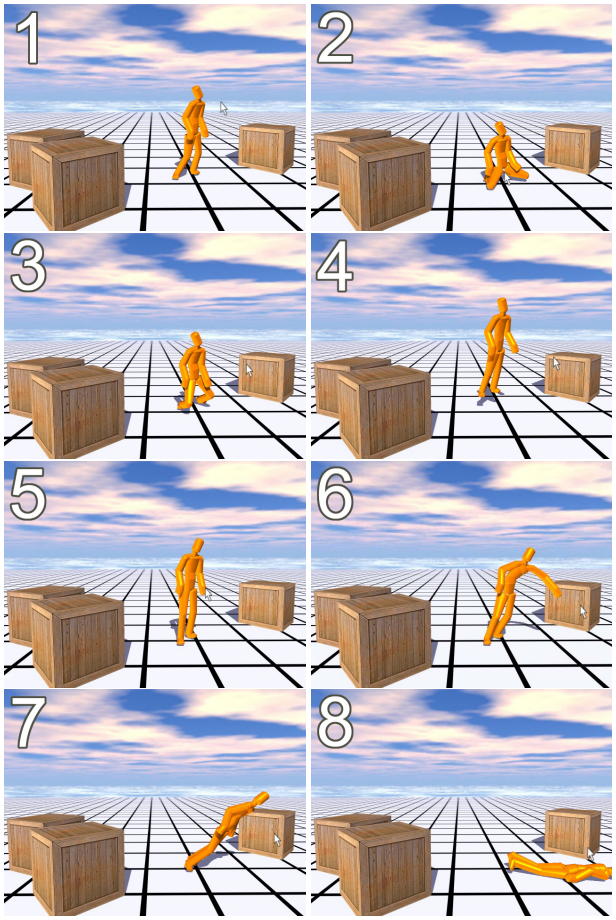

Figure 5: Sequence of an animated Dynamo character under the influence of strong external forces; in this case, the user is dragging the character about with the mouse.

acting plausibly to strong external forces and user-imposed constraints. We placed the character in an environment littered with obstacles and then violently dragged it about with a mouse-driven cursor. When pressed into the ground, the Dynamo character bends at the knees and then stands upright when released. If knocked over the character lies on the ground, but if put back on its feet it regains balance. All objects react realistically when struck due to the continuous underlying physical simulation.

Our humanoids were capable of performing tasks involving plausible dynamic balance, such as locomotion and jumping. Figure 4.3 shows our humanoid using a flat ground walking motion to traverse uneven ground and walk through obstacles. Because the volume of the capture subject is not an exact match to the humanoid geometry, locomotion often incurs premature foot contacts with the ground. We account for this by applying a small upward force on a foot with forward momentum that contacts the ground. In Figure , we show a Dynamo-controlled humanoid driven by a jumping motion. The character achieves flight off of the ground and is able to stay upright upon landing. However, the character lags behind the mocap due, which we attribute to overcoming gravity.

4.4 Impacts

In the absence of a task-driven artificial intelligence (AI) controller, the simulation will attempt to continue playing motion capture even after a character has lost balance. This produces a character lying

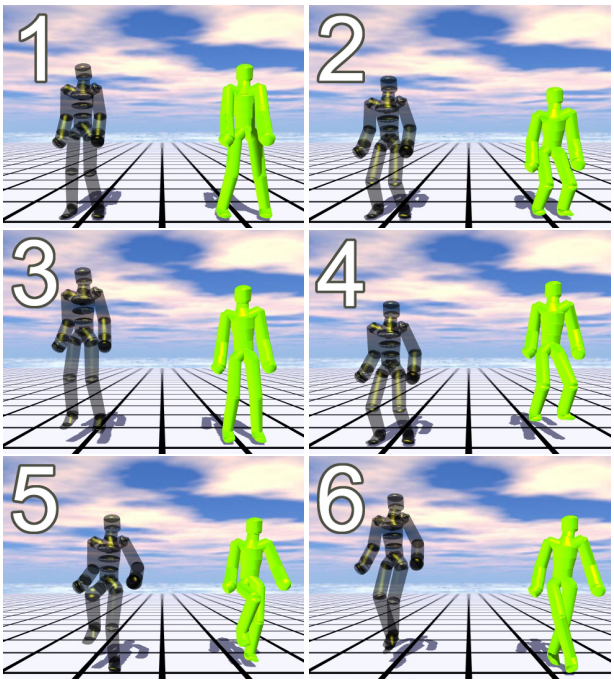

Figure 6: A Dynamo character (right, green) driven by motion capture (left, black) to perform ballistic jumping motion, hopping on one leg, and leg crossovers. The Dynamo character is capable of jumping off the ground and maintaining dynamic balance, though not in complete synchrony with mocap due to gravity.

on the ground but attempting to walk or make other movements. We claim that this is a desirable result from the simulation system– it is what a robot would do, and is what a human would do given extremely rigid instructions about matching motion capture. However, this is not a desirable behavior for a plausible intelligent being.

A simple AI on top of Dynamo can produce very plausible results when the character is knocked over but still conscious. For the example we implemented, the AI detects when root spring has been broken for more than an epsilon amount of time (0.5 seconds). When broken, we systematically decrease the gains (by applying a scale factor) to stun the character towards a rag doll state. If the spring is restored, the gains are ramped back up. the root spring is broken and the character is in contact with the ground. Figure 7 shows an additional result that combines the root spring and stunning. This example shows three Dynamo characters repeatedly hit with “beach ball” objects and finally impacted with a crate. Each root controller maintains balance during the light impacts of the beach balls. The heavier crate impact demonstrates the spectrum of reactions Dynamo can produce, varying from maintaining upright balance, falling with continuing actuation, and falling to unconsciousness.

4.5 In-Game Boxing Result

We have developed a demonstration boxing game that pits two autonomous characters against each other. Each character constructs a motion graph [Kovar et al. 2002] from an extended sequence of punches and a standing up motion. A characters driven to perform various punching motion until stunned and knocked down by a punch from the other character. Once down, the character takes desired motion from the standing up motion and returns to the punch part of the motion graph. Figure 8 shows a frame-by-frame comparison of Dynamo-driven boxing characters with mocap playback.

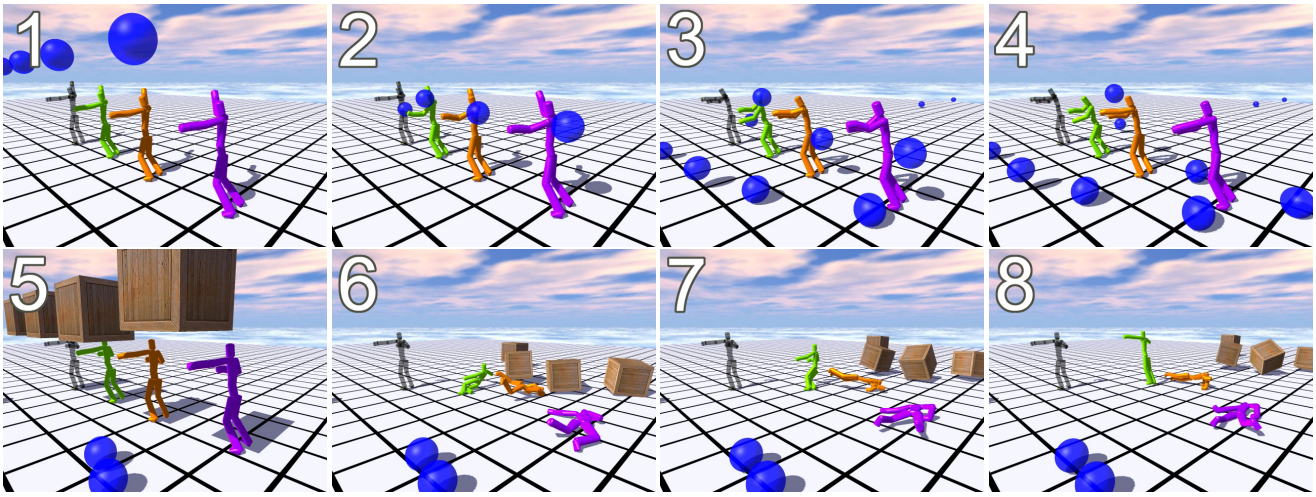

Figure 7: Effects of the weak root spring algorithm and AI. Eight frames of animation, spaced 0.1s apart showing four characters executing mocap while being hit with light beach balls and heavy wood crates. In each frame, the characters in the row are using the following algorithms from left to right: mocap with no simulation (black) as a baseline; meathook (green), weak root spring with stunning (orange); and weak root spring with a simple AI controller for switching to "unconsciousness." They are buffeted by the balls but only knocked over by the crate. Without an AI to change animations, the orange root spring character attempts to animate even after being knocked over, whereas the final purple character with an AI lies still in rag-doll mode.

Our boxing demo is similar to the upper-body example used by Zordan and Hodgins [Zordan and Hodgins 2002]. In our case, however, our characters are driven to perform full-body motion and maintain plausible foot contacts over uneven ground.

5 Limitations and Future Work

In its current form, Dynamo produces physically plausible control of articulated virtual characters. Due to our focus on video games, we have taken a few physical liberties in the control system that could be addressed as future work. A natural future line of work is to appropriately simulate the control limitations of real world robotic and prosthetic systems. This emphasis attempts to bring the real and virtual worlds closer together for leveraging their relative strengths. Virtual worlds with enhanced physics bring greater realism and expression. Physical simulation and control platforms allow for faster, more accurate, and broadly practical tools for developing control of real robotic and prosthetic systems. The success and broad usage of the Player/Stage Project [Gerkey et al.] is a clear example this benefit for mobile robotics.

Dynamo currently applies external motor forces without joint limits and unrefined torque restrictions. Specifically, motor torques are applied externally to bones individually to match a desired world space orientation. These torques are limited only by the servo spring and damping gains. A more realistic actuation model would translate these external bone forces into torques about its inboard joint with assurances about applying opposite and equal torques to parent and child bodies.

The lack of joint and torque limits for the current Dynamo has its strengths and weaknesses. Towards the positive, Dynamo characters are highly capable of physical task and are implicitly attracted towards plausible poses demonstrated in the mocap data. In addition to violating reality, however, we do not have limits that prevent the character from encountering implausible situations such as displaying super-human abilities or becoming stuck in extreme poses (Figure 9). However, exposure to extreme poses is a seldom occurrence, often due awkward physics solutions resulting from violent user interactions (refer to Figure 5). We believe advances in physical simulation make these anomalous situations even more remote.

The root spring balance model is a compromise between simulating realistic phenomena and real-time execution. A true balance model would completely replace the root spring with an AI-driven or inverse kinematic system that balances the character in a realistic manner. In addition, mocap performed by a human is typically not directly applicable to a humanoid character. While our system can plausibly balance, locomotion is particularly hampered by the complexities of mapping mocap onto humanoids. Artifacts from this mapping are often occur as dragging of the feet during locomotive motions. Unlike humans, our characters currently have no sense of the environment and, thus, make no decisions about appropriate foot contacts. Our current reactive rules yield plausible locomotive behavior. However, the real core of the locomotion problem lies within developing AI strategies to determine appropriate contacts with the world. Developing realistic balance and locomotion strategies for real-time computation is open research problem.

More generally, there is a distinction and synergy between high-level AI and low-level servoing that should be maintained. An AI system is concerned with decision making to achieve objectives. This responsibility involves factors in the global world space and setting the desired motion for low-level servo control. In this respect, the AI should take as input the current state of the world and output the desired global coordinate frame and bodycentric pose. Once decision making is performed, our methods suggest that poses in bodycentric coordinates should be used for driving low-level motor control.

6 Conclusion

We have presented Dynamo as an approach to controlling animating characters in a physically dynamic virtual world. Dynamo controlled characters are driven to perform kinematic motion under the continual influence of physical simulation. Using Dynamo, we have achieved stable motion control control without excessive gain tuning to find a specific critical damping. Our control system provides gain insensitivity by representing desired poses in world

Figure 8: Demonstration of how Dynamo makes animations appear more realistic. The top and bottom rows each show four frames from a boxing game. The top row shows a traditional pure mocap approach. Note the artifacts: feet floating, feet penetrating the ground, characters inter-penetrating, and overall lack of a reaction to hits. The bottom row shows the same scene with Dynamo. Note that feet adapt to the uneven ground by both raising and lowering as needed, and the characters react to one another's blows.

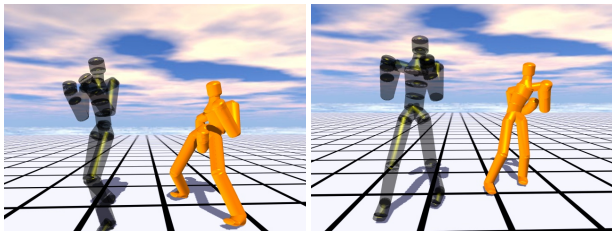

Figure 9: Situations of local extrema in pose space. Attempting to servo to a ready fighting posture, the character's right arm is stuck between its legs (right) and behind its back (left). These situations are difficult to correct without without a higher-level AI or external intervention.

space rather than the traditional parent-bone reference frame. The strength of servoing in world space is tempered a weak spring at the character root. This root spring provides an adjustable parameter to react plausibly when balance should break. Dynamo-controlled characters perform desired motion robust to dynamic interactions with physically plausible transitions between motions without explicit blending. Our basic servoing system is a foundation to begin emphasizing deeper issues in articulated character control, namely artificial intelligence.

7 Acknowledgements

The authors wish to thank Daniel Byers, Graham Rosser, and Jean Tsong for their assistance in game asset creation and movie editing. We thank the contributors to the Open Dynamics Engine and G3D projects for providing freely-available open-source physics and graphics engines. The data used in this project was obtained in part from mocap.cs.cmu.edu. This database was created with funding from NSF EIA-0196217.

References

ARIKAN, O., FORSYTH, D. A., AND O'BRIEN, J. F. 2002. Motion synthesis from annotations. *ACM Transactions on Graphics 22*, 3, 402–408.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composable controllers for physics-based character animation. In *Proceedings of ACM SIGGRAPH 2001*, 251–260.

GERKEY, B., VAUGHAN, R. T., AND HOWARD, A. The player/stage project: Tools for multi-robot and distributed sensor systems.

GLEICHER, M. 1998. Retargetting motion to new characters. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 33–42.

HODGINS, J. K., AND POLLARD, N. S. 1997. Adapting simulated behaviors for new characters. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 153–162.

HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., AND O'BRIEN, J. F. 1995. Animating human athletics. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 71–78.

HSU, E., PULLI, K., AND POPOVIĆ, J. 2005. Style translation for human motion. *ACM Trans. Graph. 24*, 3, 1082–1089.

JENKINS, O. C., AND MATARIĆ, M. J. 2004. Performance-derived behavior vocabularies: Data-driven acqusition of skills from motion. *International Journal of Humanoid Robotics 1*, 2 (Jun), 237–288.

KOMURA, T., LEUNG, H., AND KUFFNER, J. 2004. Animating reactive motions for biped locomotion. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM Press, New York, NY, USA, 32–40.

KOVAR, L., AND GLEICHER, M. 2004. Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph. 23*, 3, 559–568.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. *ACM Transactions on Graphics 21*, 3, 473–482.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformations: A unified approach to shape interpolation and skeleton-driven deformation. In *Siggraph 2000, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, K. Akeley, Ed.

LIU, C. K., HERTZMANN, A., AND POPOVIĆ, Z. 2005. Learning physics-based motion style with nonlinear inverse optimization. *ACM Trans. Graph. 24*, 3, 1071–1081.

MANDEL, M. 2004. *Versatile and interactive virtual humans: Hybrid use of data-driven and dynamics-based motion synthesis.* PhD thesis, CMU.

POPOVIĆ, Z., AND WITKIN, A. P. 1999. Physically based motion transformation. In *Proceedings of SIGGRAPH 99*, ACM SIGGRAPH / Addison Wesley Longman, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, 11–20.

ROSE, C., COHEN, M. F., AND BODENHEIMER, B. 1998. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics & Applications 18*, 5 (Sep-Oct), 32–40. ISSN 0272-1716.

SAFONOVA, A., AND HODGINS, J. K. 2005. Analyzing the physical correctness of interpolated human motion. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 171–180.

SHAPIRO, A., PIGHIN, F., AND FALOUTSOS, P. 2003. Hybrid control for interactive character animation. In *Pacific Graphics*, 455–461. short paper.

SHOEMAKE, K. 1985. Animating rotation with quaternion curves. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 245–254.

WOOTEN, W. L., AND HODGINS, J. K. 2000. Simulating leaping, tumbling, landing, and balancing humans. In *Intl. Conference on Robotics and Automation*, 656–662.

ZORDAN, V. B., AND HODGINS, J. K. 2002. Motion capture-driven simulations that hit and react. In *Proceedings of the ACM SIGGRAPH symposium on Computer animation*.

ZORDAN, V. B., MAJKOWSKA, A., CHIU, B., AND FAST, M. 2005. Dynamic response for motion capture animation. *ACM Trans. Graph. 24*, 3, 697–701.

A Matrix to Axis-Angle

The following C++ source code implements the Δ function described in eq. 2. It is based on source code is from the G3D (http://g3d-cpp.sf.net) library and David Eberly's defunct Magic Software library.

```
Vector3 Delta(const Matrix3& D, const Matrix3& A) {
    Vector3 axis;
    float angle;
    // For orthonormal matrices, A^-1 == A^T
    (D * A.transpose()).toAxisAngle(axis, angle);
    return axis * angle;
}

void Matrix3::toAxisAngle(Vector3& axis, float& angle) const {
    float trace = elt[0][0] + elt[1][1] + elt[2][2];
    float c = 0.5f * (trace - 1.0f), half;
    angle = acos(c);

    if (angle > 0.0) {
        if (angle < PI) {
            axis.x = elt[2][1] - elt[1][2];
            axis.y = elt[0][2] - elt[2][0];
            axis.z = elt[1][0] - elt[0][1];
            axis.normalize();
        } else if (elt[0][0] >= elt[1][1]) {
            // r00 >= r11
            if (elt[0][0] >= elt[2][2]) {
                // r00 is maximum diagonal term
                axis.x = 0.5f * sqrt(elt[0][0] - elt[1][1] -
                             elt[2][2] + 1.0f);
                half = 0.5f / axis.x;
                axis.y = half * elt[0][1];
                axis.z = half * elt[0][2];
            } else {
                // r22 is maximum diagonal term
                axis.z = 0.5f * sqrt(elt[2][2] - elt[0][0] -
                             elt[1][1] + 1.0f);
                half = 0.5f / axis.z;
                axis.x = half * elt[0][2];
                axis.y = half * elt[1][2];
            }
        } else if (elt[1][1] >= elt[2][2 ) {
            // r11 is maximum diagonal term
            axis.y = 0.5f * sqrt(elt[1][1] -
                         elt[0][0] - elt[2][2] + 1.0f);
            half = 0.5f / axis.y;
            axis.x = half * elt[0][1];
            axis.z = half * elt[1][2];
        } else {
            // r22 is maximum diagonal term
            axis.z = 0.5f * sqrt(elt[2][2] -
                         elt[0][0] - elt[1][1] + 1.0);
            half = 0.5f / axis.z;
            axis.x = half * elt[0][2];
            axis.y = half * elt[1][2];
        }
    } else {
        // Zero angle
        axis.x = 1.0f;  axis.y = 0.0f;  axis.z = 0.0f;
    }
}
```