A Dynamic Controller Toolkit

Ari Shapiro Derek Chu Brian Allen Petros Faloutsos
University of California, Los Angeles

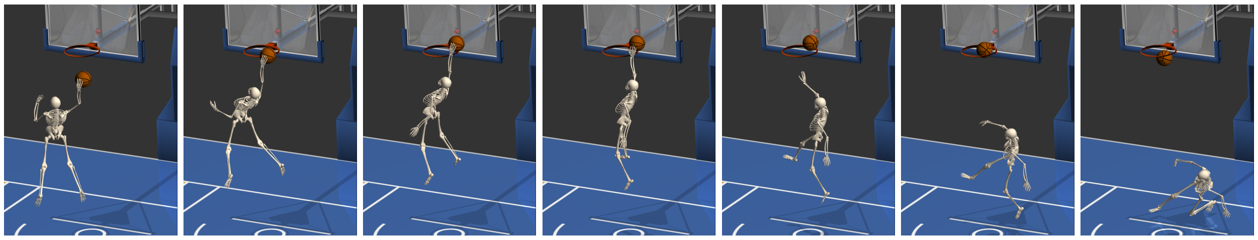

Figure 1: *A simple controller demonstrates the use of inverse kinematics, dynamic attachments and simple logic to earn the skeletal team two points.*

Abstract

We introduce a toolkit for creating dynamic controllers for articulated characters under physical simulation. Our toolkit allows users to create dynamic controllers for interactive or offline use through a combination of both visual and scripting tools. Users can design controllers by specifying keyframe poses, using a high-level scripting language, or by manipulating the rules of physics through a group of helper functions that can temporarily modify the environment in order to make the desired animation more feasible under physical simulation. The goal of the toolkit is to integrate dynamic control methods into a usable interactive system for non-computer scientists and non-roboticists, and provide the means to quickly generate physically based motion.

CR Categories: I.3.4 [Computer Graphics]: Graphics Systems—Graphics packages

Keywords: Physical simulation, dynamic control, character animation

1 Motivation

Motion capture techniques effectively animate humanoid characters by recording and reproducing live performances. However, they present several limitations, most notably, an inability to adjust to new environments and different virtual characters. Motion editing is often non-trivial and prone to problems such as constraint violation (e.g. footskating).

Physical simulation offers an alternative approach to generating motion for animated characters. Synthesizing motion by numerically simulating the laws of physics offers the most general constraint over the resulting motion and ensures realistic results. The dynamic nature of such approaches makes them especially suitable for interactive applications where characters may interact in ways that cannot be predicted or pre-computed. For this reason, physical simulation is becoming increasingly popular in interactive applications such as computer games.

Although physical simulation is well-understood, controlling a physically simulated character in order to make her perform desired actions is not. The difficulty in developing dynamic controllers is partly due to the complexity of control algorithms and partly due to a lack of tools for animators to use to easily create dynamically-based motion. While it is unlikely that the members of the graphics community without robotics or biomechanical knowledge will develop better control algorithms than those in the robotics community, the ability to change the virtual environments to suit the desired behavior of the animated characters is an advantage for graphics and animation developers. For example, motion capture can be combined with dynamics, characters can be constrained to maintain balance, and different collision geometries can be added and removed during simulation runtime. This is the focus of our work.

1.1 Contributions

We present a toolkit for use in the development of dynamic controllers. The toolkit allows animators to develop dynamics-based controllers through a combination of:

- key-framed based control,
- reduced dimensionality physics,
- scripting controllers via a controller language,
- interactive control of dynamic characters.

Although researchers have identified various control algorithms for specific scenarios, the barrier to entry for developing dynamic controllers is due to a very sparse toolset. We present a set of tools that will allow animators with little or no skill in dynamics to create dynamic controllers for use in physical simulation. We demonstrate that such tools can be effectively used for the development of character animation.

Our paper is organized as follows: Section 2 discusses related work. Section 3 gives details of our pose-based control method. Section 4 discusses the use of modified physical environments. Section 5 details our scripting environment. Section 6 discusses our use of interactive control for physical simulation. Finally, Section 7 discusses the problems facing the development of dynamic control and concludes.

2 Related Work

The focus of this paper is dynamic control of humanoid characters. We review work related to this topic.

Dynamic controllers have been developed for athletic maneuvers, such as running, bicycling, vaulting and balancing in [Hodgins et al., 1995]. Control schemes using limit cycles were developed in [Laszlo et al., 1996]. [Faloutsos et al., 2001] described a priority-based control scheme and used support vector machines to automatically determine the appropriate dynamic controller for the character's state. [Yang et al., 2004] layered different levels of controllers to achieve dynamics-based swimming. [Zordan et al., 2004] used simple controllers for breathing. [Pollard and Zordan, 2005] used passive controllers for grasping and gripping. [Smith, 2003] used a neural net to generate a stable walking cycle. Dynamic controllers that use proportional derivative (PD) control strategies tend to produce motion that appears robotic. [Neff and Fiume, 2002; Neff and Fiume, 2003; Neff and Fiume, 2004] used dynamic control algorithms in order to achieve natural-looking motion and create expressive animation with human-like fluidity.

Our work integrates many techniques developed for dynamic control into a usable, interactive system.

Another set of methods leverage well known motion constraints, such as those from motion capture or keyframing, to modify or guide dynamic motion. [Komura et al., 2004] and [Zordan and Hodgins, 2002] enabled reactive motions by tracking motion capture and responding to physical disturbances. [Yin et al., 2003] tracks motion capture by fixing the root of the character then applying dynamic effects on the rest of the body through physical simulation. [Oore et al., 2002] presented localized dynamic models in combination with kinematics.

Hybrid strategies combine the use of motion capture or other data-driven kinematic motion with physical simulation, depending on which method is more likely to generate the desired effect. [Shapiro et al., 2003] applied kinematic controllers for ordinary motion then switched to dynamic controllers under physical disturbances, and also specified a criteria for determining when such a switch is appropriate. [Mandel, 2004] used the same control strategy under physical contact and emphasized reactive motions, such as falling and recovering. [Zordan et al., 2005] switched from motion capture to physical simulation, tracked the projected path using a simple pose-based controller, then blended the motion to match the beginning of another motion capture sequence. [Wrotek et al., 2006] tracked motion capture data using global-space instead of local-space under an environment with modified physics.

Other research handles the problem of interaction with humanoid characters during simulation. [Laszlo et al., 2000] interactively controlled 2-D characters with the keyboard and mouse. A physically-based skier was controlled through basic input of the mouse or keyboard causing a 2-D character to crouch, lean or jump in the Ski Stunt Simulator [van de Panne and Lee, 2003]. [Zhao and van de Panne, 2005] used a control pad and interactive controls to control a 3-D humanoid character during diving and skiing. Most recently, [Joe Laszlo, 2005] used predictive feedback to determine the best control strategy for a 2-D character. Our toolkit allows the interactive control of 3-D characters through a combination of pose-based control, reactive controllers and script-based controllers.

Kinematic motion can also be parameterized in order to approximate impacts from external forces. [Arikan et al., 2005] synthesized new motion from a set of motion data of people being push as a reaction to external forces in an interactive environment. [KangKang Yin and van de Panne, 2005] created a data-driven model to simulate the effects of external forces.

[Liu et al., 2005] synthesized motion that adheres to dynamic constraints by using optimization and enforcing linear and angular momentum constraints. Ballistic motion is handled through optimization in [Liu and Popović, 2002] and [Fang and Pollard, 2003].

Many commercial dynamics-based systems have been developed that simulate passive rigid bodies, such as [Kačić-Alesić et al., 2003]. Of relation to our work are those that focus on dynamics and the usage of control mechanism for humanoid animation. A limited number of commercial systems support the creation of physically animated motion. Havok Behaviors [Havok Inc., 2007] and Massive Prime [Regelous, 2005] provide support for physically tracking existing animation (such as from motion capture) and using simple passive control ("rag-doll") when the motion diverges sufficiently from the target trajectory. NaturalMotion's Endorphin [NaturalMotion Ltd., 2007] goes further by providing a variety of parameterized controllers to perform simple actions such as standing, jumping and tackling. While none of these commercial products allow the user to design new controllers, DANCE [Shapiro et al., 2005], a research system, does support the creation of dynamic controllers, but by providing a low-level C++ interface. Our work differs from these in that we are providing a toolset and interface for creating entirely new dynamic controllers.

Several commercial and research systems are available to assist the development of controllers for physical and physically simulated robots, for example, the Webots platform [Michel, 2004] and the Microsoft Robotics Studio [Microsoft Corp., 2007]. Both provide physical simulation and support legged mechanisms. Our work differs in that we focus on the development of controllers for animated characters, not robots. In this respect, we have the freedom to change various aspects of the virtual environment. For example, we can change the presence or effect of gravity, allocate different collision geometry during the simulation, and so forth. In addition, our sensors provide unrestricted knowledge of the state of the system, and are not limited to information gathered by sensors that could be realistically implemented (e.g., range finders, touch sensors, incremental encoders for wheels, etc.).

3 Pose-Based Control

Keyframing is a popular kinematic technique that can automatically generate sequences of animation from a small set of user-described poses. Physically simulated characters can also use keyframed poses by treating the keyframe as the desired pose and using proportional derivative (PD) control to drive the character to the desired position. Our dynamic poses are defined as a 3-tuple as:

$$\Phi = (\psi, \sigma, \delta) \tag{1}$$

where ψ is the set of desired pose orientations for all the character's joints (except for the root joint which is not actuated), σ is the set of gain values, and δ is the set of damping values.

We obtain the amount of force, τ_p applied to each connected body of our character as derived from the pose as:

$$\tau_p = k_s(\theta - \theta_d) - k_d(\dot{\theta}) \tag{2}$$

where τ_p is the torque to be applied to the body. θ represents the current angle between joints, θ_d is the desired angle, k_s is the gain constant, k_d is the damping constant and $\dot{\theta}$ is the rate of change of the joint angle. The gain constant is determined by:

$$k_s = k_{sj} k_{sg} k_{sp} \tag{3}$$

where k_{sj} is the gain of the joint, k_{sg} is a global gain term and k_{sp} is the gain of the desired pose. The k_{sp} term is used to describe how quickly and how strongly the character should attain the pose.

In order to maintain joint limits, we also apply exponential spring forces, τ_e to the bodies if the joints have passed their limits:

$$\tau_e = k_{se} e^{(k_{se}(\theta - \theta_{limit}) - 1)} - k_{de} \dot{\theta} \tag{4}$$

where k_{se} is the gain constant for the exponential springs, θ_{limit} is the joint limit, k_{de} is the damping term for the exponential springs.

3.1 Pose Interface

The interface for keyframed pose creation is straightforward. Animators can set the position of the joints through inverse kinematics (IK) or by explicitly specifying joint angles. In addition to specifying individual poses, animators can specify sequences of poses that automatically transition based on particular events, as shown in Figure 2. Events that can trigger transitions between poses include timer-based events, collision-based events, or sensor-based events. In addition, the user can use the controller scripting language described in Section 5 to test and implement transitions as well.

4 Modified Physics Environments

One of the most difficult goals to achieve for dynamic characters is balance. A number of different balancing strategies exist for humanoid characters [Kudoh, 2004]. For example, a character can attempt to balance by keeping its feet on the ground and moving the rest of the body to adjust the center of mass (CM). Alternatively, a character can maintain balance by ensuring that the zero moment point (ZMP) trajectory stays within a subset of the convex hull of the support polygon [Tak et al., 2000]. In addition, there are other strategies that allow balance by stepping [Vukobratovic et al., 1990]. For large perturbations, however, many of these balancing strategies fail. Since we are not always able to model the impressive balancing ability of the human body, it can be desirable to modify the constraints of our animation system in order to better accommodate our target animation.

4.1 Reduced Physical Dimensions and Forces

We allow the animator to change the virtual environment in order to make the design of controllers easier. For example, we can immediately constrain the forces affecting the character to a particular axis, as well as reduce or eliminate accelerations (and thus velocities) in other directions.

In Figure 3, we use a pose controller to raise the hands of a character to block an oncoming object. Without the use of a balancing controller, the character falls down due to the momentum created from both the movement of the character's arms and the impact of the object. In the same figure, we eliminate forces that impact the root body of the character, thus keeping the character upright during contact. Such changes can reduce the realism of the final animation. However, it is often desirable to achieve a particular effect during animation, more so than it is to achieve full physical validity.

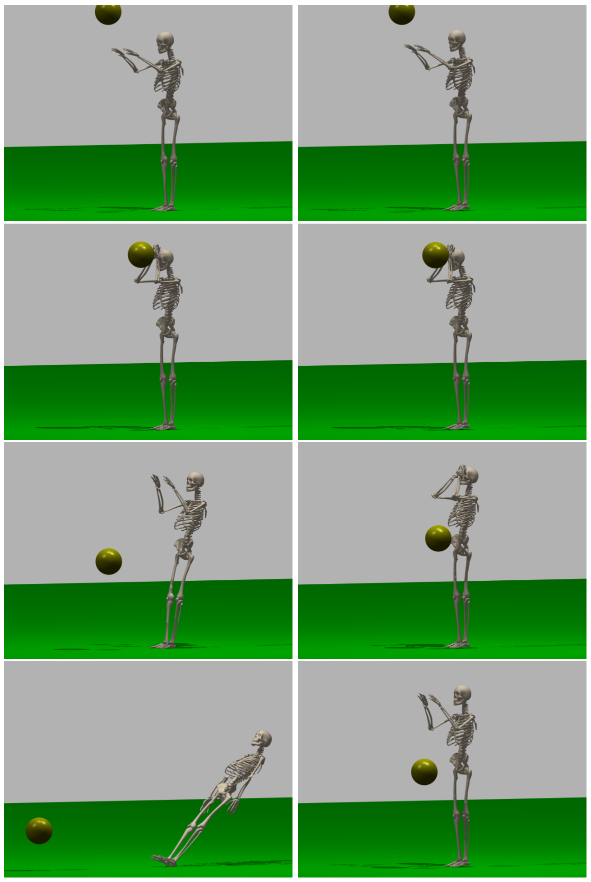

Figure 3: *Physical forces can be modulated on a character based on certain events. The left character uses a pose controller blocks the ball, causing the character to fall. For the character on the right, forces on the hip are disabled for a short period of time after contact, allowing the character to remain upright. However, forces are still applied to the remainder of the character and the impact can be seen on the upper body.*

The effect of reducing physical forces on a simulated character can change dramatically the ability of an animator to create effective dynamic controllers. For example, walking in 2D is much easier to do than in 3D and can even be accomplished for limited steps by interactively selecting the proper pose as shown by [Laszlo et al., 1996] and in our accompanying video. A 2D character is statically balanced and cannot fall sideways. By constraining the forces orthogonal to the facing direction of our simulated character, we can greatly simplify the development robust walking controller. In addition, the constraint forces that keep the character balanced can be disabled when a different task that requires 3D movement is needed, freeing the character to move in 3D again.

4.2 Modified Collision Environment

In addition to changing the forces on the character, the character's collision geometry can be changed during particular times of the simulation. For example, in Figure 4 we give our character long skis in place of normal sized feet. This makes the process of balancing easier due to an enlarged support polygon. At the same time, this change would make movement more unrealistic and physically invalid. However, the animator can retain control over the animation by restoring the normal-sized feet during the simulation via

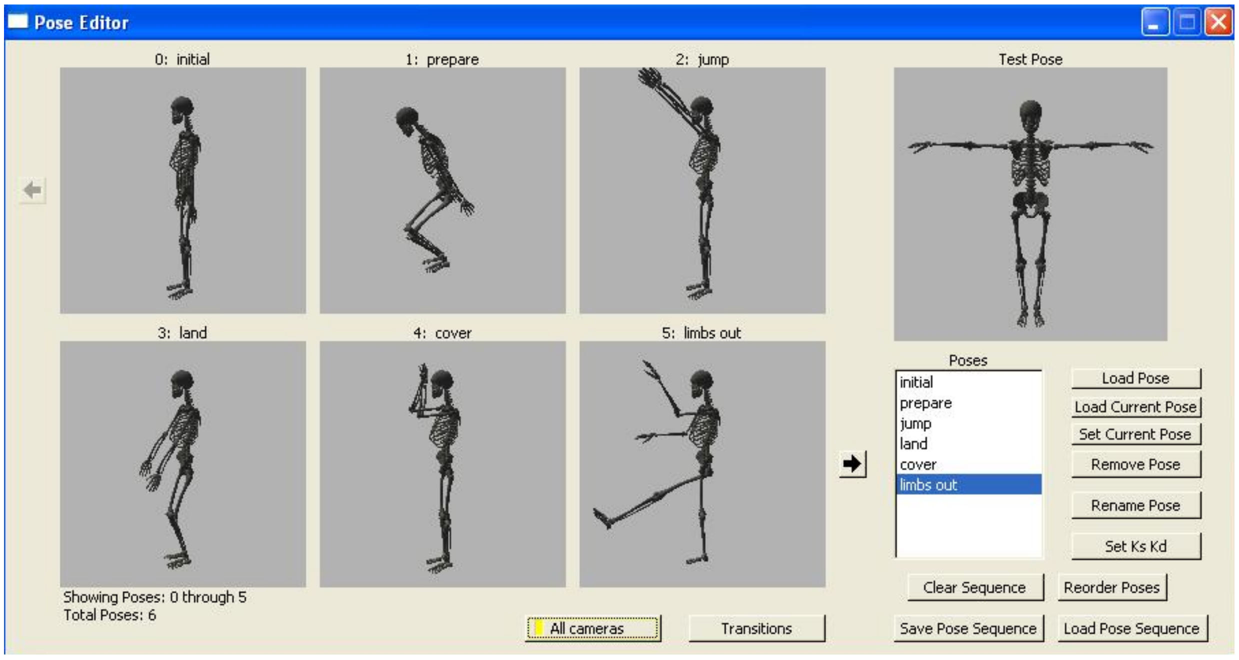

Figure 2: *An animator can keyframe poses and place them in sequence. Transitions between poses are handled through simple event based transitions or through script code.*

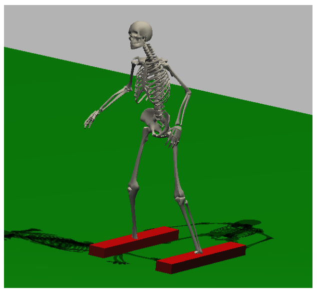

Figure 4: *The toolkit allows the user to change the physical characteristics in order to make complex tasks more easy. Here, a character uses ski-like feet in order to promote balance.*

controller scripts.

5 Controller Scripting

Dynamic controllers can be designed through a combination of pose-based sketching as well as through scripting with a controller-centric scripting language.

We use scripting functions that allow the designer to access various aspects of the sensors. Sensor-based information includes the location of the character's limbs, presence of other objects in the environment, whether or not the character has been impacted by another character or object, and so forth.

The presence of a scripting language, as opposed to writing the controllers in a compiled language such as C++, means that users of the system do not have to know low-level programming constructs. Scripts can be readily written, changed, tested, and then modified according to the performance of the animation.

Although not novel to this work, the presence of a scripting language allows much of the complicated balancing and movement strategies to be abstracted for the end user. Experts can write parameterized functions that accomplish simple tasks to be used by animators. For example, a controller that handles graceful falling might require that the upper body of a character is facing towards (or directly away from the ground). In order to orient the upper body of a character, the script writer must know which way the character is facing, how quickly the body can turn, which direction to turn and so forth. By abstracting this logic, an animator can use this functionality without needing to understand how this is accomplished.

An example controller, provided in Appendix A, illustrates that the toolkit allows useful behaviors to be obtained from relatively simple scripts.

6 Interactive and Reactive Control

Interactively controlling physically-based characters is also difficult. The difficulty stems from both the high number of parameters needed to control an interactive character as well as the difficulty in specifying proper parameters to accomplish meaningful movements.

In our system, a user may instruct the character to perform specific tasks by selecting a pose-based controller from our interface. These pose-based controllers can be simple keyframed poses, as described in Section 3, script-based controllers as described in Section 5, or other autonomous controllers written using C++. Script-based controllers can include modified physics instructions, such as disabling or enabling lateral forces, which can assist the user in controlling the character. With a 2D character or a constrained 3D character, keyframed poses can be used in sequence to achieve a walk cycle, throw punches or kicks, hop, lean backwards, pick up objects and so forth.

Users can toggle between interactive and autonomous control. We achieve this flexibility by allowing the user to combine controllers that perform simple, posture-oriented movements and those that perform complex control strategies such as balance. For example, we have developed a set of falling, balancing and protective strategies for 2D and 3D characters which can be activated interactively at any time. This incorporation of autonomous control gives the character reactive skills that allow it to recover gracefully from various undesirable positions and situations, such as being prone or supine on the ground, or under attack from another character.

This division of control layers also allows users from different areas to contribute towards a more robust, interactive character. Animators and lay users can specify keyframed poses and interactively control the characters, while more technical users can contribute reactive controllers and better control algorithms.

Thus, we can control our interactive character with three different layers of control: 1) a keyframed pose control, 2) a scripting layer that combines keyframes with sensor information, and 3) a layer of reactive control that can be activated during physical interaction or imbalanced states.

Figure 5 demonstrates the use of interactive control of a 3D character. Please refer to our accompanying video to see our system in use for this character, as well as for walking and recovering with a 2D character.

7 Discussion and Conclusion

Developing plausible motion for animated characters is difficult. There are a variety of different techniques that are used to control interactive characters including the use of motion capture, dynamics and optimization. However, no currently known technique can perform a complete range of motions while interacting dynamically with the environment.

Developing dynamic controllers for autonomous behavior is also a very difficult task. Techniques range from hand-coding controllers based on intuition, to borrowing techniques from robotics and biomechanics, to tracking motion capture. Most dynamic controllers for animated characters are either simple or brittle. The simple controllers can only perform basic tasks, while the brittle controllers can only succeed under specific environments.

Our application seeks to fill a void in the capabilities of animators and researchers to create character animation. This gap in capability can be seen in the amount and quality of tools available for kinematically-driven character animation versus that for dynamically-driven character animation. Consider that keyframing is a popular technique for animating kinematic characters. It is powerful because it allows the animator to express a nearly unlimited variety of movement and behavior given enough time and effort on the part of the animator. In addition, the barrier to entry for designing keyframed animation is relatively low. Thus, a large number of animators can develop a wide range of motion and behavior. On the other hand, no such tool or technique yet exists for the development of dynamic control and interactive physical control. Therefore, the barrier to entry for the development of dynamic controllers is high and left in the hands of the experts in robotics, graphics, artificial intelligence and similar technical fields. This is a major obstacle in the adoption and development of dynamic control in character animation, a field often populated with artistically-oriented rather than technically-oriented people.

Acknowledgements

This work was partially supported by the NSF grant CCF-0429983. We would like to thank Sung-Hee Lee, Demetri Terzopoulos and the anonymous reviewers for their comments. We would also like to thank Intel Corp., Microsoft Corp., Ageia Corp., and ATI Corp. for their generous support through equipment and software grants.

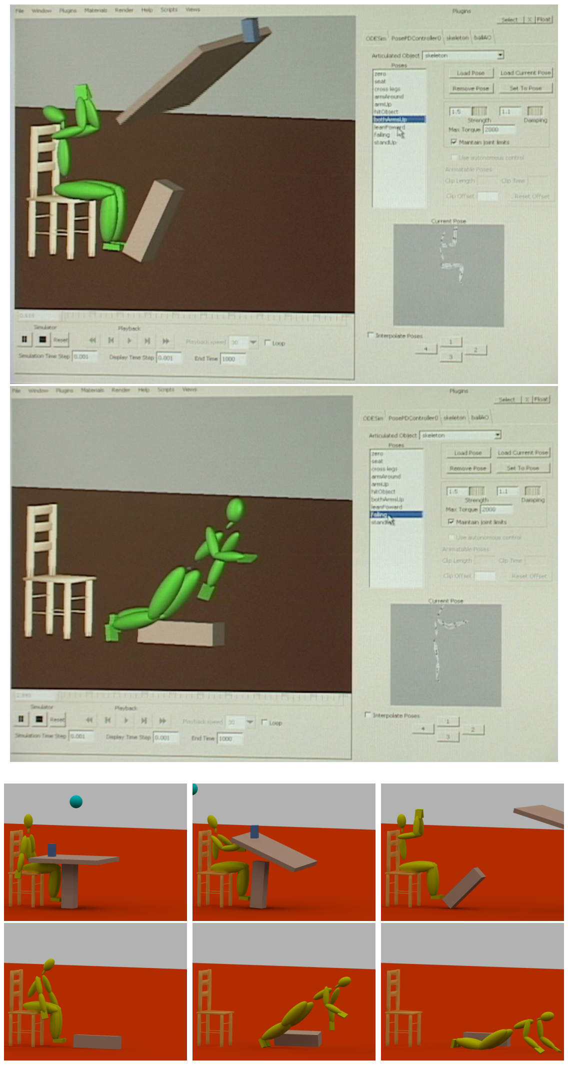

Figure 5: *The user controls the 3D character interactively (top). The controllers are chosen at a timing indicated by the user, and can be simple pose based controls or complex, scripted controllers. Several frames of the resulting animation are shown below.*

Appendix A: Example Control Script

A simple script is provided to illustrate building a controller using the described toolkit. As shown in Figure 1, (and the supplemental video) inverse kinematics is used to position the ball toward the rim. This script is applied after the character has jumped toward the basketball goal. `setIKTarget()` both solves the inverse-kinematics problem and updates the target pose using the computed joint angles.

Overall pose control is managed via the `setTargetPose()` command. Simple sensory information is provided by the functions `getPosition()` and `isColliding()`.

The character's simulated hand is not modeled accurately enough to grip the ball, so dynamic attachments (`attachLink()` and `detachLink()`) are used to simulate "palming" the ball.

The controller is implemented as a Python [van Rossum and Drake, 2006] class:

```
class DunkController:
  def start(self):
    skeleton.setTargetPose("PrepareDunk")
    skeleton.attachLink("RightArm", "Ball")

  def step(self):
    # As long as we have the ball,
    # continue to attempt the basket.
    if skeleton.isLinkAttached("RightArm", "Ball") :
      rimPos = getPosition("Rim")
      skeleton.setIKTarget(rimPos)

    # When the ball or the arm touches the rim,
    # the dunk attempt is finished,
    # so release ball and try to land.
    if isColliding("Ball", "Rim") \
      or isColliding(skeleton, "Rim") :
      skeleton.detachLink("RightArm", "Ball")
      skeleton.setTargetPose("Landing")
```

Two methods are required by the toolkit. `start()` is called once to initialize the controller when it is first applied to a character during a simulation. `step()` is subsequently called with each simulation time-step and provides the main control functionality.

References

Arikan, O., Forsyth, D. A., and O'Brien, J. F. (2005). Pushing people around. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 59–66, New York, NY, USA. ACM Press.

Faloutsos, P., van de Panne, M., and Terzopoulos, D. (2001). Composable controllers for physics-based character animation. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 251–260.

Fang, A. and Pollard, N. (2003). Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):417 – 426.

Havok Inc. (2007). Havok behavior. http://www.havok.com/.

Hodgins, J., Wooten, W., Brogan, D., and O'Brien, J. (1995). Animating human athletics. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 71–78.

Joe Laszlo, Michael Neff, K. S. (2005). Predictive feedback for interactive control of physics-based characters. In *Proceedings of Eurographics 2005*.

Kačić-Alesić, Z., Nordenstam, M., and Bullock, D. (2003). A practical dynamics system. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pages 7–16. Eurographics Association.

KangKang Yin, D. K. P. and van de Panne, M. (2005). Data-driven interactive balancing behaviors. In *Pacific Graphics '05*.

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*, pages 32–40, New York, NY, USA. ACM Press.

Kudoh, S. (2004). *Balance Maintenance for Human-Like Models With Whole Body Motion*. PhD thesis, University of Tokyo.

Laszlo, J., van de Panne, M., and Fiume, E. (1996). Limit cycle control and its application to the animation of balancing and walking. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 155–162.

Laszlo, J., van de Panne, M., and Fiume, E. (2000). 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. ACM Press/Addison-Wesley Publishing Co.

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.

Liu, C. K. and Popović, Z. (2002). Synthesis of complex dynamic character motion from simple animations. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 408–416, New York, NY, USA. ACM Press.

Mandel, M. (2004). Virtual humans. Master's thesis, School of Computer Science.

Michel, O. (2004). Webots(tm): Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):39–42.

Microsoft Corp. (2007). Microsoft robotics studio. http://msdn.microsoft.com/robotics/.

NaturalMotion Ltd. (2007). Endorphin. http://www.naturalmotion.com/.

Neff, M. and Fiume, E. (2002). Modeling tension and relaxation for computer animation. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 81–88, New York, NY, USA. ACM Press.

Neff, M. and Fiume, E. (2003). Aesthetic edits for character animation. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 239–244, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

Neff, M. and Fiume, E. (2004). Methods for exploring expressive stance. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 49–58, New York, NY, USA. ACM Press.

Oore, S., Terzopoulos, D., and Hinton, G. E. (2002). Local physical models for interactive character animation. *Comput. Graph. Forum*, 21(3).

Pollard, N. S. and Zordan, V. B. (2005). Physically based grasping control from example. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 311–318, New York, NY, USA. ACM Press.

Regelous, S. (2005). *Crowd and Group Animation*. SIGGRAPH Course Notes.

Shapiro, A., Faloutsos, P., and Ng-Thow-Hing, V. (2005). Dynamic animation and control environment. In *GI '05: Proceedings of the 2005 conference on Graphics interface*, pages 61–70.

Shapiro, A., Pighin, F. H., and Faloutsos, P. (2003). Hybrid control for interactive character animation. In *11th Pacific Conference on Computer Graphics and Applications*, pages 455–461.

Smith, R. (2003). *Intelligent Motor Control with an Artificial Cerebellum*. PhD thesis, Department of Electrical and Electronic Engineering.

Tak, S., young Song, O., and Ko, H.-S. (2000). Motion balance filtering. *Computer Graphics Forum*, 19(3).

van de Panne, M. and Lee, C. (2003). Experiments with interactive dynamics. In *Proceedings of the 14th Western Computer Graphics Symposium*.

van Rossum, G. and Drake, F. (2006). *Python Reference Manual*. Python Software Foundation.

Vukobratovic, M., Borovac, B., Surla, D., and Stokic, D. (1990). *Biped Locomotion: Dynamics, Stability, Control and Application*. Springer.

Wrotek, P., Jenkins, O. C., and McGuire, M. (2006). Dynamo: Dynamic, data-driven character control with adjustable balance. In *Proceedings of the Sandbox Symposium on Video Games*. ACM, ACM.

Yang, P.-F., Laszlo, J., and Singh, K. (2004). Layered dynamic control for interactive character swimming. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 39–47. ACM Press.

Yin, K., Cline, M. B., and Pai, D. K. (2003). Motion perturbation based on simple neuromotor control models. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 445, Washington, DC, USA. IEEE Computer Society.

Zhao, P. and van de Panne, M. (2005). User interfaces for interactive control of physics-based 3d characters. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 87–94, New York, NY, USA. ACM Press.

Zordan, V. B., Celly, B., Chiu, B., and DiLorenzo, P. C. (2004). Breathe easy: Model and control of simulated respiration for animation. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 29–37. ACM Press.

Zordan, V. B. and Hodgins, J. K. (2002). 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. ACM Press.

Zordan, V. B., Majkowska, A., Chiu, B., and Fast, M. (2005). Dynamic response for motion capture animation. *ACM Trans. Graph.*, 24(3):697–701.