UNIVERSITY OF CALIFORNIA
RIVERSIDE

Interactive Performance for Movies and Video Games

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Computer Science

by

Marc Anthony Soriano

December 2009

Thesis Committee:

Doctor Victor B. Zordan, Chairperson
Doctor Eamonn Keogh
Doctor Christian Shelton

The Thesis of Marc Anthony Soriano is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I would also like to thank everyone from my lab, the Riverside Graphics Lab, especially Dr. Victor B. Zordan and Muzaffer Akbay for their advice, support, and willingness to test out my system. Additionally, I would also like to thank Steve Suh and Roby Atadreo for their technical support for interface, graphics and display. Without their help and motivation, this may have never happened.

Contents

List of Figures **vi**

1 Introduction **1**
1.1 General System Constraints 3
1.2 Capture Methods and Previous Work 4

2 Kinematic-Dynamic Continuum **6**
2.1 Kinematic Control 7
2.2 Dynamic Control 10
2.3 Integrated Control 11

3 Balance **14**

4 System Details **18**
4.1 Character Model and Network Setup 18
4.2 Motion Capture System 20
4.3 Dynamics Simulation 22
4.4 Visual Feedback 23

5 Examples and Results **26**
5.1 Ballroom 26
5.2 Boardroom 27
5.3 Punching Bag 27
5.4 Boulder 28

6 Limitations and Future Extensions **31**

7 Conclusion **35**

Bibliography **36**

List of Figures

2.1 Data flow of the system. Starting with the actor being picked up by the motion capture system, the captured pose ($\hat{p}$) is sent to the Kinematic Controller (KC) and Dynamic Controller (DC). The forces generated by KC (F) are augmented by the balance controller (BC) and the current character state, p. The resulting forces (F') are then combined with the toques (τ) generated by DC into the simulation. The resulting simulated character pose (p) is then fed back into DC and KC for the next time step, as well as sent to the graphics display for rendering. 8

2.2 The character model used in our system. A distribution of how the kinematic forces (yellow) and dynamic forces (blue) are applied across the 15 separate body parts of the character, assuming both feet are supplying support . . . 13

3.1 How we determine the balance term. Given the support polygon (orange) generated by the support bodies (blue), we calculate the center of mass (green) and its velocity (black). The balance term $\beta(t)$ is calculated based on the distance of the center of mass and the intersection of the support polygon and center of mass velocity (red). 16

4.1 Data Flow of the System. (Starting left going clockwise) (1) The actor's motion is captured and converted into skeleton poses (2) The posture information is converted in the simulation into joint torques and kinematic forces to allow interaction with the physically simulated world (3) The actor views the character's modified motion, and reacts accordingly. 19

4.2 Character state network packet breakdown by bytes: [0-6] ID of the character or object, [7] number of body parts n, [8-63] p_0, [64-119] p_1, etc. Each p_k is represented as a septuple of doubles that hold the cartesian position (x, y, z) and quaternion orientation (qw, qx, qy, qz) of that body part. 21

4.3 Projector display used to give the actor visual feedback 23

5.1 Character manipulates a ball with his hands and arms to prevent it from rolling into the containment area. He does this by rolling the ball off of the ramp 27
5.2 Character kicks a plank aside, and proceeds to catch and carry the larger board 28
5.3 (Top Strip) Character pushes the punching bag with his full body (Bottom Strip) Character then grabs and hangs off of the punching bag 29
5.4 (Top Strip) Character dodges the oncoming boulder and is slightly nicked by it (Bottom Strip) After being knocked down by the next boulder, the character stands back up 30

6.1 Incorporating a head mounted display (HMD) would allow the actor further immersion into the virtual world 32

Chapter 1

Introduction

Throughout the brief history of 3D animation in cinema and video games there have been two primary methods of producing realistic character animations: motion capture and physical simulation. As computer generated worlds have become more realistic and interaction between actors and their environment becomes more important, the demand of integrating realistic motion capture performances into physically simulated worlds has increased. A major bottleneck to achieving this full integration comes in the form of the interaction and subsequent reaction between the actors prerecorded motion and objects within the virtual world. While many methods attempt to retarget said motions by using physical simulations or key framing, the live action recordings and virtual environment remain disjoint entities, and synchronization between the two usually require an additional, offline procedure.

We introduce a hybrid method in the form of a physically simulated, real-time

human performance in order to generate motions with rich interactions in complex scenes and scenarios. The goal is to remove the disjoint between the motion capture and physical simulation stages of character animation while retaining a level of realism acceptable for use in movies and video games. This is done by through a weighted combination of two different animation signals: the pure, kinematic motion capture obtained from the actor, and the dynamic simulation that dictates how the character moves and interacts in the virtual world. The kinematic animation signal is converted into a dynamic signal that follows the source motion capture as closely as possible and is added to the dynamic simulation with weights that vary across body parts as a function of time. The result is an full-body, actor controlled, physically simulated character capable of interaction in a virtual environment in real time.

An important part of this work is the concept of human-in-the-loop control. This places a constraint preventing any motion editing a priori, but also removes any need for a prerecorded motion library since the actor is an active participant in the system supplying the motion. Consequently, this allows for real-time reactions by the actor, producing motions that would be difficult to achieve with prerecorded data. In addition, this not only creates richer motions, it gives some insight on how humans respond to varying outcomes from a single scene as well as how humans respond to many different scenarios.

1.1 General System Constraints

Motion capture has already proven extremely valuable for generation human-like motion by recording motions directly from a human subject. As a popular method for animating characters, many research efforts attempt to adapt, combine, and retarget motion data for scenarios completely different from their original recording conditions. Our approach deviates from this standard by utilizing online, non-recorded, human motion, making a priori motion editing impossible. Our system gives the actor control over the character, allowing him or her to make the decisions and reactions when acting upon other virtual objects.

Fundamentally, integrating real-time performance with a physically simulated character will introduce discrepancies between the real world capture and the resulting virtual simulation. If there is physical interaction between the character and a virtual object, the characters motion must respond to the impact from the interaction and accommodate an appropriate deviation from the performance. In this case, the character movement should remain physically plausible for the duration of the interaction, obey the performance as closely as physically permissible, and return to the performer's motion in a timely and believable fashion once the interaction subsides. Generally speaking, this thesis tackles the challenge of integrating real-time actor performance with simulated physics for complex interaction in a real-time simulated virtual world.

1.2 Capture Methods and Previous Work

Using online human performance to control a virtual avatar has been proposed by many researchers. Although input devices range from video cameras [6], accelerometers [14], foot pressure sensors [18], and full-body motion capture devices [15, 9], the underlying algorithms need to solve the same problem of mapping the performers intention to the characters action. Each method has its strengths and weaknesses. While [6, 14, 18] provide simple but intuitive ways of giving user input, the simplicity of their interfaces requires interpretation, and in some cases preprocessing, of the motions which may deviate radically from the user's original intention. While [15, 9] attempt to address this problem [15] focuses on end effector interaction and [9] uses dynamics to only simulate the global state of the character. As a consequence, neither achieves full-body, human-like response and interaction. In contrast, our system gives the actor full-body control of the character as well as allows for fully-body physics-based interaction with his or her surroundings, yielding more realistic responses in dynamically varying environments.

Recent work has been done to make characters more responsive to their surroundings, but such responsiveness is extremely limited. In approaches that do not include physics, motion transitions and blending are used to emulate responses to interactions [4, 19]. Such systems are specialized in the type of responses they can generate and require examples from a database to make their decisions. In approaches that do use physics, several researchers use a setup of transitioning from motion capture to physical simulation in the event of an interaction, and then attempt to go back to motion capture when

possible [21, 17, 2, 16, 12]. Similar to the non-physics approaches, these and related techniques [22, 1] are very specific in the types of responses they support by modifying existing motions. Others have also investigated similar works using space-time optimization [20]. Despite this variety of methods, they all fall short in their reliance on prerecorded data to drive their motions. The level of interactivity and reaction is limited to what is available in the database, and can appear ignorant to dynamically changing scenes. By having a live human actor supply motions to our system, our character appears capable of intelligently responding to varying environmental conditions by allowing the actor to assess, react, and control what character does.

To make characters truly human-like, they have to be able to assess, respond, and plan [8]. Recent research in Artificial Intelligence, while nowhere near emulating human-like intelligence, has primarily focused on animating characters through examples. Such methods attempt to control characters by searching through large motion databases structured similarly to motion graphs [10]. A limitation to this method is the resulting computational complexity tied to searching the graph to any substantial depth. This problem has been addressed using different routines to extend the planning horizon [13, 3, 7, 11]. In contrast, our approach sidesteps the planning problem by exploiting the intelligence and (relatively) long-term planning associated with the performer and the performance, and removing the need for any prerecorded motions. Innately, we rely on the performer to plan and trust that if we can remain faithful to the motion performance, our character will reflect the intelligence of the actor.

Chapter 2

Kinematic-Dynamic Continuum

Our proposed hybrid technique does not rely on switching between kinematic and dynamic simulations as in [22], but rather forms a continuum between the two. Recent work [1, 5] has begun to propose hybrid systems that use dynamics to achieve key frame constraints. In addition to using key frames as constraints, we convert those key frames into supporting forces to make the resulting simulation more closely follow the kinematic system. We have created a Kinematic-Dynamic (KD) continuum that uses the combination of dynamics and kinematics to allow the character to be physically responsive to virtual forces while still remaining faithful to the actor's input.

The domain of the KD controller is defined as a time varying continuous space that combines two distinct interpretations of the source motion performance, $p\hat{(}t) = (p_1\hat{(}t)$, ... , $p_n\hat{(}t))$, where $p_k\hat{(}t)$ represents a composite vector of the global position and rotation of body part, k. The first interpretation treats $p\hat{(}t)$ as a pure kinematic pose and attempts

to match it using external forces. The second interpretation treats $p(\hat{t})$ as a desired point that the physical model, $p(t)$, attempts to follow using internal joint torques. We treat the latter as a forward simulation tracking $p(\hat{t})$ via a control routine as in [21, 17, 1]. Our KD framework uses two naive controllers rather than more complex, coordinated controllers with the understanding that the intelligent, high-level control is supplied by the actor in the loop. (See Figure 2.1 for a full system diagram)

2.1 Kinematic Control

To ensure the kinematic controller is integrated into the dynamic simulation, we treat the kinematic controller as a special case of the dynamics. Without loss of generality, motion capture playback can be interpreted as a set of accelerations for each body that change over time. By taking into account mass and inertia, these accelerations can be converted into corresponding forces needed to move each body. External forces such as gravity or collisions can also be accounted for by simply adding those forces into the calculation. While our general outline is very similar to inverse dynamics, we pose a simpler problem which is to derive a set of forces, $f_k(t)$, which not only move the character despite gravity and contact forces, but also leads the character to follow the motion capture.

We define $f_k(t) \in \mathbb{R}^3$ as an external force applied to body k. Unweighted, $f_k(t)$ pulls the simulation to the global position of the kinematic source, effectively overriding external forces such as gravity and contact collisions. To prevent an excessively stiff and unresponsive simulation, each $f_k(t)$ is scaled by a corresponding $y_k(t) \in [0,1]$. In the KD

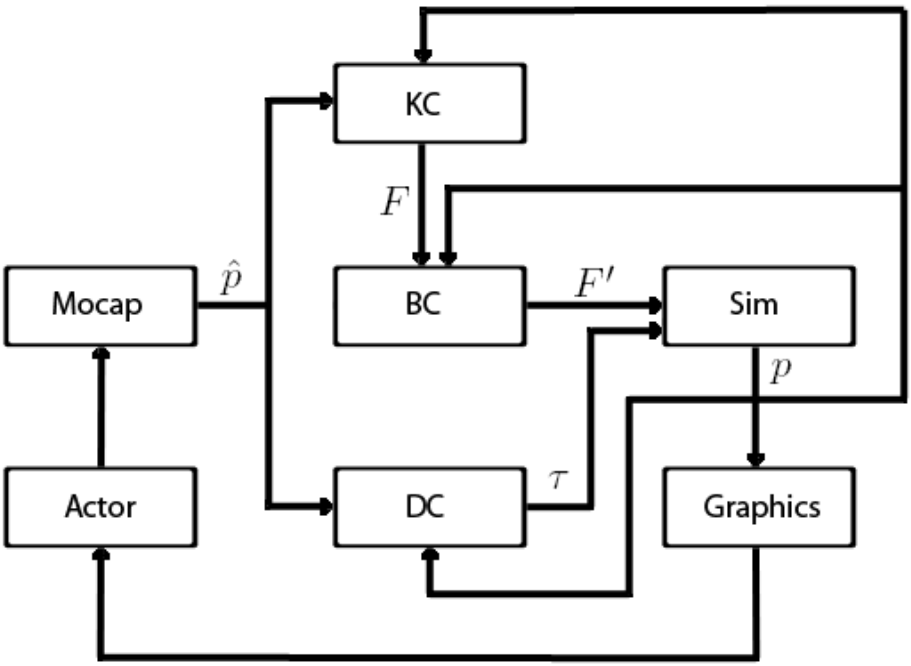

Figure 2.1: Data flow of the system. Starting with the actor being picked up by the motion capture system, the captured pose ($\hat{p}$) is sent to the Kinematic Controller (KC) and Dynamic Controller (DC). The forces generated by KC (F) are augmented by the balance controller (BC) and the current character state, p. The resulting forces (F') are then combined with the toques (τ) generated by DC into the simulation. The resulting simulated character pose (p) is then fed back into DC and KC for the next time step, as well as sent to the graphics display for rendering.

continuum, we define the vector $\mathbf{y(t)} = (y_1(t), ..., y_n(t))$ as purely kinematic when $\mathbf{y(t)} = \mathbf{1}$ and purely dynamic when $\mathbf{y(t)} = \mathbf{0}$. Effectively, $y_k(t)$ reduces the influence of the force $f_k(t)$ on body part k, with the amount of that influence varying both with time and from body to body. Our choice for $\mathbf{y(t)}$ is discussed in the Integrated Control section.

To compute $f_k(t)$, we use a Cartesian-based PD-servo as

$$f_k(t) = k_p F(||\hat{p(t)} - p(t)||) \frac{\hat{p(t)} - p(t)}{||\hat{p(t)} - p(t)||} - d_p \dot{p(t)} \tag{2.1}$$

where $p(t) = (p_1(t), ..., p_n(t))$ is the global position of all body parts of the simulation, $\hat{p(t)}$ is the global position of all body parts of the actor, and gains k_p and d_p are manually tuned constants. We employ a modified logistic function to saturate the kinematic forces so they do not go unbounded

$$F(x) = \frac{1}{1 + e^{-sx}} - \frac{1}{2} \tag{2.2}$$

where the value of the term s controls how quickly the controller saturates to the maximum. $F(x)$ limits the bounds of the forces to avoid undesirably large influences and assist in keeping the system stable. The values of k_p and d_p are chosen such that these values should be large enough to become the dominant influences (overcoming gravity and contact forces) without leading to instability.

In the limit, as the error between $\hat{p(t)}$ and $p(t)$ reaches infinity, it is apparent $f_k(t)$ would be incapable of matching the actor's input in a graceful manner. This is especially obvious in the upper body, where body parts deviate more due to both the lack of ground forces to keep them in place and their distance from body parts that have ground forces applied to them. This issue is addressed in the Integrated Control section.

2.2 Dynamic Control

Our dynamic controller follows the performance by tracking the joint angles of the motion capture data using internal joint torques. We perform inverse dynamics by explicitly calculating the joint torques needed to actuate the character to match the actor's pose. However, since the joint tracker does not control the root body, the character falls over very easily without any sort of coordination for balance. Rather than focusing on this coordination using a balance controller to apply additional joint torques, we employ this tracker in conjunction with $f_k(t)$ (see Kinematic Control) and transform the character from a purely dynamic simulation to a hybrid that combines both balance and responsiveness in a simple fashion.

Firstly, the dynamic controller converts $p\hat{(}t)$ into its joint space equivalent, $q\hat{(}t) = (q_1\hat{(}t), ..., q_m\hat{(}t))$, where each $q_i\hat{(}t) = (q_1(t), ..., q_m(t))$ represents the joint angle between two of the actor's body parts. Similarly, $p(t)$ is converted into its joint space equivalent, $q(t)$, where $q_i(t)$ represents the joint angle between two of the character's body parts. The joint tracker then computes joint torques based on the tracking error between $\hat{q} - q$ as well as the joint velocities $\dot{q}$ as follows

$$\tau(t) = k_q F(||q\hat{(}t) - q(t)||) \frac{q\hat{(}t) - q(t)}{||q\hat{(}t) - q(t)||} - d_q \dot{q\hat{(}t)} \tag{2.3}$$

which follows the pose $q\hat{(}t)$ based on the maximum value k_q and damping d_q. The result is an array of joint torques, $\tau(t) = (\tau_1(t), ..., \tau_m(t))$, where $\tau_i(t)$ is applied to the character to actuate joint i. Similar to Equation 2.1, $F(x)$ controls how quickly the controller saturates to k_q by using Equation 2.2. This function allows the joint tracker to resist against deviations

from the performance with a substantial torque even for small errors, but, similar to human joints, are limited in their maximum physical strength.

2.3 Integrated Control

The two KD controllers both guide the character to follow the performance, but in different ways. The forces computed in Equation 2.1 will maintain global Cartesian positions and help to follow the actor's original performance, but results in stiff physical interactions that resist or even ignore external forces in an unrealistic manner. The joint torques calculated in Equation 2.3, on the other hand, will maintain local joint orientations and help follow the actor's original pose, but lack the coordination to maintain balance. The Cartesian forces give the actor more precise control and can aid in balance control while the joint torques provide interactivity and physical realism in the virtual environment. Our technique combines these two complementary inputs in order to get the best of both.

While the Kinematic Control is useful for precise movement and balancing the character, the KD continuum limits its use based on the proximity of each body part to the ground. The goal is to allow the joint tracker to interact as much with the virtual environment as possible while still maintaining the control, balance, and style inherent in the original motion capture. When the character is in contact with the ground, we draw from the assumption that the ground supplies reaction forces that lead to analogous corrections embedded in the forces from Equation 2.1. These forces, $f_k(t)$, make up for the under-actuation of the root when the joint tracker was used alone.

Over time, the root body changes depending on what body parts are in contact with the ground. With our setup, we detect and account for the fact that there can be several root bodies at any given time. As such, we proposed a simple temporally changing allocation rule to combine the KD signals and incorporate the following value for $y_k(t)$

$$y_k(t) = \alpha^{j(k,t)} \tag{2.4}$$

where $\alpha \in [0,1]$ is the discount applied for each subsequent body in the shortest chain between the body and the ground. $j(k,t) \in [0,n) \cup \infty$ is the count of body parts from the current body k to the nearest support body, which changes over time. The support bodies which are in contact with the floor are assigned to have $j(k,t,) = 0$, which results in $y_k(t) = 1$, and are assumed to be able to resist external contact and follow the data kinematically. All other bodies are discounted based on α and their body counts from the floor. In cases where the character is in the air, $j(k,t) := \infty$ for all k and the character is considered to be airborne, making $\mathbf{y(t)} = \mathbf{0}$. For the examples in these papers, we set $\alpha = 1/2$ and when the character is standing on both feet, the pelvis, which is the third body up from the feet, feels 1/8 of the kinematic forces computed. This result is visualized in Figure 2.2 and the result makes the character more resilient to impacts which hit body parts close to support bodies, makes the upper body more intractable, and allows the system to track well across the entire body.

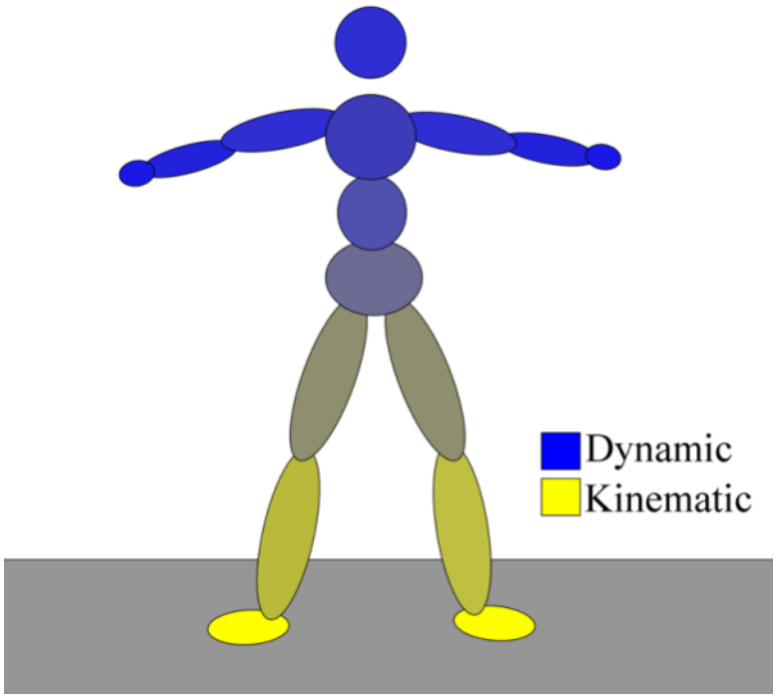

Figure 2.2: The character model used in our system. A distribution of how the kinematic forces (yellow) and dynamic forces (blue) are applied across the 15 separate body parts of the character, assuming both feet are supplying support

Chapter 3

Balance

The system described thus far is sufficient enough to generate Figures 5.1 and 5.2 (discussed in Examples and Results), in that it can deal with relatively small perturbations and external forces. In the case where the character is hit with a large force and falls down or nearly falls down, the kinematic controller is still too strong to produce believable off-balance motions. In addition to scaling the kinematic forces for bodies not in contact with the ground (as in Integrated Control), we also maintain a measure of balance to scale the bodies in contact with the ground. This introduces an additional scaler to the KD continuum that assists in smoothly turning the kinematic controller off or on in cases of extremely dynamic motion.

To calculate the balance term $\beta(t)$, we first compute the center of mass, COM_t, of the character along the bounds of the support polygon, SP_t. Given the center of mass from the previous time step, COM_{t-1}, we can determine the direction the center of mass

is moving and consequently, the point at which the center of mass would cross the support polygon, denoted as b_t, assuming the center of mass maintains its current trajectory. Given these values, we calculate the balance term as such

$$\beta(t) = \begin{cases} 0, & \text{if COM is out of SP} \\ \sin(\frac{\pi}{2} * \frac{||COM_t - b_t||}{th}), & ||COM_t - b_t|| \leq th \\ 1, & \text{otherwise} \end{cases} \tag{3.1}$$

where th is a threshold distance between center of mass and the bounds of the support polygon (see Figure 3.1). th is used to determine how close to the support polygon bounds to start degrading the balance term. The resulting value of $\beta(t)$ is then multiplied into $\mathbf{y}(t)$ to scale the influence of the kinematic control.

As the distance of the center of mass nears the support polygon boundary point, b_t, the balance term smoothly drops to 0, indicating the kinematic control is turned off and the character is in full dynamic control. Conversely, as the center of mass moves towards the center of the support polygon, the balance term saturates to 1, indicating the kinematic control is fully functional, as described in Integrated Control. We give the actor visual feedback on the status of his or her balance by binding $\beta(t)$ to the coloration of the character. A green coloration indicates the character is fulling in balance, a red coloration indicates the character is out of balance, and any gradient between indicates a transition between them. This visual cue is used to alert the actor of an impending fall, allowing him or her to react, either by leaning back into balance, or stepping to catch him or herself.

In more extreme scenarios in which we want the character to fall and the center

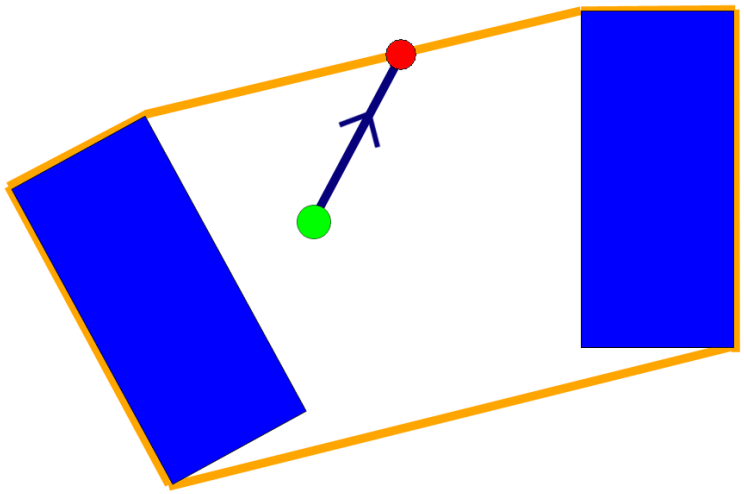

Figure 3.1: How we determine the balance term. Given the support polygon (orange) generated by the support bodies (blue), we calculate the center of mass (green) and its velocity (black). The balance term $\beta(t)$ is calculated based on the distance of the center of mass and the intersection of the support polygon and center of mass velocity (red).

of mass is outside of the support polygon, $\beta(t) = 0$. Consequently, as the character falls out of balance, f_k drops to zero, the character is completely dynamically controlled, and we fix $\beta(t)$ to zero until a user commands otherwise. In the case the actor can move the character back into balance, $\beta(t)$ returns to non-zero values and the system resumes. In the case the character falls to the ground (now purely simulated based on full-body dynamics) we have a setup to allow the actor to resynchronize with the character. Firstly, the system is paused, allowing the actor to lie down on the ground and line up with the character. Secondly, the system is then unpaused with the actor's root position offset to align with the character's root. Lastly, the continuation of the simulation is followed by a smooth linear blend to return the character to kinematic control. This procedure creates an online, performance-based response similar to that described by Zordan et al [22]. To allow the

actor to stand the character back up, $\beta(t)$ is fixed at 1 for a short period of time following the user command signal. In doing so, we avoid undesirable effects resulting from using a static balance measurement in a dynamic action such as getting up.

Chapter 4

System Details

Although we focus on the details of how the simulation works, our system is broken into a loop of three major components: the motion capture system, the KD simulation, and the visual feedback. Data from each component is fed from one component to the next in a cyclic fashion (see Figure 4.1) using either a Transmission Control Protocol (TCP) network connection for remote feeds or a hard-wired display used for local displays. The data sent from component to component is dependent on our character model.

4.1 Character Model and Network Setup

Our character model is broken up onto 15 separate body parts (See Figure 2.2) connected together with ball joints, with the exception of the elbows, knees, and wrists. This model was chosen to give enough complexity to the character to allow for believable full-body human motion while simple enough to avoid slowing down either the simulation

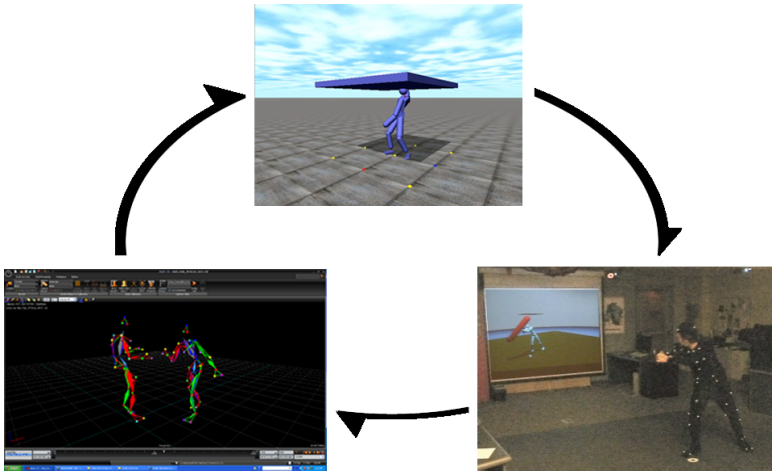

Figure 4.1: Data Flow of the System. (Starting left going clockwise) (1) The actor's motion is captured and converted into skeleton poses (2) The posture information is converted in the simulation into joint torques and kinematic forces to allow interaction with the physically simulated world (3) The actor views the character's modified motion, and reacts accordingly.

frame rate or the overloading the network connection. Ball joints are used to connect most of the character's body parts, with the exception of the elbows and knees, where joint limits are implemented to prevent joint hyperextension. The only other non-ball joints are the wrists, which are fixed and the hands of the character are fused to the forearms. This was a design choice implemented due to the limitations of our motion capture setup and to prevent unnecessarily noisy data.

The data sent from component to component uses a TCP network connection which is a widely used internet protocol that helps to ensure each component is connected and data is received in order. Due to its ease of use for file streams and transfers, the TCP protocol was also chosen to allow for further expansion of our system for online multi-actor interaction in the future.

Data packets contain state data of each character in the scene in the form of an

array of doubles. With our given character model, each data packet takes up $8 + 7*8*n$ bytes of information (see Figure 4.2), where n is the number of body parts of the character. Bytes [0-6] are used to identify which character or object is being described, byte [7] indicates the number of bodies the character has (this limits the character to less than 256 body parts), and the remaining $7*8*n$ represent the positions and rotations of each body part. Starting from byte [8-...], each eight bytes represent a number type cast as a double. Each septuple of doubles represent the cartesian coordinates (x, y, z) and quaternion rotations (qw, qx, qy, qz) of their corresponding body part, p_k. In the cases where multiple characters or objects are in the scene, character packets are appended to one another to allow for a single transmission per time sample. Using this compressed format we are able to send state information from the motion capture to the simulation, or from the simulation to the display at 120 samples per second with a minimal network hit in a scene containing a single character and several objects.

4.2 Motion Capture System

In order to capture the actors movements, we use the Vicon Motion System to track the actors motions. Those motions are read as a cloud of markers, which are then converted into a custom made skeleton using the Vicon Blade real-time solver (for Vicon Motion Systems, www.vicon.com). For every solved frame, the skeleton is interpreted as a series of global positions and quaternion rotations for each body part.

Although most marker sets would most likely suffice, in order to correctly track

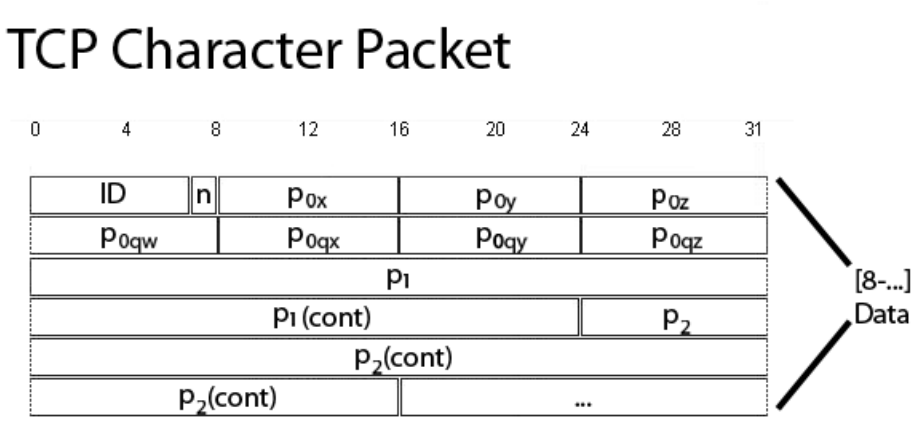

Figure 4.2: Character state network packet breakdown by bytes: [0-6] ID of the character or object, [7] number of body parts n, [8-63] p_0, [64-119] p_1, etc. Each p_k is represented as a septuple of doubles that hold the cartesian position (x, y, z) and quaternion orientation (qw, qx, qy, qz) of that body part.

each body part reasonably there have to be enough markers to prevent tracking errors due to occlusion. The marker set should be versatile enough to allow the actor to move around the capture space, roll or lay on the ground as necessary, and bend and gesture to interact with virtual objects; all with reasonably accurate data from the real-time solver. As such, we place enough markers to cover every possible angle each body part could be seen, while trying to keep the total number of markers low. Major marker emphasis was placed on the head, chest, and waist to allow for realistic head motion for looking, and sufficient actuation of body parts with 3+ body parts attached to them. A corresponding skeletal model is created which associates those markers to specified body parts. The skeleton is tailor made to the actor, approximately matching height, width, and length of each of the actor's body parts. The weights of each body part are determined under the assumption that their density is approximately that of water. Offline, a similar model is generated for

the dynamics simulation to incorporate the inertia of each body part.

Once the markers are read into the motion capture system, the Blade real-time solver uses an inverse kinematics solver to fit the model given the markers. This gives a reasonable approximation of the actor's pose, which is then, for each time step, analyzed to extract the positions and rotations of each body part. That data is then packaged and sent to the dynamics simulation through a TCP package, as described in the Character Model and Network Setup section.

4.3 Dynamics Simulation

For our system, we use the Open Dynamics Engine (www.ode.org) as our real-time physics simulator. The data received from the motion capture system is converted into joint torques (See Dynamic Control) that are used to actuate the characters body. In addition, the motion capture data is also used as a basis to apply external forces that more closely follow the actors movement (see Kinematic Control) and maintain balance (see Balance Control). Finally, we employ ODEs collision handler to manage all interactions with objects in the virtual environment. The resulting physical state is then sent to the visual feedback component, either by hard-wired cable or through another TCP network connection in the form of global positions and rotations of every body (including virtual objects) in the simulation.

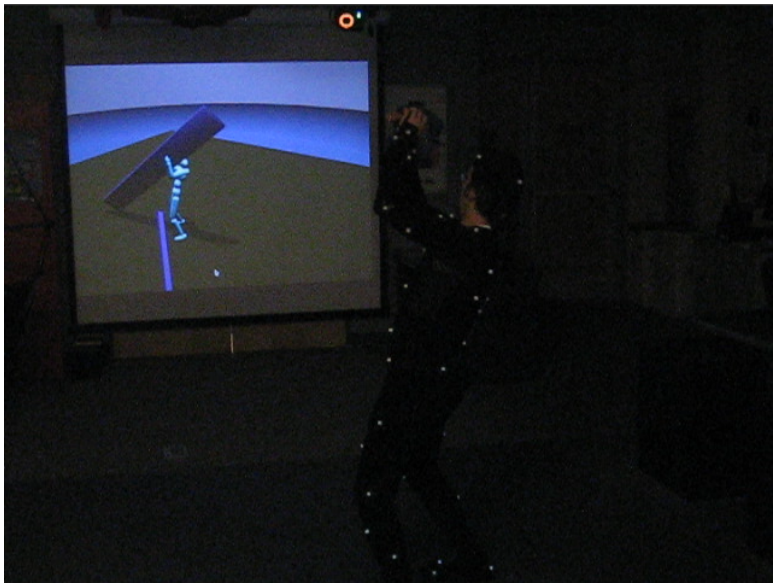

Figure 4.3: Projector display used to give the actor visual feedback

4.4 Visual Feedback

In the local version of the system, the dynamics simulation is simply connected to a projector and displayed on a projector screen (See Figure 4.3). The networked version, similar to the motion capture and dynamic simulation components, receives positional/rotational data of each body part and recreates them either on a monitor or projection screen. As with the dynamics simulation, it is assumed the remote display has the measurements of the objects and character prior to the simulation in order to display everything in the correct proportions. Although both are suitable for general audience viewing, each is used for very different purposes.

The local display has additional visual cues, not contained in the network display, concerning balance, actor movement, and motion discrepancies that, while suitable for

general audience view, are intended as an interactive aid for the actor. For balance, a color scheme is applied to the character to indicate when he or she is in balance (green) or out of balance (red). In the case the character is moving out of balance, the color of the character changes from shades of green to shades of red and vice versa. In addition to coloring, a visual display of the character's center of mass and support polygon are used to help further indicate when the character might fall down. A secondary, "ghost" character, which represents pure motion capture, is also added to the display, allowing the actor to clearly see the differences between his or her motions and how they are modified by the simulation. In the case the character falls down, the ghost character is essential to assisting the actor in realigning him or herself for resynchronization. These add ons have the power of showing how closely the character follows the motion capture, how much deviation occurs during interactions, and aids the actor in starting again when the deviation becomes too great.

The network display acts as a graphical shell for the simulation, taking all of the objects in the world for additional rendering primarily aimed towards a general audience. Our local display uses simple objects to construct the character and objects in the world. As a tradeoff for actor aiding cues, the local display renders the environment as simply as possible. To prevent rendering from bottlenecking the system, motion data is sent to an external computer to allow for a much more detailed graphics rendering of the environment. Our final renders in Figures 5.1, 5.2, 5.3, and 5.4 utilize the Open Graphics Rendering Engine (OGRE) to renderer a scene that allows for more detailed shading, textured objects, and a

cleaner looking character. The result is a higher quality render aimed towards an audience.

These two different visual feedback setups will eventually split how this system is used. The local setting focuses on giving the actor enhanced visual feedback on how the character is performing in the virtual world, while the network setting focuses on an additional layer of rendering to make the motion more closely match production quality. Although both displays can be brought up to movie production quality, the local display, with its functionality geared more towards aiding the actor, lends itself more towards a video game environment, where realtime interaction and knowing the exact state of your character are essential. The networked display, on the other hand, gives the director more control over the look of the final environment as well as camera placement independent of the actor's concerns. By separating the simulation and display, the result is setup that is both focused on the actor as well as the director.

Chapter 5

Examples and Results

We crafted as series of different scenarios to show off specific features of the system as well as its overall flexibility. The scenarios described show a progression from precision control of the small objects to little or no control when the character is knocked down by exceptionally large objects. More specifically, the first two examples primarily show off the KD continuum while the latter two additionally showcase the balance controller. The results appear in the film strips in Figures 5.1, 5.2, 5.3, and 5.4.

5.1 Ballroom

We set up a scenario to showcase the actor manipulating objects in a free-style manner. The character moves the light-weight (10kg) objects around to prevent them from entering the containment area. If the ball does fall within the boxed-in area, the character can still extract the ball, by either grabbing or kicking it out of the box. This example

showcases simple hand manipulation of objects. (See Figure 5.1).

5.2 Boardroom

To show off the capabilities of sustained contact, we present the character with a simple room with two planks, one propped up and leaning on the other. The actor is instructed to kick out the first plank and then catch and carry the larger of the two (mass is 40kg). The character interactively manipulates the block by dragging it around and eventually throws the board down and gives the board a frustrated kick. The sustained contact shown is unlike many of the interactions seen in previous work where the emphasis has been focused on fast impacts and instantaneous impulses. (See Figure 5.2).

5.3 Punching Bag

A hinged cylindrical mass is suspended from the ceiling and the character is able to interact with the object in various ways. The first example shows the effect and reactions

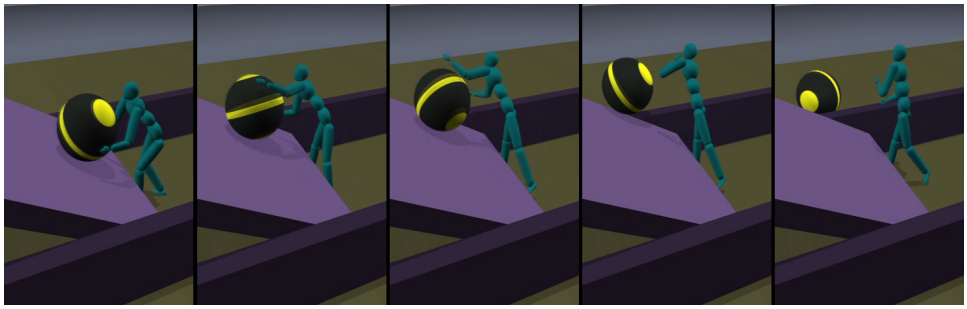

Figure 5.1: Character manipulates a ball with his hands and arms to prevent it from rolling into the containment area. He does this by rolling the ball off of the ramp

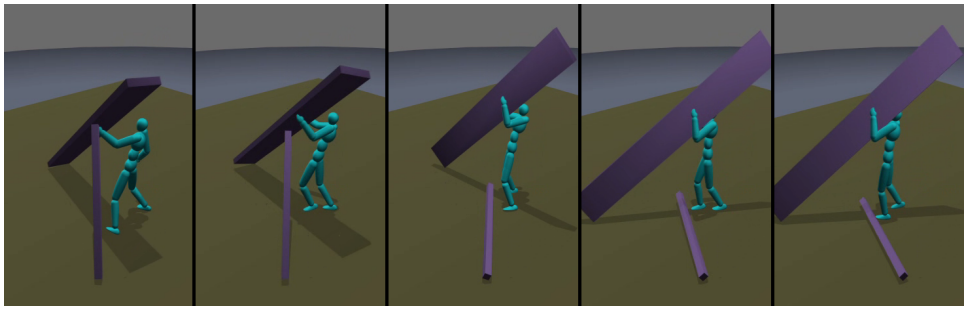

Figure 5.2: Character kicks a plank aside, and proceeds to catch and carry the larger board

of the character's body as the actor starts and stops the large mass (50kg) from swinging. The actor uses his full body and leans forward to stop the punching bag, and then leans back then forward to push it back up using his arms and upper torso. With varying speeds of the bag and varying motions of the actor, the interaction between the character and bag can be as smooth or abrupt as the actor desires.

The second example using the punching bag showcases the dynamics controller and balance controller when the character is "airborne". The character first hugs and literally hangs off of the bag, effectively turning off the balance and kinematic controllers and relying solely on the dynamics controller to remain hanging. In this case, the joint torques are the only actuators keeping the character suspended. (See Figure 5.3).

5.4 Boulder

The boulder world is the most extreme scene with a large (150kg) sphere being flung straight at the character in a confined space. Given the constraints of the world, the character/actor has few options and can either attempt to squeeze to the side and let

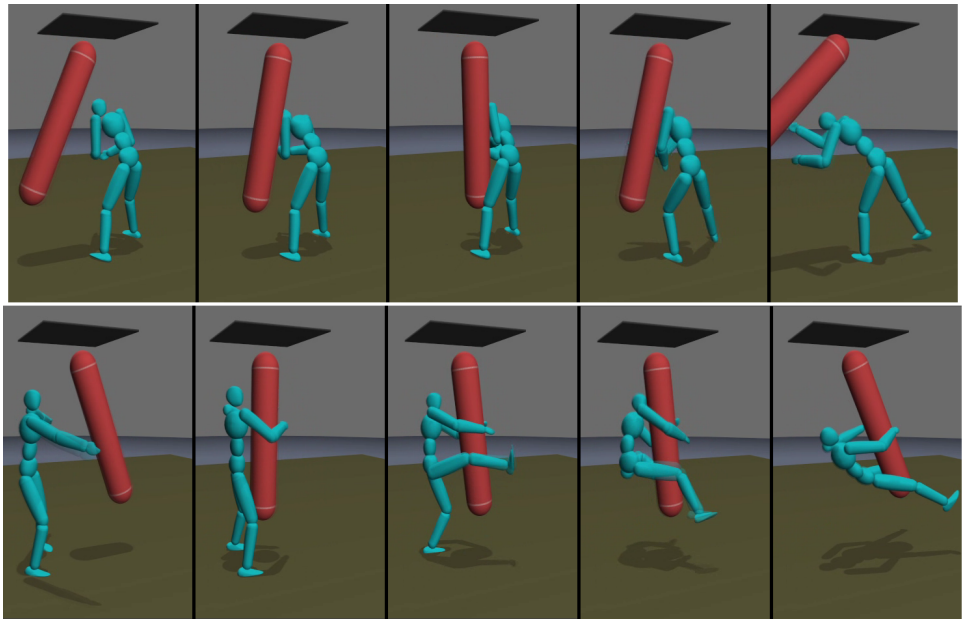

Figure 5.3: (Top Strip) Character pushes the punching bag with his full body (Bottom Strip) Character then grabs and hangs off of the punching bag

the boulder pass by or be run over by the boulder and try to recover afterwards. While the former does not knock the character over, it does nick him, forcing the character to fight losing balance. In the latter case, the character is knocked down, but by using the recovery process mentioned in the Balance section, the actor is able to resynchronize with the character and stand him back up. (See Figure 5.4).

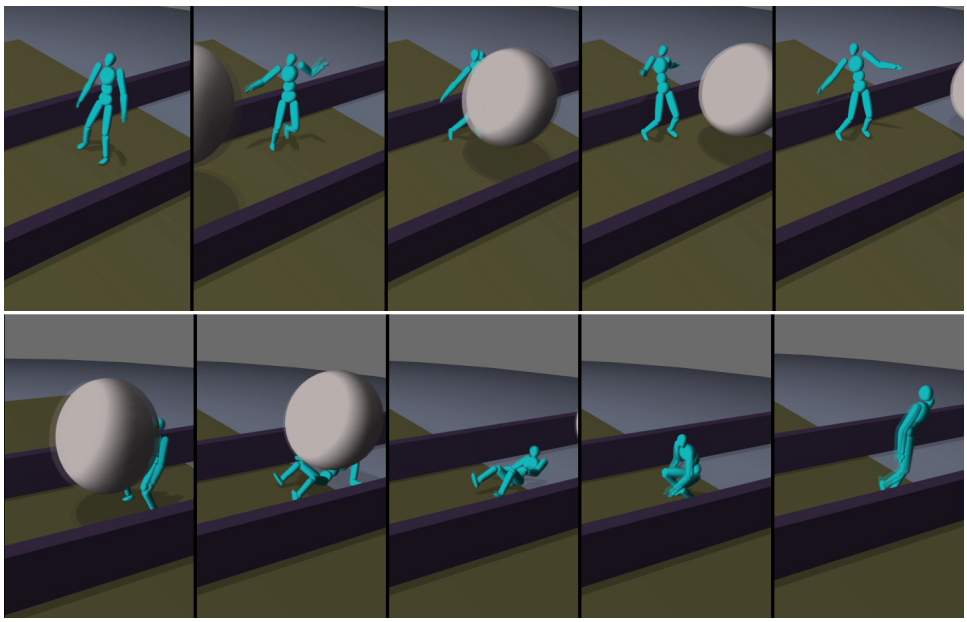

Figure 5.4: (Top Strip) Character dodges the oncoming boulder and is slightly nicked by it (Bottom Strip) After being knocked down by the next boulder, the character stands back up

Chapter 6

Limitations and Future Extensions

While the power of this real-time technique is shown by the unique interactive animations mentioned in Figures 5.1, 5.2, 5.3, 5.4, there are several limitations to the current system. We highlight a few of the key issues here and offer some suggestions for extensions to the described system.

Currently, we do not have an actuated, functioning hand so grabbing is less accurate than in real life. We do show examples of manipulation done with the hands, arms and other body parts but the character cannot carefully grab objects with finger level accuracy. This was omitted since by using the current system setup, hand tracking would become a major network bottleneck in terms of data packet size. In addition, the level of detail obtained by the motion capture system would be too unreliable to determine if the actor is opening or closing the hands carefully. This issue can be easily overcome by either using a more precise motion capture setup, or by localizing the hand control to the simulation

Figure 6.1: Incorporating a head mounted display (HMD) would allow the actor further immersion into the virtual world

and using some sort of controller to indicate grabbing or opening the hand. As with many theater and video game setups, this described implementations forces the actor look in a specific direction to see what the character is doing. This setup can be problematic in cases where the head's orientation plays a key role in the animation. This window into the virtual world can be replaced with eyes by incorporating a head-mounted display (HMD) from which the actor can view the world (see Figure 6.1). The HMD allows the actor to see the virtual environment in first person, further immersing the actor into the world and allow him or her to focus on the interactions as if he truly were part of that world.

We currently require that the performer act as an accomplice to the system because, as is, the system can lead to unmanageable situations. For cases when the character falls down, the actor has to lay down to resynchronize with the character. Additional heuris-

tics can be added to resynchronize the character to the actor instead, but would most likely require either prerecorded motions, in this case, of a person standing up, or more complicated automated controls. As is illustrated with this example, while this system can be applied to video games, additional play mechanics and resynchronization techniques would have to be incorporated for it to be viable in a commercial video game setting.

Aside from additions, there are several issues with our current system. Our kinematic and balance controllers can at times remain too powerful and lessen the effect of physical interactions. While a more elaborate balance controller, either still using the kinematic controller or one incorporated into the dynamic controller, could allow for less stiff physical interactions, there exists a fine line between automation and user control. We purposefully shied away from elaborate controls to give the user direct, intuitive control over the character. Not to say a more elaborate balance controller would not help; our system assumes the character is standing on flat ground and was not designed to deal with uneven or dynamic terrain. By simply allowing the character to navigate through variable ground would open opportunities for a variety of additional scenes and scenarios applicable to more action oriented level and environment designers.

The network setup of the system was designed with multiplayer scenarios in mind. With a single simulation component acting as a server, multiple motion capture components could stream their data to the simulation and back to their displays to create a physically simulated, massively multiplayer virtual world. This would lead to very immersive cooperative or adversarial play between different people, allowing them to communicate remotely

with physically realistic, or exaggerated, avatars. As online gaming has already shown, cooperative and adversarial online play is a very large market where thousands, even millions, communicate frequently. Add into it more immersive controls, and the possibilities are limitless.

Chapter 7

Conclusion

We describe a technique that incorporates a physical simulation with a real-time motion capture performance in order to allow an actor to interact with virtual objects in complex scenarios. While still following the actor's motions, the simulated character is able to interact with objects in the virtual world. By giving the actor direct control over the character, we eliminate the need for preprocessing or prerecording motions by allowing the actor to input his or her intelligent control. We have created a continuum between kinematics and dynamics which follows the actor's motions as closely as possible, while still remaining physically faithful to the virtual environment. This is done by smoothly switching between the two signals on a per-body-part basis, and the result allows the actor to roll, carry, push, grab, dodge, and brace many different objects in physically based, virtual world.

Bibliography

[1] Allen B., Chu D., Shapiro A., Faloutsos P.: On the beat!: timing and tension for dynamic characters. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2007), pp. 239-247.

[2] Abe Y., Da Silva M., Popovic J.: Multiobjective control with frictional contacts. In *Eurographics/SIGGRAPH Symposium on Computer Animation* (2007), pp. 249-258.

[3] Arikan O., Forsyth D. A.: Synthesizing constrained motions from examples. *ACM Trans. on Graphics (SIGGRAPH) 21*, 3 (July 2002), 483-490.

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

[5] Barbic J., Popovic J.: Real-time control of physically based simulations using gentle forces. *ACM Transactions on Graphics 27*, 5 (Dec. 2008).

[6] Chai J., Hodgins J. K.: Performance animation for low-dimensional control signals. *ACM Transactions on Graphics (SIGGRAPH 2005) 24*, 3 (Aug. 2005).

[7] Choi M. G., Lee J., Shin S. Y.: Planning biped locomotion using motion capture data and probabilistic roadmaps. In *Proceedings of ACM SIGGRAPH '03* (2003), vol. 22(2), ACM Press. pp. 182-203.

[8] Funge J., Tu X., Terzopoulos D.: Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *Proceedings of SIGGRAPH 99* (Aug. 1999), Computer Graphics Proceedings, Annual Conference Series, pp. 29-38.

[9] Ishigaki S., White T., Zordan V., Liu K. C.: Performance-based control interface for character animation. *ACM Transactions on Graphics (SIGGRAPH 2009) 28*, 3 (Aug. 2009).

[10] Kovar L., Gleicher M., Pighin F.: Motion graphs. *ACM Trans. on Graphics (SIGGRAPH) 21*, 3 (July 2002), 473-482.

[11] Lau M., Kuffner J. J.: Behavior planning for character animation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2005).

[12] Silva M. D., Abe Y., Popovic J.: Interactive simulation of stylized human locomotion. In *ACM Transactions on Graphics 27*, 3 (July 2008).

[13] Schoedl A., Essa I. A.: Controlled animation of video sprites. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2002).

[14] Shiratori T., Hodgins J. K.: Accelerometer-based user interfaces for the control of a physically simulated character. *ACM Transactions on Graphics 27*, 5 (Dec. 2008) 123:1-123:9.

[15] Shin H. J., Lee J., Gleicher M., Shin S. Y.: Computer puppetry: An importance-based approach. *ACM Trans. on Graphics 20*, 2 (Apr. 2001), 67-94.

[16] Yin K., Coros S., Beaudoin P., van de Panne M.: Continuation methods for adapting simulated skills. *ACM Transactions on Graphics 27*, 3 (July 2008).

[17] Yin K., Cline M., Pai D.: Motion perturbation based on simple neuromotor control models. In *11th Pacific Conference on Computer Graphics and Applications* (Oct. 2003), pp. 445-449.

[18] Yin K., Pai D. K.: Footsee: an interactive animation system. In *2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aug. 2003), pp. 329-338.

[19] Yin K., Pai D., van de Panne M.: Data-driven interactive balancing behaviors. *Pacific Graphics* (2005).

[20] Ye Yuting , Liu K. C.: Responsive characters with dynamic constraints in near-unactuated coordinates. *ACM Transactions on Graphics 27*, 5 (Dec. 2008).

[21] Zordan V., Hodgins J.: Motion capture-driven simulations that hit and react. In *ACM SIGGRAPH Symposium on Computer Animation (SCA 02)* (July 21-22 2002), ACM Press, pp. 89-96.

[22] Zordan V., Majkowska A., Chiu B., Fast M.: Dynamic response for motion capture animation. *ACM Trans. on Graphics (SIGGRAPH) 24*, 3 (July 2005), 697-701.