Versatile and Interactive Virtual Humans: Hybrid use of Data-Driven and Dynamics-Based Motion Synthesis

Michael J. Mandel

August 2004

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA

Thesis Commitee
Jessica K. Hodgins, Chair
Nancy S. Pollard

Submitted in partial fulfillment of the requirements for the degree of Master of Science.

Keywords: Animation, Motion Capture, Human Motion, Motion Graph, Physical Simulation, Motion Synthesis, Kinematics, Dynamics, Interactive, Autonomous Virtual Human, Entertainment, Feedback Controllers, Nearest Neighbor Search, Hybrid Architecture, Ragdoll Physics, Videogame, Virtual Training

Abstract

Highly interactive characters that behave in situationally appropriate ways are an important goal of researchers, film makers, and game developers. For example, synthesizing believable boxers would involve representing the stylistic bobbing and weaving motions while generating realistic dynamic responses to blows that are given and received, including the resulting interactions with the boxing ring. Realizing that no single motion synthesis technique is perfect for every situation, we propose a hybrid system that favors the one most suited to the current objectives. Motion graphs are an effective tool for synthesizing realistic and easily directable characters that accurately reproduce the stylistic nuances of human motion. However, because motion graphs rely on splicing data obtained before runtime, they are not adequate for applications where the external forces or detailed interactions with the environment cannot be predicted. Simulation allows the physical interactions between a character and its environment to be modeled realistically, but does not provide a wide range of behaviors because of the difficulty in constructing control systems for complex behaviors. This thesis combines the complementary strengths of these two techniques so that complex animation tasks in novel environments may be synthesized interactively. Our system attempts to reasonably resolve when either technique is most appropriate and provide the facilities to transition between them as the character's goals and interaction with the environment evolve. To ensure these transitions are smooth, the Approximate Nearest Neighbors search algorithm is used to locate a good correspondence between the simulation and the motion database. Joints are actuated by physical controllers to guide the simulation near existing motion found using this correspondence search. These proportional derivative controllers are also used to add human-like subtlety to the simulated motion in a biomechanically inspired way for behaviors such as protective falling. We demonstrate and evaluate the power of this approach by switching between simulated and data-driven tasks in a context dependent way, triggered by physical interactions with the virtual human. As simulation techniques improve, such an architecture can support the future goal of fully autonomous simulated characters, while still being able to fall back on motion data for hard to simulate behaviors.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An ambitious goal shared by many researchers is that of creating autonomous virtual characters capable of generating motion for any behavior, in any context, with the subtlety and style of real human motion. Film production, videogames, and virtual training environments could all benefit from virtual characters that can adaptively generate human-like motion for any situation, even when detailed interaction with the environment and other characters is necessary. Approaches to the problem of synthesizing human motion for this purpose can be roughly divided into two categories:

1. Collect and organize a large amount of motion data covering all the necessary behaviors and their natural transitions. Animator-specified keyframes and motion capture data are among the sources of motion data for this data-driven approach.

2. Generate the motion procedurally using some high-level behavioral logic coupled with a low-level motor control technique. Inverse Kinematics and physical simulation are example techniques that could provide the low-level motor control in this category.

Despite the power of these approaches, researchers have yet to create autonomous characters without limitations with respect to their abilities and/or natural human-like qualities.

As a motivating example, videogames today often use animator keyframes and/or motion capture data to drive most of the character's behaviors because they accurately capture the stylistic nuances of human motion. Unfortunately, the characters sometimes look unrealistic when interacting with dynamic environments because their movements are predetermined by this static motion data. For example, a boxer's reaction to an opponent's punch should account for the strength, location, and direction of the blow in addition to how the body will collide with the surrounding boxing ring. This a difficult problem to model with motion data because these factors create an almost infinite set of possible responses that cannot be easily predicted until runtime. More recently, "ragdoll" simulation has been introduced to physically model the detailed interactions between the character and the environment. Unfortunately, the motion has a lifeless, passive quality because it lacks control systems to generate active, natural behavior. There is also no obvious way to seamlessly return character control to motion data after the simulation has started because the pose of the character can change in unpredictable ways. The relative strengths of motion data and simulation motivate the need to reasonably understand when to switch character control between the either technique in a context sensitive manner.

No single motion synthesis technique previously developed has proven perfect for every situation. We therefore propose a hybrid system capable of utilizing more than one motion synthesis technique, favoring the one most suited to the current objectives. Such a hybrid system would allow the goals of the character and the limitations of the available synthesis techniques to guide the selection of which technique is most appropriate. Figure 1.1 shows an example of the high-level flow of character control for a videogame utilizing this methodology.

1.1 Problem Statement

We focus our exploration of a hybrid motion synthesis system on utilizing physical simulation and data-driven motion synthesis in a context dependant way. We explore how and

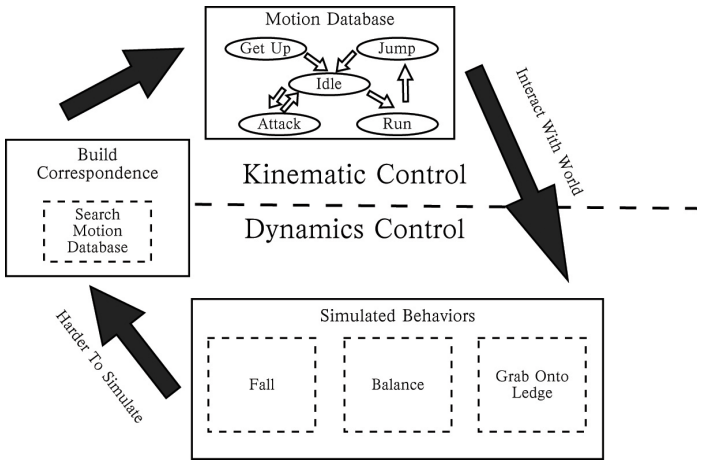


Figure 1.1: A videogame using our hybrid methodology might make transitions between data-driven and dynamics-based control depending on how the character interacts with the environment and the system's ability to simulate the requested behavior. Moving back to motion data control requires a correspondence step that registers the current simulated pose with closely matching ones found in the motion database.

when to allow character control to switch between these two complementary techniques to enable the most extensive repertoire of behaviors possible. In addition, we wish to ensure that our simulated behaviors capture as much human subtlety and situational appropriateness as possible. Our simulated behaviors will focus on reactive responses to external forces that create a dramatic loss of balance in the character.

The main technical challenges of this project are

- *Quickly finding closely matching frames in a motion database to poses generated by a simulation:* The simulation is likely to change the pose of the character considerably and this change cannot be predicted beforehand because it evolves at runtime as the character interacts with the environment. If we are to allow character control to return to motion data, a fast search technique is necessary to build a correspondence with an existing motion database.

- *Ensuring that the simulation settles in a configuration that is close to existing motion*

data: The simulation must guide the pose of the character into configurations that are likely to allow motion data to take over character control, while still retaining its behavioral appropriateness.

- *Developing controllers that allow the simulation to generate natural human behavior:* Control systems that demonstrate the coordination and subtlety of human motion are difficult to develop, but are critical to the believability of autonomous characters. We are interested in developing behaviors that are manageable to create control systems for, but are particularly difficult to represent using motion data. New behaviors should be easy to add as simulation techniques are developed, requiring less reliance on pre-recorded motion data.

This thesis aims to develop solutions to these challenging problems using a variety of techniques that build on work from the graphics, biomechanics, robotics, and artificial intelligence literature.

1.2 Techniques

Our hybrid system first requires the availability of motion synthesis techniques to reproduce human motion. We utilize motion graphs, which have recently become a popular data-driven technique because they allow synthesis of easily directable motion [1, 27, 32]. By automatically processing a large unlabeled database of motion capture into an easy-to-use graph structure, motion graphs can splice together bits and pieces of data to synthesize motion in real-time that is subject to a variety of user-defined constraints. Because motion graphs automatically determine the ways in which motion can be rearranged, they are especially useful for generating natural transitions between behaviors. Unfortunately, motion graphs suffer from the same problems that most data-driven techniques do when the characters are embedded in a dynamic environment. These problems include physically unrealistic responses to external forces, body parts clipping through environment geometry

Figure 1.2: This sequence shows how our fall controller uses biomechanically inspired rules to generate protective behavior used to avoid injury.

due to inadequate collision handling, and motion blending artifacts such as foot sliding.

The second part of our hybrid system, physical simulation, provides a way to model the interaction between a character and its environment in ways that cannot be anticipated at the time motion data is captured or keyframed. We model interactions between the body and its environment, subject to various internal and external forces, using a rigid body dynamics engine. The difficulty with using simulation for human behaviors is developing robust control systems that reactively generate realistic human behaviors. We use proportional-derivative controllers that approximate internal muscle actuation to drive the body towards target postures determined by a high-level state machine. The hybrid system allows us to focus on behaviors that are manageable to simulate, allowing the data-driven approach to handle hard to simulate behaviors, such as a balanced walking gait. In particular, we develop a biomechanically inspired fall controller that can generate protective behavior when arbitrary external forces cause the character to lose balance. Figure 1.2 shows an example simulated motion generated by the fall controller. The fall controller serves as a first step towards a reactive and realistic human when our hybrid system is using simulation for character control.

The core feature of the hybrid system is switching between these two synthesis methods. Moving to simulation from our data-driven approach is relatively straight forward, but because the simulation can change the posture of the character in unpredictable ways, the return to data-driven control is challenging. Thus, we focus on building a correspondence with the motion database so that motion near the current simulated pose can be selected

when character control must transfer from simulation to data-driven synthesis. We look at how a nearest neighbor search algorithm can exploit relationships in the motion data to speed up this correspondence process. Because an autonomous character is expected to have an extensive repertoire of behaviors, the motion database can be quite large, requiring a search method with performance that scales well with the size of the database. We utilize the Approximate Nearest Neighbor algorithm to maintain real-time performance, even for large databases. The result of this fast search is the ability to give the simulation a hint as to the body pose that it should aim for so that character control can smoothly transfer to the motion data matched by the search. Special physical controllers are used to push the simulation toward the result of the correspondence search as a more physically grounded alternative to simple blending. For example, our falling controller leaves the character in a prone position. This correspondence process can be used to find and settle the body pose near clips of motion that generate the hard to simulate process of returning to a balanced, upright posture.

We assess our system's results by examining the quality of transitions made between motion graph controlled tasks and simulated responses triggered by an extensive, but controlled set of external forces. Even when the character experiences a loss of balance, our system can recover from the simulated response back to tasks synthesized by our data-driven technique. The simulated response reasonably approximates what a real human might do in a variety of situations.

1.3 Contributions

The main contributions of this work are summarized as follows:

- A hybrid system capable of moving between motion synthesis techniques; motion graphs and controlled physical simulation in particular. The system is able to select a technique that complements the required task, allowing it to support an autonomous

virtual character.

- A biomechanically inspired physical controller that is able to generate responses to a loss of balance in a virtual character.
- The use of controllers to drive a simulated character into a pose that is near existing motion data so character control can be transferred to a data-driven approach.
- The application of a fast search technique to the problem of efficiently searching a large motion database for motions near a query posture.

1.4 Thesis Overview

Following a discussion of related work in Chapter 2, we cover two techniques for synthesizing human motion: motion graphs and physical simulation. In Chapter 4, we discuss how to build a correspondence between simulation and motion data as well as how to carry out the transition from each technique to the other. Chapter 5 describes how we generate human behaviors, like protective responses to falling, using physical simulation while Chapter 6 presents the results generated by our hybrid system. We conclude the thesis in Chapter 7 with a discussion of future areas of research.

Chapter 2

Background

Reproducing human motion in a realistic and controllable way has long been a goal of researchers, film makers, and game developers. The hybrid system presented in this thesis was inspired by researcher's work on the problem of animating humans using kinematic and/or dynamics-based approaches. Kinematics refers to approaches that directly specify the pose of the human over time, such as artist keyframes and motion capture data. Using large quantities of pre-existing motion data for synthesis, as we do, is known as data-driven motion synthesis. When animating characters using simulation, researchers in biomechanics, robotics, and computer animation delegate varying degrees of importance to believability, style, accuracy, and functionality. While biomechanists are most concerned with accuracy and roboticists with functionality, computer animation, like our work, is more concerned with believability and style.

This chapter focuses on the related works of other researchers, specifically on synthesis of human motion using data-driven and simulation-based approaches. We first look at data-driven approaches and how they evolved from simple motion editing into automatic techniques that synthesize motion using pre-processed databases of motion. We finally look at how physical simulation has been used to generate human motion, emphasizing controller and trajectory optimization based approaches.

2.1 Data-driven Synthesis of Human Motion

Our kinematic motion synthesis technique generates motions subject to user-directed constraints using a large motion database. While user-directed constraints can be solved analytically using procedural approaches like inverse kinematics [14, 19, 23, 72] and constraints such as where the hands or feet should be, we focus on a data-driven approach because it reproduces human motion more accurately. Many researchers have influenced this kind of work through techniques that synthesize new motion by processing and editing existing motion. Even though motion capture technology can reproduce large amounts of human motion fairly easily, it is difficult to adapt existing motion to new situations, making it reusable. The simplest motion editing techniques involve simple averaging of animation curves, called motion blending. Early work by Perlin [45] used simple blending to generate transitions between motions and semi-random noise functions to augment the motion with perceived personality. Displacement mapping techniques were introduced by Witkin and Popović [70] to warp the animation curves of existing motions to pass through keyframe constraints using a blending scheme. For example, a normal walking motion could be modified with this technique to bend through a doorway. Bruderlin and Williams [5] introduced similar curve editing techniques, as well as the concept of timewarping motions to align them temporally before blending. Lee and Shin [33] later used hierarchical displacement maps and inverse kinematics to warp motions to specified constraints and adapt motions to characters with different proportions. We use simple blending techniques like these to perform transitions between motion segments.

Other blending schemes were later developed that explored more complicated blending weights to produce better transitions and allow the creation of parameterized motions. Unuma *et al.* [66] looked to the Fourier domain to produce blend weights while Rose *et al.* [53] and Gleicher [15] used spacetime optimization to generate transitions between motions that minimized joint torque. For a more detailed survey of some of these earlier constraint-based, motion editing techniques, see Gleicher [17].

Using a number of aligned example motions and carefully varying the blend weights, an intuitive space of parameterized motions can be created by interpolating between the example motions. Rose *et al.* [52] used manually aligned example motion clips and radial basis function interpolation to synthesize motions parameterized on emotions like happiness and anger as well as on locomotion qualities such as walking direction and speed. Park, Shin, and Shin [44] developed this idea further by using a novel blending scheme to generate parameterized motions from examples that are retargeted to new characters at runtime. Interpolating example motions can also generate motion satisfying inverse kinematics constraints such as hand position and kick height [54, 68]. Kovar *et al.* [25] demonstrated automated methods of aligning example motions and generating parameterized blends of example motions using registration curves, later extending this [26] to identify logically related motions that do not necessarily contain similar poses. Parameterized motions using related techniques are viable additions and/or alternatives to the synthesis techniques we chose in our hybrid system. Staying truer to the original data, we chose a data-driven synthesis technique that relies on splicing unaltered clips from a large motion database rather than interpolating between a small number of example motions.

With the availability of larger motion databases, researchers have focused on methods to organize the data to make the motion synthesis process more robust. Pullen and Bregler [48] keyframed a small number of degrees of freedom and filled in the rest by matching with lower frequency bands from a motion database, generating fully realized motions. Some researchers have attempted to synthesize motion by capturing a state-based, statistical model of large motion databases [4, 35, 64]. Because these techniques build abstractions of the original data, they can sometimes lose important details present in the original data. We therefore look at methods that give guarantees on motion quality by relying on rearranging the original data. These approaches attempt to organize the motion database into a graph structure that indicates relationships between pieces of data close enough to splice together. Traditionally, game developers have generated this graph structure by a tedious process of manually specifying potential transitions in the motion database [39]. Automatic methods

for generating these transition graphs for animation were motivated by Schödl *et al.* [57] with their work in generating seamless, infinitely looping video textures. Small windows of frames were matched using a similarity metric to identify potential cuts that could be made in an input video sequence without loss of visual continuity. These same ideas were later applied to the development of automatic techniques that discover potential transitions in a motion database, arranging the data into a motion graph for future synthesis [1, 27, 32]. We use motion graphs as our data-driven synthesis technique and describe them in detail in Chapter 3.

While these blending and curve manipulation techniques are effective in many situations, they have several limitations. Data-driven techniques have limited flexibility in situations that require complex physical interactions with the environment. Motions such as dangerous stunts are difficult to capture and adapt to new environments without enforcing physical laws between the environment and the character. We therefore look at how physically based techniques for human motion were developed for these situations where kinematic approaches are not sufficient.

2.2 Physical Simulation of Human Motion

Using physical simulation to drive the motion of a virtual human can give a level of physical connectedness to the environment that data-driven techniques cannot match unless the motion was recorded precisely for a particular situation. Techniques that use physical simulation to generate human motion can be roughly divided into trajectory-based approaches and controller-based approaches. We prefer controller-based approaches in our work as they are more suited for interactive characters because of their computational efficiency. However, we cover both here since either approach would be valid in a hybrid architecture.

2.2.1 Trajectory-Based Approaches

Trajectory-based simulation techniques attempt to optimize for a physically realistic trajectory from one point in the human figure's state space to another. The user inputs a rough sketch of the requested motion, such as starting and ending keyframes that are then used as constraints in a search for a physically valid motion. Witkin and Kass [69] first introduced such an approach to the graphics community with a simple hopping Luxo lamp. Because of the highly underconstrained nature of many of these systems, the trajectories are often optimized for properties such as minimum energy consumption, smoothness, and/or minimum time. Humans have a large number of degrees of freedom, making them computationally expensive to optimize for, thus many researchers have worked with simplified optimization or human body models. Fang and Pollard [12] found that more efficient optimization performance could be obtained by reformulating the optimization in a way that restricts the use of joint torques in the objective function or constraints. Popović and Witkin [47] showed how the optimization could be performed on a human model containing only the important degrees of freedom and then transformed back onto a more complicated human model. Safonova *et al.* [55] explored how using principal component analysis to reduce the dimensionality of the human body could both improve optimization performance and lead to more natural looking solutions.

2.2.2 Controller-Based Approaches

A number of researchers have used physical controllers to generate motion for a variety of human behaviors. Laszlo [30] created fully 3D controllers for periodic motions using an open loop control system that produced parameterized and balanced motions for behaviors such as walking. Hand-tuned feedback controllers have been demonstrated for simulating such athletic behaviors as running, vaulting, and bicycling [22]. Hodgins and Pollard [21] explored how scaling algorithms could be developed to adapt such simulated behaviors to new characters, making controllers more reusable. Developing composable controllers also

contributed to reusability as it allowed more complex behaviors to be derived from simpler controllers applied in sequence. Wooten [71] first explored this idea by developing controllers that generate the motion of human divers by sequencing individual controllers such that the valid starting states of a controller matched the valid ending states of a previous controller. Faloutsos *et al.* [11] further developed the use of composable controllers by stringing together many behaviors like falling, standing, and balancing to create a virtual stuntman. Developing these hand-tuned controllers is difficult, however techniques have been developed to search for control rules using simulated annealing or genetic algorithms [62]. Interactively controlling a physically simulated character is an important application of physical controllers for virtual characters. Laszlo *et al.* explored using intuitive interfaces to allow users to take direct control of a simulated human to produce running, climbing, and gymnastics movements [31].

As in many of the referenced works, we implement proportional-derivative controllers as the low-level mechanism that drives the movement of our simulated behaviors. The low-level details of how we implement physical simulation and compute controller torques is covered in Chapter 3 while the details of our high-level simulated behaviors are discussed in Chapter 5.

2.2.3 Hybrid Approaches

The ability to interface physically controlled behaviors with existing motion data, as our hybrid system does, is important because it helps balance the strengths and weaknesses of each technique. Playter used motion capture data to drive a simulated running behavior during the underconstrained flight stage [46]. Kokkevis *et al.* [24] developed a system that takes user-supplied keyframe data as input to a tracking controller that generates a physically grounded representation of the input motion. Oshita and Makinouchi [43] also use a tracking controller on input motion data, but model the concepts of comfort and balance to dynamically react to external forces while maintaining balance. Similar work by

Zordan [73] allows characters to physically respond to external forces and smoothly return to the tracked motion data so that realistic reactions to motion capture driven behaviors such as boxing punches or table tennis swings can be made. This work differs from ours in that it uses simulation exclusively (though it is driven by motion data), rather than allowing hybrid control, and the responses are mostly small corrective ones compared to the responses to dramatic loss of balance that our characters perform.

Shin *et al.* [60] took a more motion heavy approach of cleaning up blended motion edits by dividing the edited motion into ground and flight stages and enforcing physical constraints with hierarchical displacement maps. Most similar to our work is the hybrid system developed by Shapiro *et al.* [58] that allows simulation and animation data to vote for control of the character. This work only activates data-driven character control when the simulation happens to fall sufficiently close to existing motion data. We use an efficient search process to allow the simulated behavior to drive itself closer to motion data in a physically grounded way, providing more flexible opportunities for character control to switch between simulation and motion data.

2.2.4 Simulating Equations of Motion

Many software libraries are freely available to perform a rigid body simulation of a human. Some are more focused on the accuracy of the simulation, while others sacrifice accuracy for speed. For instance, libraries may enforce constraints in a less strict manner and contain mechanisms to estimate when a particular part of the system is close enough to rest to stop simulating in order to obtain better runtime performance. We are interested in simulating motion at interactive rates, so we are willing to make this trade for improved runtime performance; one that biomechanists, for instance, would be less willing to make. Commercial dynamics libraries such Havok [50] and NovodeX [40] have already been deployed into popular commercial games such as Half Life 2, Max Payne 2, and the future Unreal 3 engine technology. Meqon [20] has recently released a product offering commer-

cial dynamics middleware for games with modules that support adding physical controllers to characters. Many researchers have previously used SD/Fast [59] because it works with a reduced coordinate description of the human to produce very efficient simulation code. However, it lacks a convenient API, making it more difficult to integrate into a hybrid system because it would require dynamically linking with an external executable in order to continually activate the simulation at run-time.

There are also several freely available libraries with comparably robust features. NovodeX, mentioned previously and particularly known for its speed, offers a free educational version of its library. Tokamak [29] is another library freely available, excelling in efficient joint constraints and handling of stacked bodies, however its API is fairly limited in its support for developing physical controllers. We selected The Open Dynamics Engine (ODE) [63], primarily developed by Russell Smith, because of its:

- Active development community
- Extendability provided by its open source license
- Broadly customizable API
- Integrated collision detection
- Real-time performance using its efficient iterative solver

We use ODE to simulate the equations of motion for a human body constrained to move into natural configurations as well as to approximate muscle actuation at the joints using torque-based controllers. The details of our dynamics-based approach for generating human motion using ODE are discussed in the next chapter.

Chapter 3

Using Data-Driven and Dynamics-Based Approaches For Motion Synthesis

Interactive systems that synthesize human motion often suffer from a conflict between the need to reproduce subtle, stylistic nuances along with complex limb coordination and the need to allow high-level directability. Realistic results require consideration of how the character will adapt its behavior in response to the dynamic environment around it. Data-driven approaches and physical simulation are two popular methods for synthesizing human motion with complementary qualities achieving many of these requirements. This chapter focuses on how each technique can be individually used to generate human motion. Motion graphs are first introduced as a method to synthesize novel motions by reassembling small clips from a large motion database. We then explain how to represent a simulated human and control its parameters using feedback control systems. These techniques will be the fundamental building blocks used to create the hybrid system described later in Chapters 4 and 5.

3.1 Motion Graphs For Data-Driven Motion Synthesis

Motion capture is a reliable way to *reproduce* realistic human motion. The data alone merely allows playback of the subject's performance from the capture session in an environment that is fixed in size and layout. Many editing techniques exist to modify data to meet user constraints, however their effects are only reliable for small changes that do not fundamentally alter the original motion. In applications where interactive control of a character is necessary in complex environments, simple editing is often not enough to adapt existing motion to new situations. In such applications, we would like to reuse existing motion data in to provide full control over the character's navigation and behavioral style within new environments.

Motion graphs were introduced [1, 27, 32] to provide a solution to this need for control by automatically discovering the ways in which the original data could be reassembled to produce natural looking motion. Instead of being restricted to a linear playback of the motion clips, the algorithm automatically produces choice points where streams of motion can be smoothly spliced to create novel motion sequences. Individual animation frames act as nodes in a graph structure, with edges between two nodes indicating that a smooth transition is possible between the two pieces of motion centered at these nodes. Because the algorithm automatically detects potential transitions, the user does not need to capture motions specifically designed to connect with each other. In the following sections, we first discuss our kinematic representation for both the human skeletal structure and the motion data followed by a description of how our motion graphs are constructed. We conclude by detailing how to synthesize novel sequences of motion subject to user-defined constraints through searching the pre-computed motion graph to resequence the data.

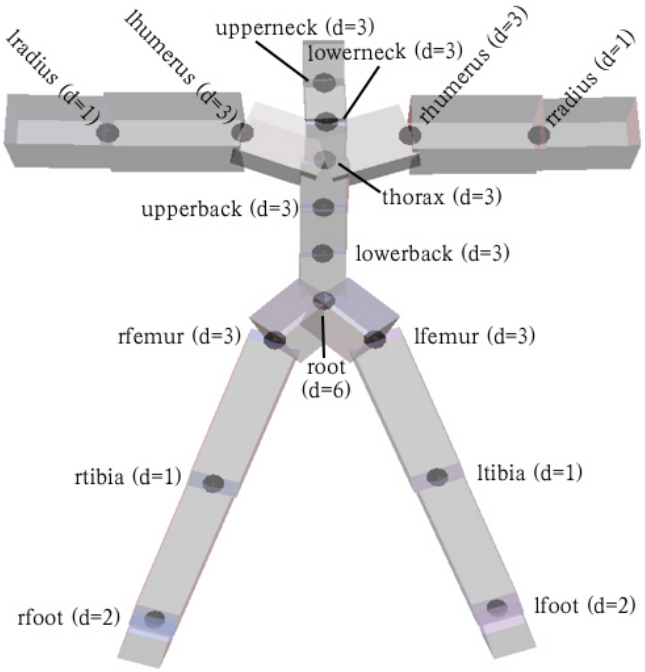


Figure 3.1: Sixteen joints representing the skeleton structure used by our virtual humans. The number of degrees of freedom we represent for each joint is also indicated.

3.1.1 Data Representation

Skeletal Structure

Our model of the human body is represented using a hierarchical set of rigid links connected by joints defining each “bone” of an underlying skeletal structure. We model sixteen joints on the body, each containing between one and three degrees of rotational freedom. The root joint contains six degrees of freedom because it includes the global body position. This hierarchical approach is preferred over others because by using relative joint angles between bones, we implicitly constrain each body part to stay connected. In addition, a skinned mesh can be animated without need to specify positions for individual vertices by deforming its geometry to the underlying skeletal structure. Figure 3.1 shows the structure of the joints used in our kinematic human model along with the number of degrees of freedom we represent for each joint.

Motion Representation

Our motions are defined as a continuous function, $M(f)$, where at each frame f is a parameter vector specifying the full posture and global position/orientation of a virtual human. Each parameter vector $P_f = (p_{root}, q_{root}, ..., q_{head})$ consists of the global position of the root joint p_{root} along with the relative orientation of each joint q_i in its parent's coordinate system. Orientations are specified as unit quaternions. In reality, a discretized version of $M(f)$ is stored at a uniform sampling rate. The parameter vector for non-integer frames is generated by interpolating between the surrounding samples to approximate the continuous form of $M(f)$. Interpolated samples are generated using linear interpolation on the global position and spherical linear interpolation [61] on the quaternion rotations.

3.1.2 Data Collection / Processing

The human motion data was captured with a Vicon optical motion capture system. The system has twelve cameras, each of which is capable of recording at 120Hz with images of 1000×1000 resolution. We use a marker set with 41 14mm markers that is an adaptation of a standard biomechanical marker set with additional markers to facilitate distinguishing the left side of the body from the right side in an automatic fashion. The motions are captured in a working volume for the subject of approximately 8'×24'. A subject is shown in the motion capture laboratory in Figure 3.2. For the skeletal structure as described in section 3.1.1, the subject's limb lengths and joint range of motion are computed automatically during a calibration phase. We resample the data to 30Hz and create a memory-mapped representation to decrease memory requirements and improve file loading performance for manipulating large databases of motion. We manually annotate the data to label individual domain-specific behaviors, such as walking, jumping, sneaking and crawling. Manual labeling was tractable because single behaviors were usually captured in long clips and many of our examples use variations of the same behavior. To create directable characters utilizing a substantial set of behaviors and their natural transitions, a more automatic tech-

Figure 3.2: A subject wearing a retro-reflective marker set in the CMU Motion Capture Laboratory.

nique would be required. Semi-automated techniques have been proposed [2] to assist in annotation by requiring a relatively small portion of the database to be manually labeled.

3.1.3 Constructing a Motion Graph

A motion graph is defined as a directed graph containing nodes representing individual frames of motion and directed edges between nodes indicating that one frame of motion may be a successor of another. Before our automatic algorithm is used to generate edges in the graph, a trivial graph can be constructed with edges between each frame of the original data, as shown at the top of Figure 3.3. To generate new motion, we need to discover ways to increase the connectivity of this graph so that two pieces of motion data may be spliced together by performing a transition at runtime. Because these transition points can represent discontinuities in the data stream, it is important to select those that would give the highest quality resulting motion. A distance metric, discussed in the next section, is applied to each pair of frames in the motion database to determine whether the posture of the character is close enough to allow a visually smooth transition. An edge is added between two frames when they score low enough using the proposed distance metric, as

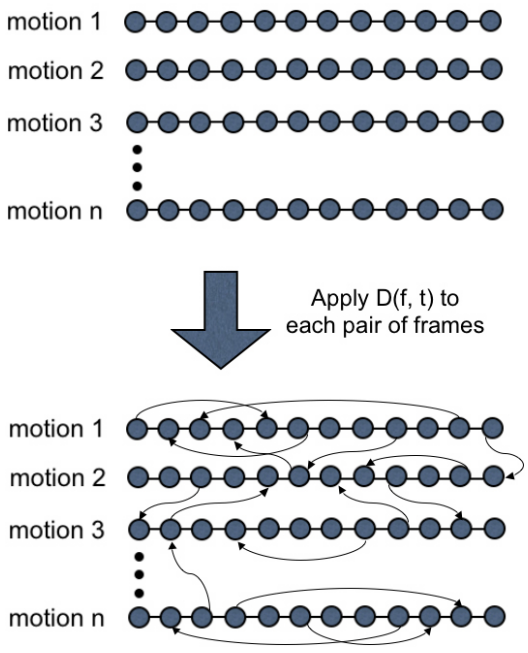


Figure 3.3: The top of the figure shows a trivial motion graph containing only the original clips of motion data. The bottom of the figure shows a possible graph after each pair of frames is then checked for similarity to find transition points, using the distance metric described in Section 3.1.3.

shown in the bottom of Figure 3.3.

Distance Metric

The key to building a motion graph is defining an appropriate distance metric between pairs of frames in the database. The metric must ensure that the position and velocity of each body part be sufficiently similar for the resulting motion to appear natural when the two pieces of motion are joined. For instance, we would not want to allow a transition between a forwards and backwards walking motion in a similar pose because of a mismatch in their velocity characteristics. Because the data is captured in the global coordinate system of the capture area, similar poses in different positions and orientations in the global space

cannot be directly compared. When the ground is flat and uniform and only the character's feet are in contact with the ground, a motion is fundamentally unchanged if the root is translated along the ground and rotated about the vertical axis. All of our data is captured under these conditions, so it is reasonable to apply a transformation to each frame to align them for comparison. We define this coordinate transformation, $T(f,t)$, aligning the poses at frames f and t in the motion database to create matching translation in the ground plane and vertical axis rotation. This computation is achieved by ignoring the global translation and finding the angle about the vertical axis that aligns the facing direction of frame *t* to that of frame *f*.

Our distance metric is modeled most closely after the one introduced by Kovar *et al.* [27]. The distance metric, $D(f,t)$, is computed between frame f and t in the database using the joint positions of the poses in a small transition time window starting at f and t. Comparison of joint positions are preferred over relative joint angles to preserve the visual structure of the body in the analysis. The purpose of computing the metric over a window of frames is to ensure that velocity mismatches between the two frames are penalized in the calculated metric. The size of the window is chosen as the amount of time of a typical transition (a 0.25 seconds worth of frames in most of our examples). $T(f,t)$, as previously defined, is computed to align f and t in the first frame of each window. This vertical axis rotation is converted to a quaternion and concatenated with the global rotation of each frame in the window starting at frame *t*. The metric is then computed as a weighted sum of squared differences of the position of each joint in the aligned frames. The equation used to compute the metric is:

$$D(f,t) = \sum_{i=0}^{\text{WS}} \sum_{j=0}^{J} w_j \left\| p(f+i,j) - (T(f,t)p(t+i,j)) \right\|^2$$

where WS is the size of the transition window, J is the number of joints in the character, w_j is the importance weight of joint j, $p(f,j)$ is the global position of joint j at frame f in x,*y*,z coordinates, and $T(f,t)$ is the coordinate transformation that aligns frame t onto frame f as previously described. Figure 3.4 shows example clips of motion, a visualization of two

Joint	Weight	Joint	Weight
root	0	thorax	.1
lfemur	.4	lowerneck	.1
ltibia	1.0	upperneck	.8
lfoot	.1	lhumerus	.4
lowerback	.8	lradius	.2
upperback	.1		

Table 3.1: Weights used by our distance metric to give more importance to visually substantial joints. The weights are symmetric left and right.

windows of joint positions, and the result of applying the coordinate frame alignment to the two windows by $T(f, t)$.

We would like the distance metric to extract perceptually acceptable transitions, however the optimal importance weights used for w_j are not known. Measuring perceptually acceptable motion is difficult, however, Reitsma and Pollard have explored perceptual metrics for measuring user sensitivity to errors in ballistic motions [49]. In general, there is no known measure of the quality of edited motion. However, Bodenheimer *et al.* suggested optimizing the weights based on selections made via human subjects and verifying the results through cross validation and a user study [67]. While they found a set of optimized weights that worked well for their example motions, we have not found them to work universally well for all behaviors. They weighted only four of the joints, the hips and shoulders, which for some of our behaviors, missed important qualities of the motion. We selected a set of weights that favor visually substantial joints and found them to work well in practice over a variety of common behaviors. Table 3.1.3 contains our empirically tuned weights.

Our distance metric is applied to each pair of frames to determine whether an edge representing a valid transition should be added to the graph between the pair. An edge

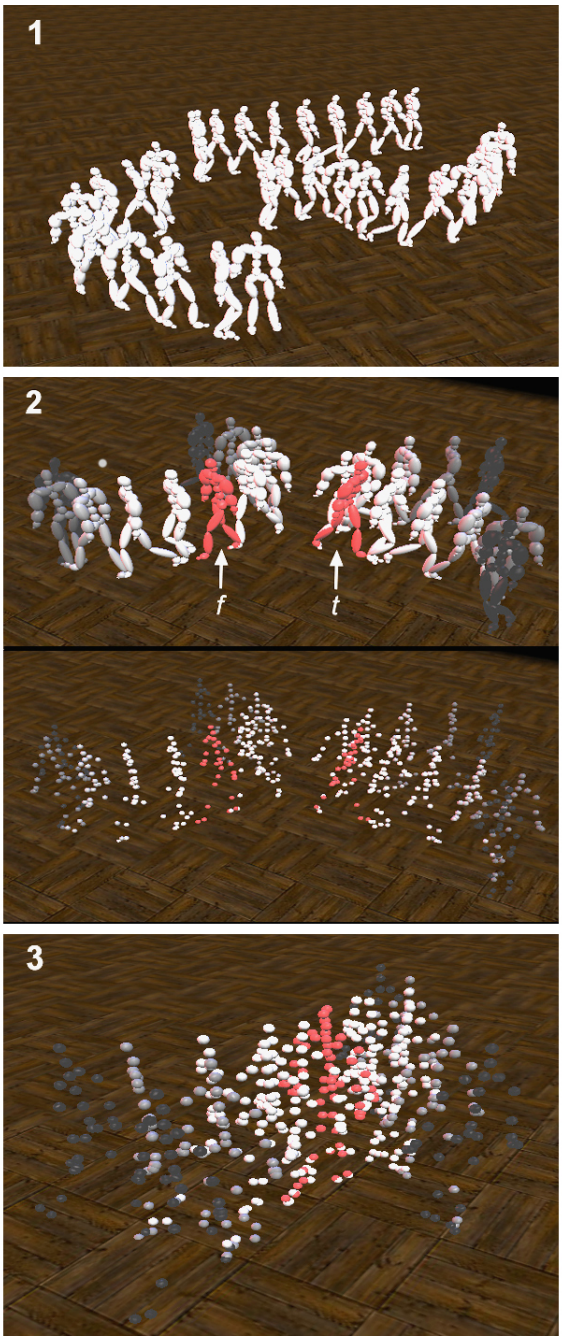


Figure 3.4: Panel 1 shows three example motion clips in the database. Panel 2 shows the two candidate windows of frames starting at frames f and t to be compared for similarity using by our distance function, $D(f, t)$. The joint marker positions which are compared are rendered in the bottom of the panel. Panel 3 shows the same two windows of frames after the coordinate frames have been aligned. The metric takes a weighted sum of the differences of these aligned marker positions.

connecting two nodes (frames) in the motion graph is added whenever the distance between the two frames is below a specified threshold. This threshold may be adjusted to balance the transition smoothness with the size and versatility of the graph. Typically, it is empirically set so that transitions exhibit only small visual discontinuities. Edges tend to be similar for nearby frames in the original motions, often causing a number of sequential frames to become edges for a particular node. Thus, we keep only the edges representing the local minima of $D(f, t)$.

Post Processing

Once the distance metric has been computed between every frame, we employ a pruning strategy adapted from [27, 32] to remove troublesome edges from the graph; this operation prunes edges that might cause the character to get stuck in a small subset of the motion database. *Sinks* are nodes whose outgoing edges restrict reachability to a small subset of the total nodes. *Dead-ends* are nodes that are not part of a cycle and would prevent the generation of further motion. We remove *sinks* and *dead-ends* in the graph by keeping only the largest strongly connected component (SCC) of the graph. This operation can be performed efficiently [65]. Figure 3.5 contains a simple example of how keeping the largest SCC affects the motion graph. An example motion graph for an idle behavior containing 737 frames of original data was pruned to 599 nodes with 237 transitions.

3.1.4 Synthesizing Motion at Runtime

Once the graph structure has been pre-computed for a motion database, we generate novel motion sequences at runtime. By searching the graph structure to produce a sequence of nodes, motion can be synthesized to satisfy user-defined constraints. Because edges in the graph represent points where two pieces of motion data may be joined, the simplest synthesis approach just walks through the graph randomly and renders the frame associated with each node encountered while maintaining the proper transformation for the character

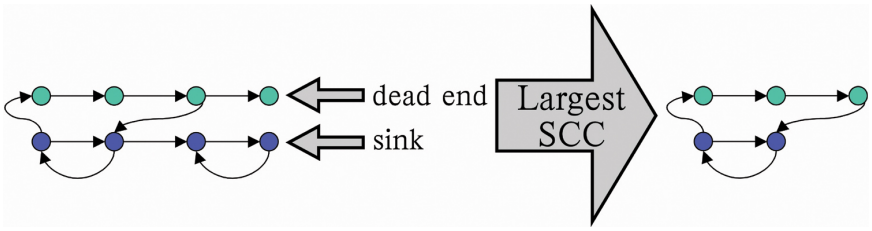


Figure 3.5: We remove sinks and dead-ends in the graph so that following transitions during motion synthesis does not restrict areas of the graph we can reach. Computing the largest strongly connected component (SCC) ensures each node is reachable from every other node.

and smoothing over discontinuities caused by the transitions.

Following edges in the graph selects pieces of motion to follow each other, and care must be taken to rotate and position the root properly. Each frame rendered must have the appropriate 2D rigid transformation applied so that the original motion data is aligned with the current position and orientation of the character. This corrective transformation begins as the identity transformation and remains so until an edge is followed that leads to a frame not sequential in the original data. When these transitions are followed, the transformation $T(f, t)$, that was used to align the frames for the distance metric calculation, is concatenated to the corrective transformation.

When a transition is made between two frames, a small jump will likely be visible unless the threshold for the distance metric was set very low. We smooth this discontinuity out over a small time window. The incoming and outgoing motion streams are overlapped by a small amount (0.25 seconds in most of our examples) and blended with time varying weights. The blend weights are varied using a sinusoidal "ease in/ease out" function. During blending, root positions are linearly interpolated while the quaternion orientations use spherical linear interpolation. Blending can introduce artifacts, the most common being the feet sliding on the ground. Lee *et al.*[32] eliminate this problem by only allowing transitions to occur when the window over which the blend is performed contains the same foot

contact state in each blended motion. The artifacts can also be eliminated in a post process step by strictly enforcing constraints. We leave it to future work to implement a more robust strategy to eliminate blending artifacts similar to those presented in the referenced works [16, 28, 34].

For the examples presented in this work, we use simple random graph walks when synthesizing motion under motion graph control. For more sophisticated applications, methods have been proposed to generate graph walks subject to user-defined constraints. Arikan *et al.* [1, 2] propose a solution appropriate for an artist directed animation tool by allowing users to specify pose and behavior constraints along a time line. Kovar and his colleagues [27] use an optimization to apply their graphs to the problem of synthesizing motion along a user specified path. In another work, Kovar [18] proposes automatically organizing the graph with "hub" nodes containing high connectivity to facilitate interactive control of a large variety of behaviors. Finally, Lee *et al.* [32] used clustering to aid in their searching while providing three interfaces for interactive control, including a performance-based one.

3.2 Physical Simulation of Human Motion

Simulation has been widely applied to generate realistic motion subject to the laws of physics. Video games, blockbuster movies, and animated films have all used physics to model the motion of both passive objects and living creatures. For human motion, physical simulation provides a way to model the interaction between a character and its environment in ways that cannot be anticipated at the time the motion data was captured or keyframed. Physical laws of motion subject to gravity and other external forces govern the movement of limbs that are constrained to move into only physically plausible configurations. This basic setup provides what is known in the interactive entertainment industry as "ragdoll physics." Like a ragdoll, it often looks lifeless, with limbs flopping because of a lack of control systems to coordinate limb movement and generate natural human behavior. Controllers provide the low-level mechanism to drive the limbs of the virtual character by com-

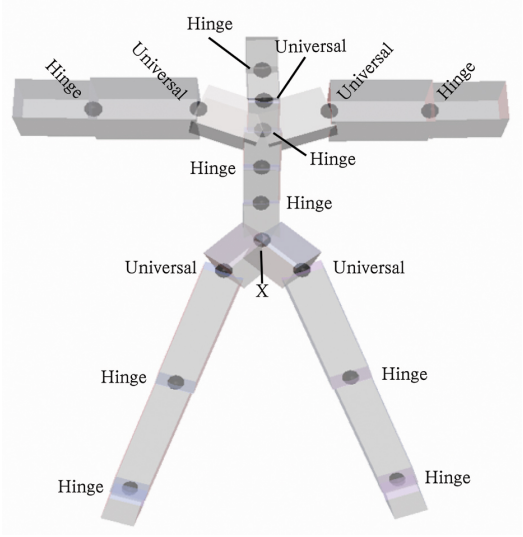


Figure 3.6: Each limb is represented in the simulation as a rigid box connected to other limbs by joint constraints. Joints are labeled to indicate whether they are represented by a hinge or universal joint (1 or 2 DOF). The root joint is marked with an 'X' because it constrains the connection between the lower back and hips not to move.

puting internal joint torques that approximate muscle actuation. This section begins with an overview of how we represent the human body as a series of rigid links and set up constraints to limit its movement to natural configurations. We later provide an introduction on how the equations of motion are solved by a rigid body simulator as well as how feedback control systems compute the joint torques used to control the simulation.

Our virtual characters are represented as an articulated figure consisting of a series of rigid links connected by joints. We use the freely available Open Dynamics Engine (ODE) to simulate the rigid body dynamics and resolve collisions between the simulated human and the environment [63]. ODE is supplied box primitives representing each limb, joints of appropriate type to connect each limb, and constraints that limit the movement of each joint to realistic ranges. We represent all 16 joints in our simulated model, the same used by our kinematic model described in Section 3.1.1. Each joint is specified as either a hinge or universal joint, with one and two degrees of freedom respectively. Figure 3.6 shows the

Joint	**Mass**	Low_x	$High_x$	Low_y	$High_y$
lfemur	6.051	-165.0	45.0	120.0	20.0
ltibia	3.541	0	165.0	-	-
lfoot	1.111	-11.5	11.5	-	-
lowerback	13.409	-45.0	90.0	-	-
upperback	10.0	-2.5	5.0	-	-
thorax	5.877	0	0	-	-
lowerneck	3.0	-50.0	89.0	-60.0	60.0
upperneck	5.877	-45.0	45.0	-	-
lhumerus	2.212	-85.0	85.0	-75.0	150.0
lradius	1.563	0	120.0	-	-

Table 3.2: A listing of the mass of each body (in kilograms) and joint limit constraints (in degrees). Two high and low joint limit ranges are given for Universal joints and one for Hinge, representing each DOF. The values are symmetric left and right.

structure of the simulated body with each joint type indicated.

Joint constraints are used to limit the motion of each joint to the physical limits of the human body. Lower and upper joint limits are supplied to the simulator as constraints for each degree of freedom. We derive our joint limits from those used by Faloutsos [9], originally generated through intuition about the body as well as from the biomechanical literature. Additionally, masses for each limb are specified using values from Hodgins *et al.* [22], derived from biomechanical data. Table 3.2.1 contains a listing of the joint limits for each degree of freedom in addition to the masses used for the rigid bodies attached to each joint.

3.2.1 Representation

The initial positions and orientations of each link are specified in a world Cartesian coordinate system. The simulation is initialized using a pose from our motion database to produce a smooth transition from motion graph control. The kinematic model used by the motion data, described in Section 3.1.1, uses a reduced coordinate system specified by the position of the root link with the orientations of links defined relative to their parent link's coordinate system. Because the dynamics library has no knowledge of hierarchy between links, poses in the reduced coordinate system must be converted to the global coordinate system before being supplied to the simulation. For our control algorithms, it is convenient to stay in the reduced coordinate system when computing the joint torques used to drive limbs to desired positions. We have implemented conversion routines to move between the global and reduced coordinate systems when necessary.

3.2.2 Equations of Motion

The motion of a simulated human's degrees of freedom q can be represented using a set of second order differential equations that relate Newton's law, the accelerations of the degrees of freedom, and the external/internal forces that act on the system. The external forces include those generated through collisions with the environment as well as gravity, while the internal forces are joint torques that approximate muscle actuation, generated by a control system. The equations of motion have the following general form:

$$\mathbf{M}(q(t))\ddot{q}(t) + \mathbf{C}(q(t), \dot{q}(t)) = \sum \mathbf{J}_T^T \mathbf{F}_i + \sum_l \mathbf{J}_R^T \tau_{ext,l} + \sum_k \mathbf{J}_R^T \tau_{j,k}$$

where $q(t)$ is the value of each degree of freedom at a given moment in time, **M** is a symmetric, positive definite mass matrix, **C** is gyroscopic forces, **J** is the Jacobian matrix, **F** is external forces, and τ_{ext} are internal joint torques (see Faloutsos [9] for more details).

The system listed above may be solved by discretizing how q changes over small time intervals (known as time steps). A numerical integration method is typically used to solve

a transformed version of the previous equation into a linear system $\mathbf{M}\ddot{q} = b$ (b is the total applied force) and solving for $\ddot{q}$. Integrating the computed accelerations twice will give the positions and velocities of each degree of freedom over the specified time step. Collision reaction forces must be generated to resolve collisions. An impulse or "penalty" method is one of the simplest methods used to quickly resolve interpenetration by generating stiff spring forces at contact points between bodies [38]. We use the iterative solver provided by ODE to achieve good performance, while sacrificing some accuracy for near-singular systems.

3.2.3 Controlling Simulation

The basic building blocks for the control system are target poses for the joints, as well as a method to compute the joint torques that will drive the joints toward these desired targets. For computing the torques, which are analogous to internal muscle actuation, we implement the commonly used proportional-derivative controller (PD-controller). One option for specifying target poses is to use sparse artist-directed poses separated by time or event-based transitions. Known as a pose controller, these target poses guide the simulation to key poses of a behavior, such as the layout or tuck positions of a diving motion [71]. A continuous controller generates target poses automatically from the current state of the system (positions and velocities of limbs). The target points for each joint are then continuously varying rather than changing in step increments, as with a pose controller. This feedback loop allows continuous controllers to be more robust to disturbances or changes in initial conditions than pose controllers, but requires an understanding of the behavior rather than tuning of key poses. For example, a continuous falling controller might look at how the shoulder and hip velocity evolve during the simulation and constantly adjust the target position of the arms to break the fall. A pose controller, by comparison, might look at the falling direction to determine which protective posture to target from among a small library of choices. Our falling controller, discussed in Chapter 5, uses a continuous

controller because it provides more adaptive behavior than a pose controller.

While using a controller may sound very similar to keyframing, there are some major distinctions. First, the inputs to the controller are *desired* joint angles, not *actual* joint angles. The controller will compute torques to drive the joint towards the desired value, but the joint is still subject to external forces such as those created by the environment and other limbs. Therefore, the joints may not reach the desired values. For example, an arm might be pinned to the ground under the body's weight, causing the joint controller to fail to generate enough force to move into the desired position. Second, if using continuous controllers, the desired pose does not come from a static keyframe or motion captured posture. The controller instead algorithmically specifies the target postures as a function of the system's feedback sensors. Finally, the character's global position and orientation is not directly specified by the controller, rather it evolves from interactions with the environment. The character will always fall under gravity, for example.

Computing Joint Torques Using Proportional-Derivative Controllers

We use PD servos to approximate the internal muscle actuation of the body, producing trajectories that move limbs from their current configurations towards desired ones. The inputs to the controller are the desired joint angles q_{des}, the state of the system $x = [q\dot{q}]$, and various sensor data such as the contact state of the hands. The output of the controller is the internal muscle torques applied to each joint, driving it towards the desired value. This is the classic setup for a closed-loop system – closed because it requires feedback of the system's state (see Dorf [8] for more on control systems). Figure 3.7 is a graphical representation of this basic controller feedback loop.

The PD controller computes internal rotational spring forces for each degree of freedom with the following equation:

$$\tau = k_s(\theta_{des} - \theta) + k_d(-\dot{\theta})$$

where k_s and k_d are stiffness and damping gains respectively, θ is the current joint angle,

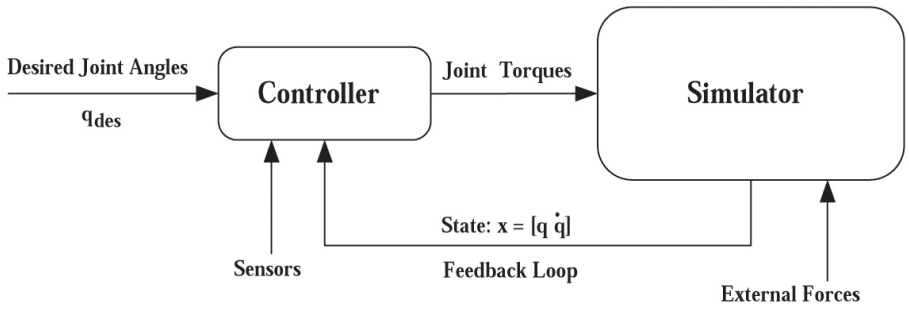


Figure 3.7: The basic closed-loop feedback system utilized by our PD controllers.

θ_{des} is the desired joint angle, and $\dot{\theta}$ is the velocity of the degree of freedom.

Tuning the stiffness and damping gains is critical to achieving natural looking movement. The stiffness gain controls the strength of the spring while the damper gain adjusts how smoothly the joint arrives at the desired value. If the system is under-damped, an oscillating response will occur as the joint overshoots the desired value, while over-damping will give an overly slow progression towards the desired value. Critical damping is achieved when the stiffness and damping gains are tuned such that the joint arrives at the desired value as quick as possible, with little or no overshoot. We hand tune the gains, as is traditionally done, and this process can be somewhat time consuming. An alternative approach is to use a heuristic to scale the gains based on the moment of inertia of the chain of bodies connected to each joint [73].

3.2.4 Example: Falling

We now compare results illustrating the difference between ordinary ragdoll physics, which only models the passive motion of the body, and simulation augmented with PD-controller driven behavior. Figure 3.8 shows a side-by-side comparison of a controller-based falling behavior with ragdoll simulation. The fall controller, covered in detail in Chapter 5, generates a more realistic behavior by using the arms to protect the the head during impact.

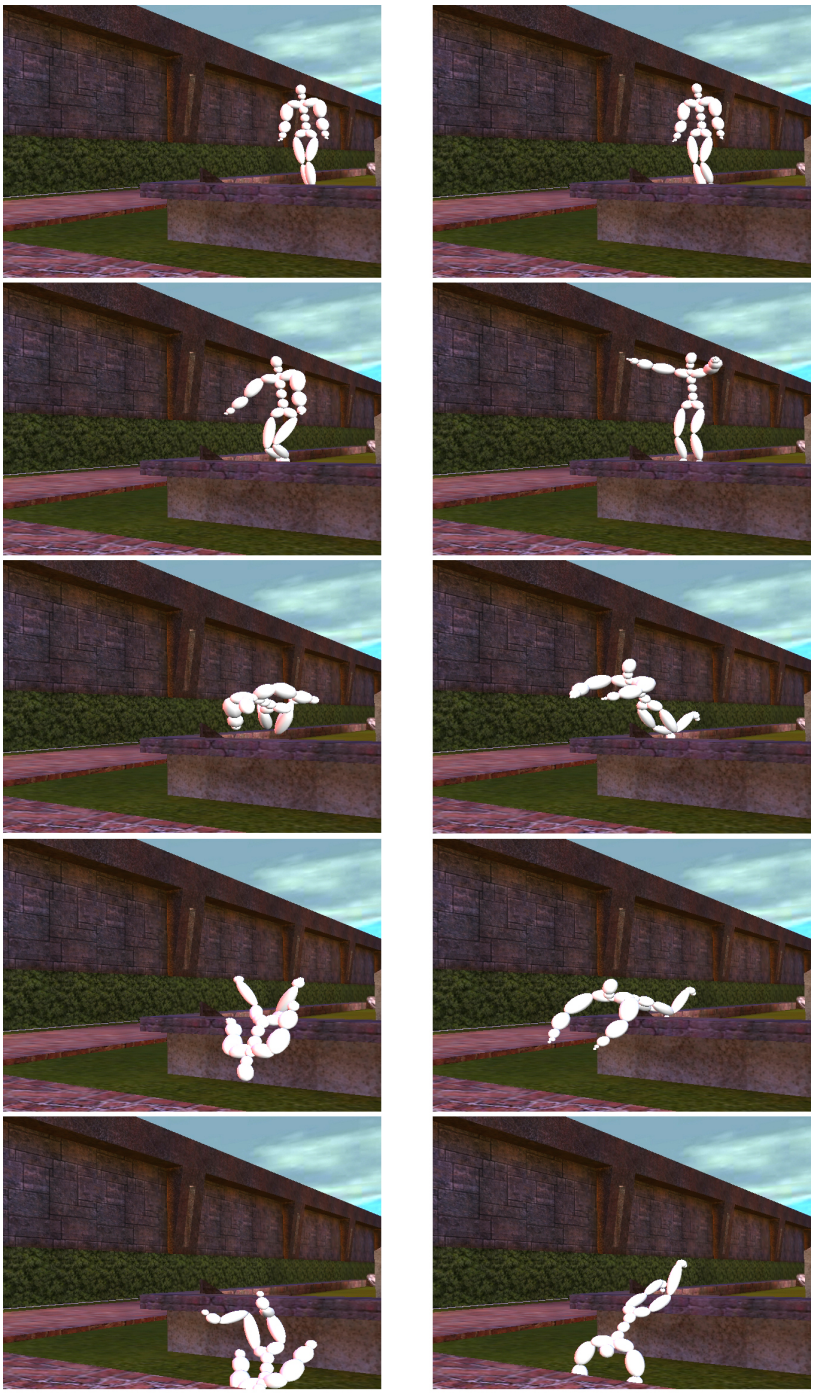


Figure 3.8: A side-by-side comparison of ragdoll physics and a controller-based simulated behavior. The PD-controller driven behavior appears more realistic by breaking the impact with the arms, while the ragdoll's lifeless movements cause a landing directly on the head.

Chapter 4

Building a Correspondence Between the Motion Database and the Simulation

Our hybrid system allows transitions between simulated motion and motion data, as well as the reverse. These transfers allow the best motion synthesis technique for a particular situation to be selected. When considering how to perform these transitions, its important to understand the state space of representable poses for each method. During data-driven control, the motion graph may change the pose of the character to any discrete posture in a motion database. The simulation, on the other hand, has a much larger state space that includes the continuous space of physically plausible configurations allowable by the joint limit constraints, including unnatural ones unlikely to be present in a motion database. In order to transition between the techniques in real-time, we need a fast method to create a correspondence between postures generated by the simulation and those contained in the motion database. Transitioning to the simulation from the motion graph simply involves setting the initial conditions of the simulation based on the pose and velocity characteristics of the motion data. Transitioning from simulation to motion data is harder, because the larger state space of the simulation makes it unlikely to be exactly in a pose present in the motion database. Our strategy is to quickly find a pose in the motion database that is

sufficiently close to the simulated posture and drive the simulation towards this pose.

This chapter defines a correspondence between simulated postures and those found in a motion database. We first describe a fast search algorithm, based on Approximate Nearest Neighbor search, to find closely matching poses in the motion database to a simulated posture. Part of the correspondence process is realizing *when* each technique is appropriate and *how* transitions between the techniques are carried out, thus we conclude the chapter with a discussion of these topics.

4.1 Nearest Neighbor Search

Our problem consists of finding frames in our motion database that are similar to a query posture generated by the simulation. More specifically, we want to find the n closest matching frames to our input posture and make a selection based on the most situationally appropriate result. Comparing the query posture to every frame in the database in a brute force manner would be prohibitively expensive. Thus, we need a fast search strategy that exploits features in our data to speed up the selection process. Nearest neighbor search is a class of algorithms designed to solve such problems.

The basic k-nearest neighbor (KNN) search problem takes as input a d-dimensional set of data D of size n. D is preprocessed into a data structure so that given any query point q, the k most similar points in D can be reported efficiently. Minkowski similarity functions are typically used for the distance metric, such as the Euclidean Distance metric that we use. Our data is high dimensional, so performance considerations are important if we expect to get real-time performance. Spatial data structures [56] speed up query times by exploiting spatial relationships in the data. Still, most algorithms tend to suffer an exponential decrease in performance as the dimensionality of the input data increases.

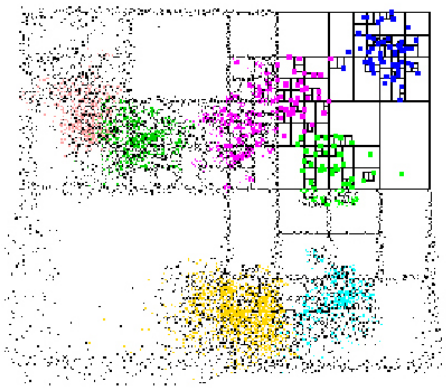


Figure 4.1: A visualization of sample data inside the *bbd-tree* used by the ANN search. The image becomes fuzzier as the distance to a query point increases, representing ANN's randomized nature.

4.1.1 Approximate Nearest Neighbor Search

We base our search strategy on the approximate nearest neighbor (ANN) algorithm developed by Mount *et al.* [41]. By using a randomized search strategy and relaxing constraints on search accuracy, the ANN algorithm does considerably better than standard KNN algorithms in both space and time performance for high dimensional data. ANN search introduces an ϵ parameter that guarantees any nearest neighbor result is within a factor of (1 + ϵ) distance of the actual nearest neighbor. This relaxed specification gives a tangible guarantee on accuracy while providing $O(log^3 n)$ expected run time and an $O(nlogn)$ space requirement. The algorithm first finds the cell the query point is located in its spatial data structure of the input data points. A randomized search then finds surrounding cells that contain input data points within the given ϵ threshold distance from actual nearest neighbors.

We selected the balanced box-decomposition tree (*bbd-tree*) data structure [3] to speed up the initial spatial query made by the ANN search algorithm to locate the leaf cell containing the query point. The *bbd-tree* is able to fit input data more accurately than *kd-trees*

[13] by allowing special shrink nodes in its tree representation. A spatially tight cluster of input points may be immediately bounded in one step in the tree by substituting the traditional axis-aligned hyper-rectangles used by *kd-trees* with these shrink nodes, essentially a hyper-rectangle within a hyper-rectangle. Figure 4.1 contains a representation of a sample 2D data set loaded into a *bbd-tree* used by the ANN search. The implementation of our search algorithm utilizes the freely available ANN C++ library [42].

Data Representation

Nearest neighbor search algorithms require that each input data point be a parameter vector of fixed dimension *d*. We must ensure that visually close human postures produce parameter vectors that evaluate to smaller values using the Euclidean distance metric. Our strategy is to ignore the root translation and align each posture along a coordinate axis so that relative orientation does not affect the comparison of two similar poses. After alignment, 3D joint positions are extracted for each posture. Each parameter vector has d=86 so that the 3D position of each aligned joint is represented. We also support including surrounding frames in the motion data as a way of encoding velocity. Because our examples utilize the search process during low velocity, prone body positions, the velocity encoding did not increase the search accuracy sufficiently to warrant the increase in dimensionality.

Performance

We now present a performance comparison of our implementation of the ANN search algorithm to a brute force method. We tested both algorithms on a series of queries and measured their runtime with respect to motion database size. We created a sample motion database of $13,183$ frames (about 440 seconds) containing several behaviors including walking, running, idling, sneaking, and jumping. Each trial used a subset of the full database and measured the average runtime over 150 searches using query points randomly selected from the database. Figure 4.2 shows the performance results for both algorithms as

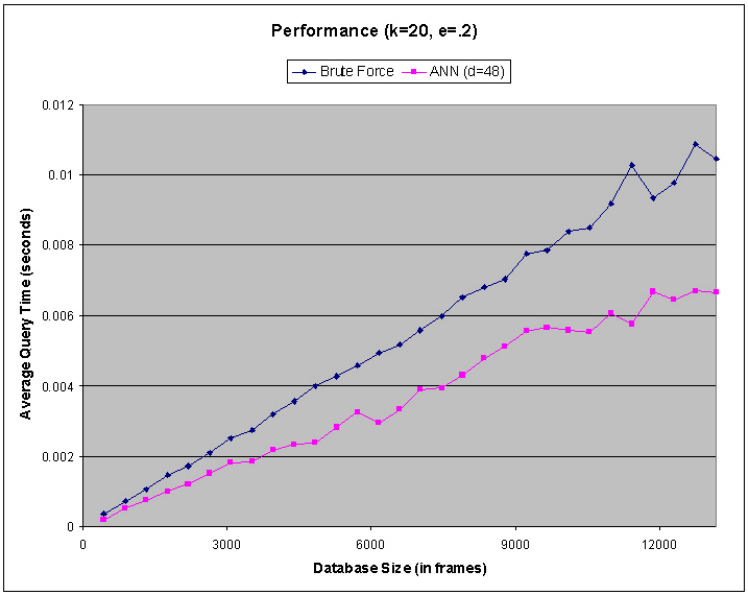


Figure 4.2: Performance comparison of ANN and brute force search. Each trial recorded the average query time over 150 random queries for increasingly large sample motion databases.

the database sample size increases. The results suggest that ANN is effective, in practice, in speeding up query times over brute force.

4.2 Transitioning Between Motion Graph and Simulation

In a hybrid system like ours that utilizes both simulation and motion data for synthesis, it is important to determine when a particular synthesis technique is appropriate because neither approach alone provides a solution in all situations. We must carefully determine which technique is best suited to the current situation based on whether we can simulate the requested behavior or how well it is represented in the motion database. The following sections outline under what conditions each technique is triggered within our system and how the transitions are carried out.

4.2.1 Motion Graph to Simulation

As explained in Chapter 3, we use motion graphs to provide high-level control of our virtual character, capable of utilizing large motion capture databases and generating smooth transitions between various behaviors. Characters under motion graph control can perform a variety of behaviors such as running, walking, sneaking, and jumping. Unfortunately, it is difficult, even with very large motion databases, to plan for interactions with an arbitrary environment and external forces. We do not want body parts intersecting objects in the environment and would like the character to respond realistically when external forces are applied by things such as dynamic obstacles, wind, gravity, and even other characters.

The simulated behaviors we implement deal with dramatic loss of balance that would be extremely difficult to model with motion data both because of the difficulty of matching the environmental conditions and the danger to the motion capture actor. Therefore, external forces of significant magnitude applied anywhere on a virtual character are used to trigger a transition to simulation. In realistic training environments, significant collisions with hazards and obstacles in the environment (either static or dynamic) could trigger a transition to simulation. For maximum flexibility in testing, we devised a user interface that allows the selection of any body part as well as specification of an arbitrary force vector. Figure 4.3 shows how a simulation is triggered from a motion graph controlled character using our interactive user interface. The transition to simulation is carried out by simply setting the initial positions and velocities of each simulated limb to those extracted from the motion data so that the transition is seamless.

4.2.2 Simulation to Motion Graph

During simulation, when a behavior is required that cannot be simulated, a transition must be made back to motion graph control. Because we simulate dramatic loss of balance, we usually (see *Chapter 5*) activate motion graphs to allow the character to stand up naturally

Figure 4.3: Triggering transition to simulation from idling behavior. From left to right: Selection of a body part, specification of force vector to apply, and beginning of simulated fall.

after a fall. In most cases, the simulation will have significantly changed the posture of the character during the time that is was active. Thus the simulation is not likely to be in a pose close to any frame in the motion database. When the arms impact the ground, the ANN search is used to find the closest frame in the database to the simulated posture and a special controller is used to allow the simulation to naturally settle near this frame. Using an arm contact event to trigger the search process provides a balance between allowing a natural falling behavior and providing time for settling near motion data before the body comes to rest. The details of this controller are covered in *Chapter 5*. When the settle controller has moved the character sufficiently close to the pose found in the search, a transition to motion graph is performed using the same blending scheme used to carry out the motion graph transitions described in Section 3.1.4. Although it would have been possible to achieve this with only blending, the settle controller is preferred because it still allows the character to remain subject to physical constraints, respond to the external environment, and avoid some types of blending artifacts such as object interpenetration and foot sliding.

Chapter 5

Simulating Human Behavior

Controllers provide a means to infuse life into a simulation by driving the limbs towards target locations determined by a high level behavioral module. If this module is well designed, it will create behaviors that mimic how a person would move his or her limbs in a similar situation. The most general version of this problem would require a deep understanding of how people coordinate their limbs while moving. We simplify the problem greatly by looking at manageable behaviors that might not require our controllers to directly specify every degree of freedom of the body. In particular, we generate reflexive responses to falling that use the natural dynamics of the system to control some of the degrees of freedom, and because we do not expect the simulation to return to a balanced posture, the control problem is simplified. The hybrid system allows the difficult control problem of returning to a balanced standing posture from a prone or supine position to be handled by our data-driven synthesis approach.

The controllers in our system are used for two purposes:

1. Biomechanically inspired falling reactions designed to realistically protect the body under a wide variety of plausible physical interactions.

2. Ensure that the simulated behavior settles into a configuration near existing motion data to allow character control to return to a motion capture graph.

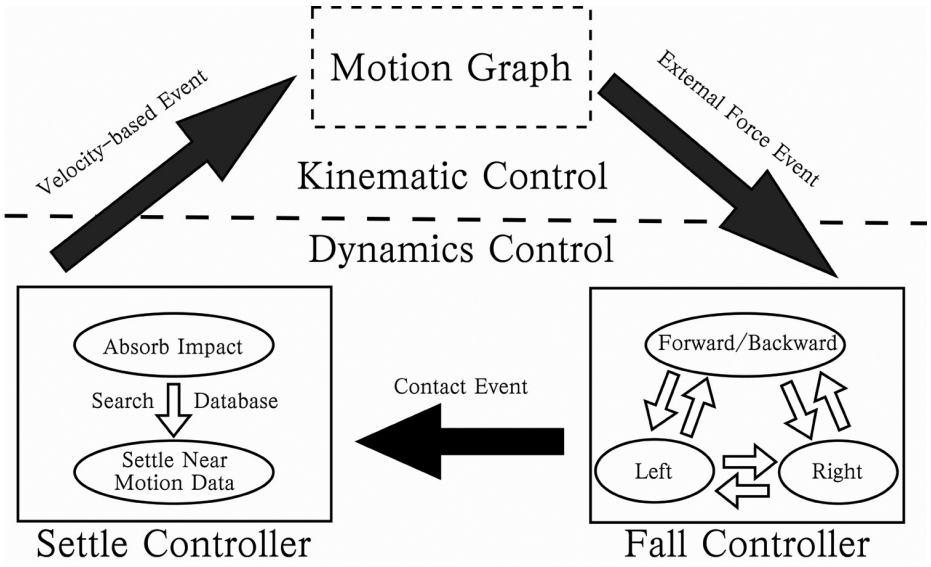


Figure 5.1: High-level states of our system. Solid black arrows indicate the transition event used to move between each high-level state. The fall and settle controllers are governed by their own finite state machines.

Physical controllers compute the internal muscle torques the virtual character must exert at each joint to perform a given motor task in a natural way. The torques are computed by proportional-derivative controllers (PD-controllers) that were introduced in Chapter 3. The controllers provide the low-level control used to drive the body towards target postures determined by a higher level mechanism. This chapter focuses on how these target postures are generated in our system to simulate natural human behaviors.

Finite state machines are a common representation for managing transitions between individual motor control states [10, 22, 30]. Each state produces a set of desired joint angles for each degree of freedom of the virtual human. The desired poses may be computed procedurally using biomechanical knowledge and the current state of the simulation as well as with poses from a motion capture database. Transitions between control states may be time or event based. Figure 5.1 illustrates the high-level state machine used in our system to generate transitions from motion graphs to the falling controller and back to motion graphs. Details of the controller states, target pose computations, and transition conditions

are covered in the following sections.

5.1 Design of Falling Controllers

This section describes the controller that generates motor control strategies for a falling human. The goal of this controller is to produce biomechanically inspired, protective behaviors in response to the many different ways a human may fall to the ground. The following sections address biomechanical considerations, continuous generation of target postures, and determination of control state.

5.1.1 Biomechanics of Falling

The human body displays a remarkable ability to quickly respond to unexpected impacts and prevent injury. The initial conditions for falls are varied, yet humans still instinctually generate motor control strategies that protect the body in many situations. Researchers in the biomechanics community have looked at unexpected and trained responses to slips induced by various perturbations. Robinovich *et al.* [51] remarks that even though the total energy available in a typical fall far exceeds that required to fracture a hip or wrist, most falls do not result in injury. This fact suggests highly effective motor strategies to protect the body during a fall. For example, people often take corrective steps or attempt to use nearby objects as hand holds when attempting to recover balance [7, 36, 37]. Cham and Redfern [6] found that the ankles play a much smaller role than the knees in these corrective strategies. They also found reaction times for the lower body to start at around 190 to 350 milliseconds after impact. Maki *et al.* [37] argue that falling is highly dependant on rapid movement of limbs to alter the base of support, with arms being the most rapid to react. He then shows how strategies are further complicated by the current ongoing activity. Maki [36] also quantifies the biomechanical limits on the body's ability to recover balance from an impact through the concept of a velocity stability margin. Finally, this work also finds

that an overly quick response to an impact can lower the person's ability to control his/her stability.

While the biomechanics literature provides many insights into the motor control strategies used by humans, translating the research into robust control strategies for virtual characters is difficult. Our understanding of what happens at the later points of falls is limited because experiments must ensure subject safety and therefore limiting harnesses are often used. Collecting natural responses to unexpected slips in a laboratory setting is difficult because the subject will be expecting a slip or a trip after the first trial.

The following biomechanics inspired rules motivated the implementation of our control strategies:

1. Falls are not always broken with both wrists [51].
2. The upper body tends to impact before the pelvis [51].
3. Upper body and pelvis kinetic energy are actively reduced at impact [51].
4. Corrective responses start occurring after 200 milliseconds on average, with arms being slightly faster [6, 37].

5.1.2 Continuous Control of Falling Behavior

Modeling the way people fall is a difficult task not only because the experimental data is limited, but also because people may vary greatly in their physique and prior training. Our controllers aim to model an average healthy adult's involuntary reaction to arbitrary impacts that cause a full loss of balance. The aim of this work is not to respond to small impacts that could be resolved with a corrective step or other reactive responses (see Zordan [73]). We look at impacts that create a loss of balance that requires protective behaviors to avoid injury.

Falls are very dynamic, thus we use continuous pose control that is coupled to the

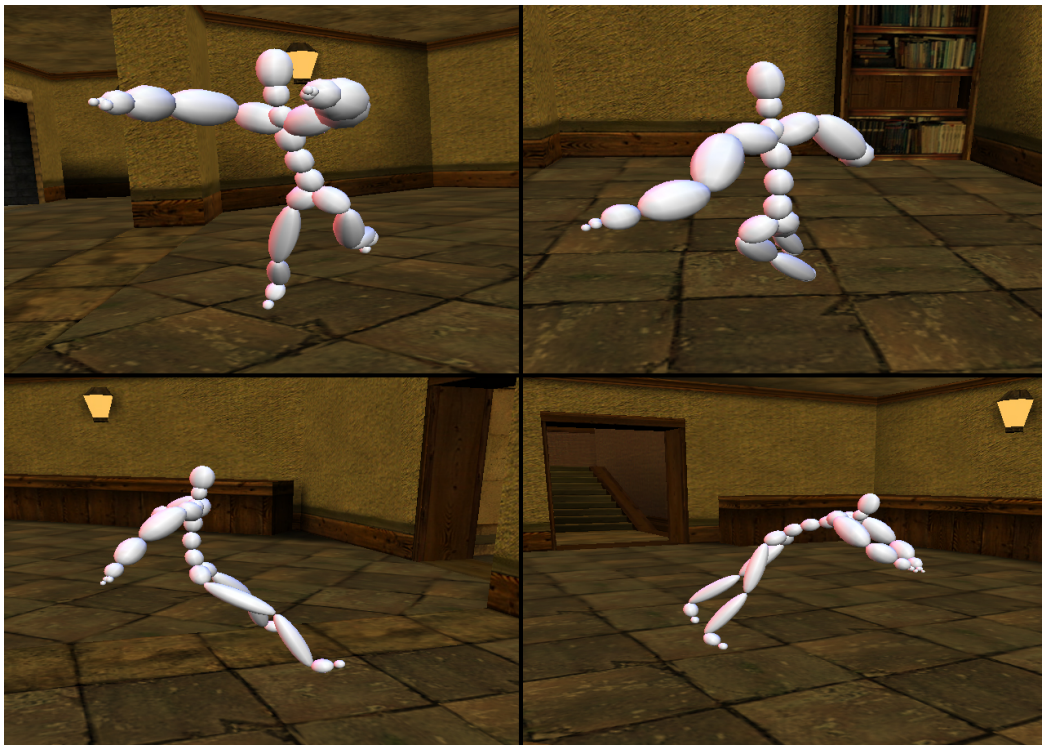
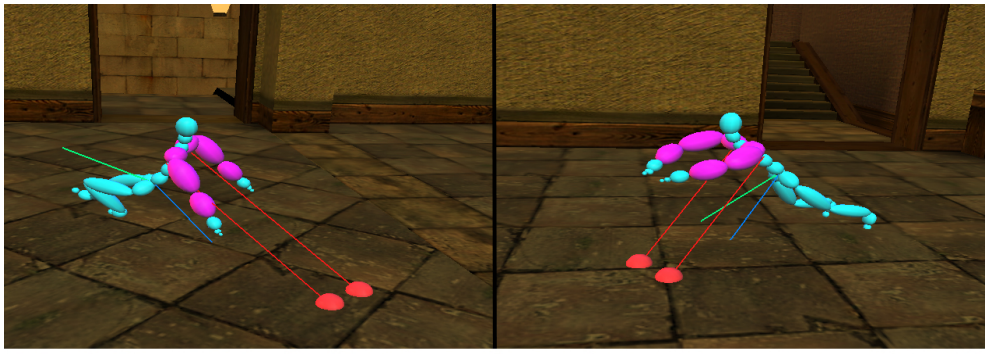


Figure 5.2: The fall controller in a variety of situations producing forwards, backwards, and sideways falls.

simulation. The predicted landing point of the shoulders is derived from shoulder velocity and used to generate a continuous stream of protective target postures. Depending on the fall direction, we determine the current controller state which causes one or both arms to actively track the corresponding predicted shoulder landing position. Figure 5.2 shows the fall controller responding to a variety of initial conditions. We now describe how the controller state is determined and specifics of the target posture generation.

Determining Controller State

Once the fall controller is activated, it may be in one of four states: Forward, Backward, Left, or Right. These correspond to the computed falling direction of the character. Velocity-based transitions may be made continuously between these four states as the



Backwards Forward

Figure 5.3: Example falls showing predicted landing positions of shoulders as red spheres. Used to determine current controller state, the blue vector indicates average limb velocity and the green vector indicates current facing direction. Controller gains and initial the distance from the target dictate the speed with which the arms will align with the predicted landing locations.

character's falling direction changes, providing a more dynamic response as the simulation evolves. The controller is divided into four states because the required response will differ depending on fall direction. The fall direction is computed using a dot product of the current facing direction of the human and the average velocity of the limbs:

$$fallState(\vec{\theta}_d, \vec{V}_{body}) = \begin{cases} Forward & \text{if } \vec{\theta}_d \cdot \vec{V}_{body} \geq .5 \\ Backward & \text{if } \vec{\theta}_d \cdot \vec{V}_{body} \leq -.5 \\ Right & \text{if } 0 < \vec{\theta}_d \cdot \vec{V}_{body} < .5 \\ Left & \text{if } -.5 < \vec{\theta}_d \cdot \vec{V}_{body} < 0 \end{cases}$$

where $\vec{\theta}_d$ is the facing direction of the character and $\vec{V}_{body}$ is the average velocity of the body.

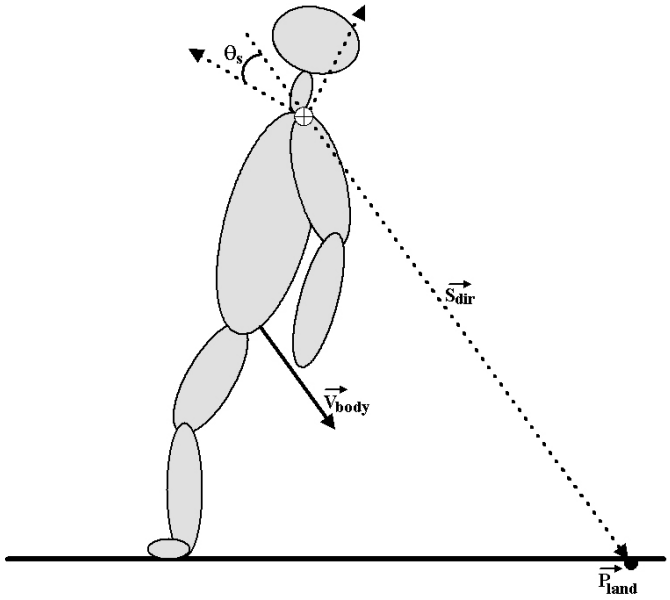


Figure 5.4: A 2D illustration of how the target joint angles of the shoulder are determined during a fall. The predicted shoulder landing position, $\vec{P}_{land}$, is determined by intersecting the vector $\vec{S}_{dir}$, which is parallel to $\vec{V}_{body}$, with the ground. θ_s indicates the relative joint angle of the shoulder necessary to align the arm with $\vec{S}_{dir}$.

Protective Maneuvers

During each of the fall controller's four states, one or both arms track the predicted landing point of the shoulders. The predicted landing positions of each shoulder are computed by intersecting the geometry of the environment with a ray from the shoulder pointing in the direction of $\vec{V}_{body}$. Figure 5.3 shows the predicted shoulder landing positions for various falls. Depending on the state of the fall controller, each arm will track the predicted landing location of the shoulder. For falling backwards and forwards, both the left and right arms are actively controlled to track the shoulder landing position. For falling sideways, only the arm in the direction of the fall tracks the landing position.

The goal of each actively controlled arm is to have the wrists intersect the line between the shoulder and its predicted landing position. Desired angles for the shoulder joint are

Joint	k_s	k_d	**Joint**	k_s	k_d
root	(0.0, 0.0)	(0.0, 0.0)	thorax	(31.5, 0.0)	(10.5, 0.0)
lfemur	(90.0, 90.0)	(22.5, 22.5)	lowerneck	(31.5, 0.0)	(10.5, 0.0)
ltibia	(15.0, 0.0)	(3.0, 0.0)	upperneck	(26.25, 0.0)	(4.2, 0.0)
lfoot	(4.5, 0.0)	(1.5, 0.0)	lhumerus	(360.0, 360.0)	(108.0, 108.0)
lowerback	(120.0, 0.0)	(45.0, 0.0)	lradius	(90.0, 0.0)	(24.0, 0.0)
upperback	(42.0, 0.0)	(15.75, 0.0)			

Table 5.1: Spring (k_s) and damper (k_d) gains used for each state in our falling controller. When falling sideways, gains are lowered for the shoulder joints that are not tracking the predicted landing position. The gains are symmetric left and right.

computed as illustrated in Figure 5.4. A small natural bend is added to the elbow and the desired angles for the rest of the body are set to initial angles at the time the fall controller is activated. Lower gains are used for the rest of the body to keep it from appearing too stiff. Table 5.1 contains the spring and damper constants used by the PD-controller during a fall.

5.2 Controlling Towards Motion Data

The posture of the simulated character after a fall cannot be easily predicted because interactions with the environment and external forces will be different for every fall. For example, arms may be pinned by the body to the ground and body parts can come to rest against objects in the environment. If we want to return character control to pre-recorded motion data, we need to ensure the simulation settles into a configuration close to one in the motion database.

The goal of allowing the simulation the freedom to naturally reproduce a behavior conflicts with the need to have the character settle into a pose near existing motion data. Be-

cause of the difficulty of recording data where all the markers are not visible, we may only have a few clips in the database of the character standing up from a prone or supine position. If the controller waits too long to try to settle near one of these clips, an arm might be pinned to the ground underneath the body, preventing the simulation from matching the prerecorded poses. Moving towards motion data early in the fall would eliminate the protective movements necessary to naturally break the impact of the fall. The goal of the settle controller is to reasonably resolve this conflict by driving the character to similar motion clips at an appropriate time. We activate the settle controller when the hands first make contact with the environment as a result of the fall controller. The following section will describe the individual controller states of the settle controller.

5.2.1 Settle Controller States

Beginning at the moment the hands impact the ground, the controller moves the limbs of the body toward a frame in the motion database while maintaining visual continuity with the exiting fall behavior. The settle controller has two states:

1. *Absorb Impact*: Gains are adjusted for the body to naturally absorb the impact with the ground thus reducing the velocity of the hips and upper body. This state also avoids an abrupt movement towards an arbitrary (but hopefully close) pose from the motion database. A time-based transition is made (currently a half second) to the ANN Search state.

2. *ANN Search*: An ANN search (see Chapter 4 for details) is executed to quickly find a frame in the motion database that is close to the current simulated posture. This frame is then used as the target joint angles while continuing to absorb the impact of the fall.

Table 5.2.1 shows the spring and damper constants used for each state in the settle controller. When the pose of the simulated human is within a threshold distance to the target

Joint	Absorb	Impact	ANN	Search
	k_s	k_d	k_s	k_d
lfemur	(72.0, 72.0)	(18.0, 18.0)	(270.0, 270.0)	(67.5, 67.5)
ltibia	(12.0, 0.0)	(2.4, 0.0)	(45.0, 0.0)	(9.0, 0.0)
lfoot	(3.6, 0.0)	(1.2, 0.0)	(13.5, 0.0)	(4.5, 0.0)
lowerback	(96.0, 0.0)	(36.0, 0.0)	(360.0, 0.0)	(135.0, 0.0)
upperback	(102.0, 0.0)	(38.25, 0.0)	(180.0, 0.0)	(67.5, 0.0)
thorax	(76.5, 0.0)	(25.5, 0.0)	(135.0, 0.0)	(45.0, 0.0)
lowerneck	(76.5, 0.0)	(25.5, 0.0)	(135.0, 0.0)	(45.0, 0.0)
upperneck	(63.75, 0.0)	(10.2, 0.0)	(112.5, 0.0)	(18.0, 0.0)
lhumerus	(108.0, 108.0)	(32.39, 32.39)	(120.0, 120.0)	(32.3, 32.3)
lradius	(63.0, 0.0)	(14.4, 0.0)	(90.0, 0.0)	(21.59, 0.0)

Table 5.2: Spring (k_s) and damper (k_d) gains used for each of the two states of the settle controller. The gains are symmetric left and right.

posture determined by the ANN search, a transition is made to motion graph character control. The motion generated by the simulation is smoothly blended into the motion data using the interpolation scheme covered in Section 3.1.4. Character control then continues using the motion graph until the fall controller is triggered again.

5.3 Example: Rolling Behavior

The fall controller represents one kind of behavior that is more appropriate to use simulation for than motion data. However, with a future goal of fully autonomous simulated characters, it is important to understand how extra behaviors could be used to adapt to new situations. A high-level planner may determine that it is urgent that the character regain balance quickly after falling, and thus a rolling behavior might allow the character to get back on his feet

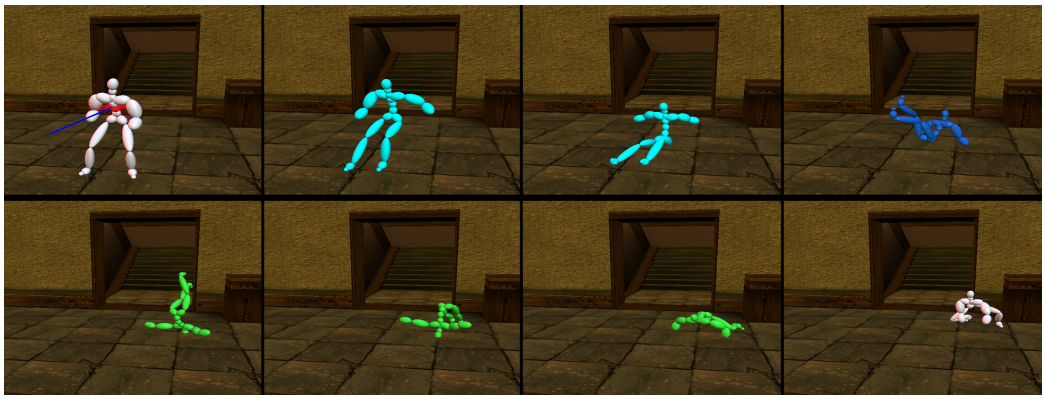


Figure 5.5: A sequence of images showing the fall controller transitioning into a simulated rolling behavior. The initial force vector is indicated in blue, and the larger this force is, the more likely a roll is to occur. The character is colored based on the current controller state (see Figure 6.3 for the color/state mapping).

faster.

As an example of this sort of approach, we implemented a simple rolling behavior that is activated based on the momentum of the human when impacting the ground. Our current system activates a rolling state only when falling backwards and uses a strategy that kicks both legs up in an attempt to generate a backwards roll. Figure 5.5 shows a fall sequence that turns into a roll. While this example is simple, it illustrates how future hybrid systems may easily add more sophisticated behavior.

Chapter 6

Assessment

This chapter discusses the capabilities of our hybrid system. We set up a number of experiments to assess the robustness of the system under a variety of initial conditions. We first apply a controlled set of input forces to the character to examine the performance and adaptive nature of the fall controller. We then apply the same set of inputs to a motion graph controlled character to assess the quality of the hybrid system performing a transition to the simulated fall controller and a return to motion graph control. This assessment shows that the system is capable of producing appropriate transitions between data-driven and dynamics-based motion synthesis and that the simulated behaviors appear natural for the tested situations.

6.1 Results

For each result sequence shown in the following sections, a controlled set of initial conditions are supplied by the user. The user can apply a force to a selected body part in one of 48 different directions. These forces are shown in Figure 6.1 for the pelvis. The 48 directions were chosen as a reasonable sampling of the external forces the character may encounter. The strength of the force is also a user-defined parameter. We used forces applied to the

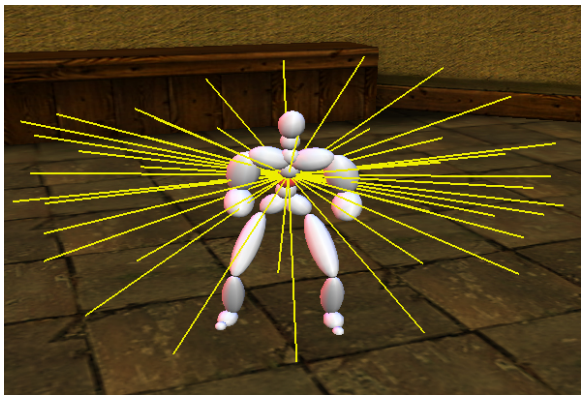


Figure 6.1: A visualization of the 48 experimental force vectors available to the user to test the adaptiveness of our hybrid system.

selected body part for a single time step that varied between 2500-7000N. Each sequence, including both the simulation and rendering, was generated at approximately 50 frames per second on a 2.4Ghz Pentium 4. The following sections examine the results of applying these experimental forces to the character to test our fall controller and the overall hybrid architecture.

6.1.1 Fall Controller

Our fall controller, described in Chapter 5, uses physical simulation to generate biomechanically inspired protective falling behaviors. We now assess the controller's ability to generate human-like behavior that naturally adapts to different initial conditions. We activated the controller from a variety of initial body poses (supplied by frames from our motion database) while applying an initial force. The experimental force vectors were applied with varying strengths to the pelvis, head, shoulders, and legs. Most of the experiments assessed the performance of the controller on flat terrain. However, the system also produced natural results when the character interacted with more complicated environments. The controller generates a protective behavior by projecting the shoulder velocity onto the environment's geometry and controlling the arms accordingly. This control scheme is robust to differ-

ent terrain geometries as the estimated shoulder landing positions is still as accurate under these conditions as it is for the flat ground plane case. However, the shoulder velocities are not filtered, causing the predicted landing locations to exhibit less temporal coherence than desired. In addition to filtering the velocities, accounting for a more ballistic shoulder trajectory may have also increased the accuracy of the landing estimates. Figure 6.2 contains four sequences of falls generated using a variety of initial conditions. The bottom row illustrates a more complex environment.

The falling controller generates reasonable behavior for the inputs in our experiments. Our controller generates responses only when a complete loss of balance occurs in the character, but does not account for simple balancing reactions that could prevent a fall. It also does not consider the variety of instinctual responses people often exhibit in the presence of hand holds in the surrounding environment. Chapter 7 contains a discussion of enhancements to the fall controller that would increase its robustness.

6.1.2 Hybrid Control

This section presents the results of our experiments with our hybrid system. Each experiment demonstrates a contextually appropriate and visually smooth transition from a motion graph controlled behavior to a physically simulated behavior, and back to the motion graph. The user may apply a force vector to a character that is kinematically driven by a motion graph, triggering the dynamics-based fall controller. As described in Chapter 5, the fall controller leaves the character in a prone or supine position on the ground. The settle controller is activated on impact, and generally causes the character to settle near an existing clip of motion data. This process relies on the ANN algorithm, described in Chapter 4, to build a correspondence between the simulation and motion database so that motion graph control can resume without creating noticeable glitches in the motion. A small set of motion clips are included in the database to bring the character from a prone or supine position to an upright, balanced posture. Returning to the balanced posture, a particularly hard be-

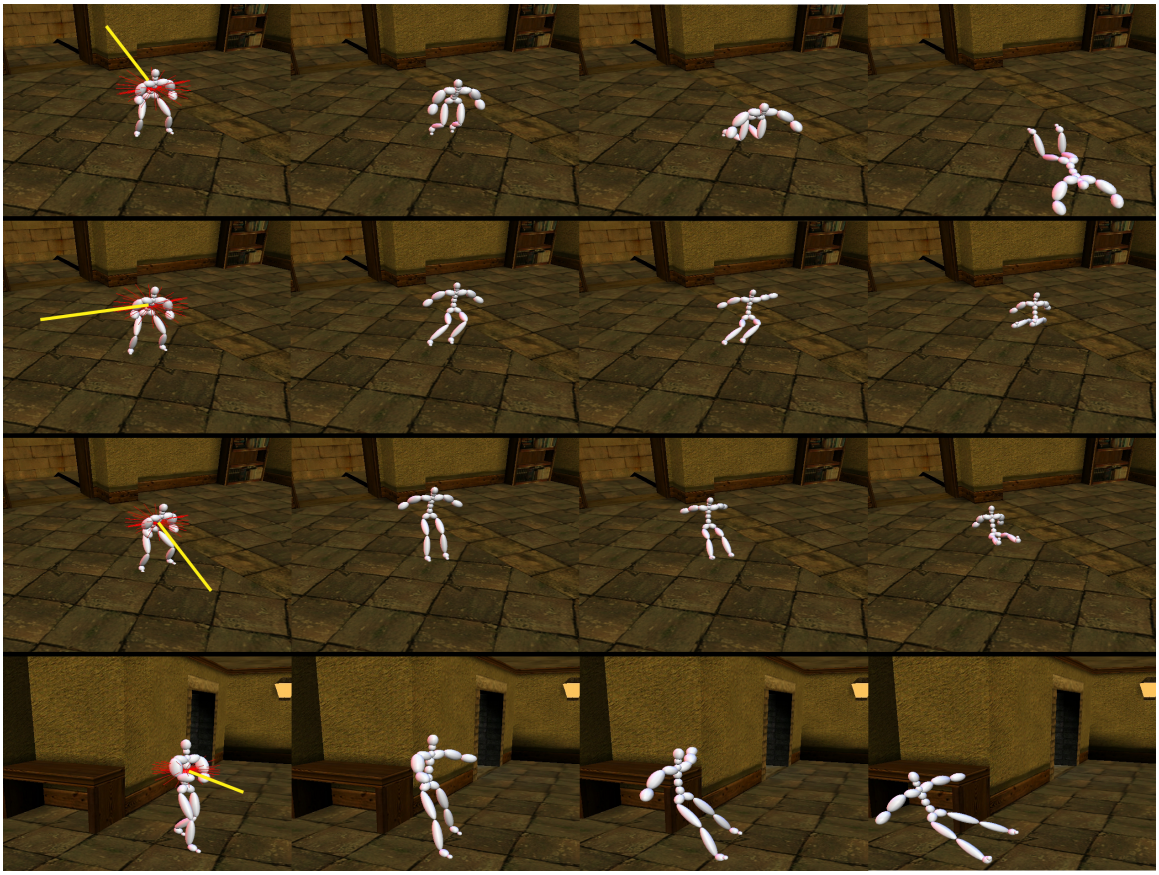


Figure 6.2: Four sequences of results illustrating the behavior generated by the fall controller using different initial conditions. An initial force is applied to the pelvis, indicated by the yellow vector in the first frame of each sequence (the red vectors indicate the other choices available to the user). The first three sequences occur in an open space while the last sequence shows a more complex interaction with an object in the environment.

havior to simulate, is handled by the hybrid system by relying on these clips to accurately reproduce the behavior. Because the clips that achieve this behavior are part of a motion graph, transitions are automatically present that allow other behaviors to continue.

Our first example uses a relatively small database containing 2,867 frames of motion (about 95 seconds) containing various idling motions as well as 5 clips of motion that move the character from prone and supine positions to an idle stance. The database is pre-processed into a motion graph, as described in Chapter 3, to automatically connect the clips with natural transitions. At runtime, we randomly traverse the motion graph to generate a seamless, infinite sequence of idle motion until the user applies a force to the character, activating the fall controller, and eventually, the settle controller. The settle controller should cause the character pose to match a pose from one of the five clips of motion that can return the character to a balanced posture. Control is then transferred to the motion graph, bringing the character to an upright posture and allowing a natural transition back to the original idling behavior. Figure 6.3 contains a sequence of images where the character is color-coded to indicate the current state of the system (motion graph, fall, settle). We also conducted a similar experiment using a sneaking behavior for the motion graph control, with a database of 1,362 frames (about 45 seconds). Figure 6.4 presents a sequence where a force is applied to the pelvis and causes a forward fall followed by a recovery to the original sneaking behavior.

The hybrid system successfully generates natural transitions between data-driven and dynamics-based motion synthesis for a wide variety of user-specified input parameters. However, the system's flexibility is limited by the availability of motions that reasonably connect plausible simulated postures to the motion database. Those sequences are particularly difficult to capture because of the number of markers that are occluded when an actor is lying down or bent over and it is challenging to cover the space of all possible outcomes of the simulation. Except in situations where the body parts are pinned between the body and the ground plane or other objects in the environment, the settle controller ensures the simulation will move the body pose near existing motion data. There are no guarantees

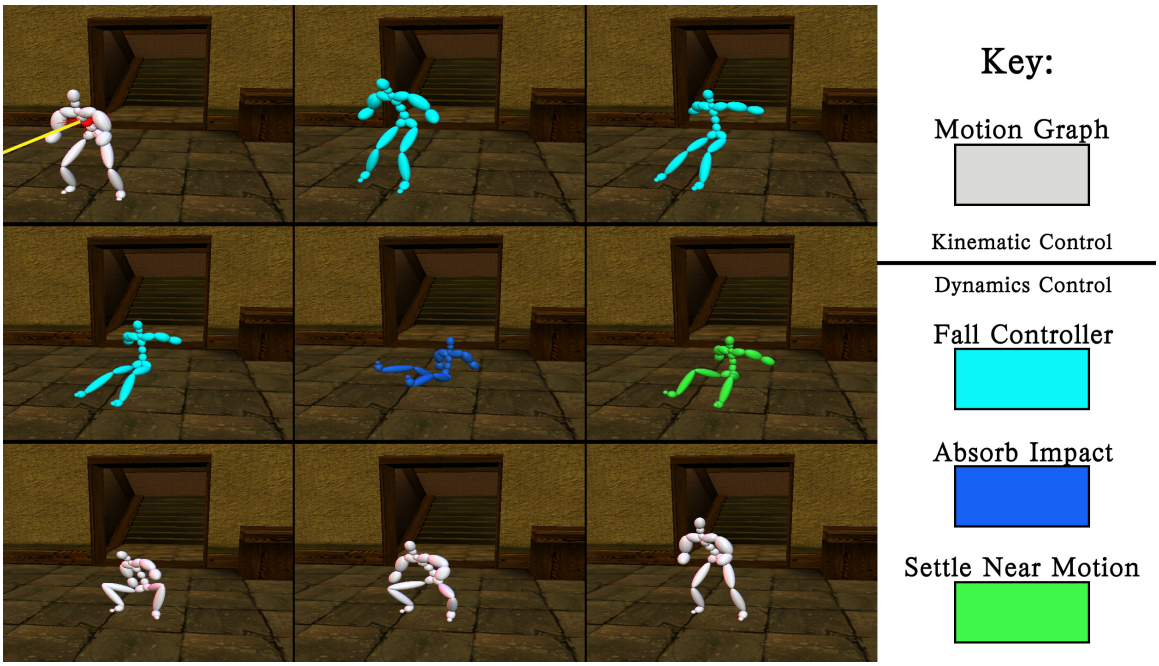


Figure 6.3: A sequence demonstrating hybrid control of a virtual character. An idle behavior is driven by the motion graph, followed by a transition to a simulated fall caused by the user-supplied force (indicated by the yellow vector in the first frame). The settle controller achieves a body pose near motion data that allows the character to return to a balanced posture. The key on the right indicates the meaning of the character's color.

Figure 6.4: Hybrid control of the character allowing a transition from a sneaking behavior to a simulated fall, followed by a recovery to the original sneaking behavior. The color of the character is as in Figure 6.3.

on how far away the closest motion data may be from the simulated pose at the time a transition is requested. For example, interactions with the environment may cause the fall controller to leave the character upright, but leaning against an obstacle, rather than the expected prone or supine position. The closest pose in the motion database might then be sufficiently distant that the movement generated by the settle controller, while still physically correct, could appear unnatural. For these cases, it might be best to augment the settle controller with a corrective balancing reaction to reach nearby motion data. A planning algorithm may also be required to reason more intelligently about how and when the simulation should return to motion graph control (see Chapter 7 for more discussion on this topic).

6.2 Discussion

Our successful application of a hybrid motion synthesis system is a step towards the development of autonomous virtual characters. Researchers can employ a bottom-up approach by augmenting the behavioral capabilities of each synthesis method incrementally. Our assessment indicates that data-driven and dynamics-based synthesis techniques have complementary qualities within a hybrid system, greatly extending the capabilities of either technique alone.

Chapter 7

Conclusion

7.1 Summary

The progress of developing fully autonomous virtual characters has been hindered by the lack of a single technique that can accurately capture all the behaviors and reactions such a character would need to perform. Until further advances are made by researchers in artificial intelligence, robotics, animation, and biomechanics, a hybrid system is a viable option, offering the best qualities of each motion synthesis technique and handling a wider range of situations. This thesis has explored constructing such a hybrid system, combining data-driven control with motion graphs and physically simulated control with proportional-derivative controllers. Our system attempts to determine when each technique is most appropriate and provide the facilities to transition between them as the character's goals and interaction with the environment evolve. We utilize a fast approximate search technique based on nearest neighbor searching to build a correspondence between existing motion data and poses generated by the simulation. We use the results of this search to guide the simulated behaviors into poses close to existing motion data, allowing a smooth transition to a data-driven technique. To our knowledge, this approach is a novel use of physical controllers, avoiding many common blending artifacts by moving towards motion data in a

physically realistic way.

A database of motion capture data containing several behaviors and their natural transitions are preprocessed into a motion graph and used to drive the character during data-driven control. For physically simulated control, we attempt to capture the natural reactive behavior of humans in a biomechanically inspired way. We contribute a falling controller capable of generating protective behavior for a wide variety of situations when external forces cause the character to lose balance. The physical controllers use finite state machines to manage the logical states of the falling behavior, using facing direction and body velocity to determine how and when the arms should protect the body to avoid injury. Extensions to the basic controller allow for extra behaviors, such as a simulated roll, when a quicker recovery is necessary.

7.2 Future Work

A number of areas for future work might extend the capabilities of our current system. In this section, we will first discuss how we might improve the current fall controller implementation by representing a larger set of reactive behaviors. Integrating additional synthesis techniques into the hybrid system, and perhaps applying them simultaneously to a single character, is another area with great potential for future work. Finally, AI planning techniques might help us choose the best synthesis method given the goals of the character and the current surroundings.

7.2.1 Improving the Fall Controller

While our fall controller generates protective behavior in a biomechanically inspired way, we assumed that any significant external forces would always generate a full loss of balance. We made no attempt to use the ankle or hip strategies commonly used by humans to maintain balance under small perturbations. Real humans often take corrective steps or

use nearby objects in the environment as hand holds in a surprisingly sophisticated way. Adding these fall prevention strategies to our controller would allow it to adapt to a much broader range of situations where the human is able to maintain balance through reflexive responses.

7.2.2 Extending Use of Synthesis Techniques

We chose to investigate two popular and effective techniques for synthesizing human motion to integrate into our hybrid architecture. Adding additional techniques to the current two would increase the system's flexibility in adapting to the current goals of the virtual character. Creating parameterized motions by interpolating between a set of example motions [44, 52] has great potential as an additional data-driven synthesis technique. Recent work [25, 26] in this area has lead to more automatic methods of aligning example motions as well as identifying logically related clips, making creation of parameterized motions more manageable. For example, a continuous space of jumps parameterized on jump distance and height could be used to allow a character to jump chasms of varying sizes to grab onto a ledge, where simulation could handle the motion resulting from the impact with the ledge. In addition, better inverse kinematics (IK) techniques have recently been explored [19] that attempt to capture more natural behavior-specific movement. IK affords control at a level that can be difficult to emulate using traditional data-driven techniques, in situations such as picking out a specific book from an arbitrary place on a book shelf. If new techniques can generate movements in a natural and convincing way, IK could be a viable addition to the available synthesis techniques of our system. We believe integrating these synthesis techniques would greatly extend the usefulness of a hybrid architecture.

The main research challenge in adding more synthesis techniques is to select the best technique given the state and goals of the character. Specialized planners could be developed to optimize the choice of techniques based on knowledge of environmental constraints, high-level goals, and an encoding of how each synthesis technique interacts with

this knowledge. Our current system is limited to simple heuristics to determine which synthesis technique to use, such as the straightforward triggering of simulation when significant forces are applied to the character. Transition feasibility between each technique could be assessed using a more general version of a supervisor controller [9], determining how the state spaces of each technique overlap. Understanding more complex situations as well as having a toolbox of techniques to use would contribute greatly towards creating a truly autonomous character.

Another interesting avenue for future work is to explore how multiple synthesis techniques might simultaneously act on a single character. Not all simulated controllers need to control every degree of freedom of the character. For example, walking motion might be generated from a motion graph while the direction of gaze might be generated procedurally based on a model of what in the environment would be likely to attract the character's attention. To ensure that the motions generated by combining synthesis techniques appear realistic, we would need an understanding of perceptual metrics and an ability to measure whether a motion is visually acceptable or not. Complementary to these metrics would be the ability to learn which degrees of freedom are important for a particular behavior. For instance, such a technique might automatically identify from example motion data that the arms are important to a waving motion and the legs are important to a kicking motion. Such information could be used to combine simpler, body localized behaviors, providing a high-level planning system with more flexibility when synthesizing the motion for an arbitrary situation.

7.3 Towards Total Simulated Autonomy

This thesis represents a step towards creating fully autonomous virtual characters. These characters should be capable of learning and executing any behavior a human may exhibit in response to the surrounding environment. Achieving this goal would require controllers for simulating as many behaviors as possible. As these control techniques are developed,

we may be able to rely less on motion data or other techniques to animate the subtle nuances of many behaviors. While this does not imply that human motion would not be used to derive control knowledge, it means a simulated autonomous character would never play back motion in a strictly kinematic sense. However, an architecture like ours lends itself to the gradual development of more simulated behaviors as it allows the use of motion data in situations that are difficult to simulate. In the future, an interdisciplinary collaboration between researchers in artificial intelligence, robotics, animation, and biomechanics will likely be necessary to fully achieve autonomous virtual characters, but hybrid use of synthesis techniques can guide the development of systems towards this goal.

Bibliography

[1] O. Arikan and D. A. Forsyth. Interactive motion generation from examples. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 483–490. ACM Press, 2002.

[2] O. Arikan, D. A. Forsyth, and J. F. O'Brien. Motion synthesis from annotations. *ACM Trans. Graph.*, 22(3):402–408, 2003.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[4] M. Brand and A. Hertzmann. Style machines. In *Proceedings of Graphics Interface*, pages 183–192. ACM Press, 2000.

[5] A. Bruderlin and L. Williams. Motion signal processing. *Computer Graphics*, 29(Annual Conference Series):97–104, 1995.

[6] R. Cham and M.S. Redfern. Lower extremity corrective reactions to slip events. *J Biomech*, 34:1439–45, 2001.

[7] M. C. Do, C. Schneider, and R. K. Y. Chong. Factors influencing the quick onset of stepping following postural perturbation. *Journal of Biomechanics*, 34:1439–45, 1999.

[8] R. C. Dorf and R. H. Bishop. *Modern Control Systems*. Addison-Wesley Longman Publishing Co., Inc., 1994.

[9] P. Faloutsos. *Composable Controllers for Physically-based Character Animation*. PhD thesis, University of Toronto, Toronto, Canada, 2002.

[10] P. Faloutsos, M. van de Panne, and D. Terzopoulos. Composable controllers for physics-based character animation. In *Proceedings of the 28th Annual Conference on Computer graphics and Interactive Techniques*, pages 251–260. ACM Press, 2001.

[11] P. Faloutsos, M. van de Panne, and D. Terzopoulos. The virtual stuntman: dynamic characters with a repetoire of autonomous motor skills. In *Computers and Graphics*, pages 933–953, 2001.

[12] A. C. Fang and N. S. Pollard. Efficient synthesis of physically valid human motion. *ACM Trans. Graph.*, 22(3):417–426, 2003.

[13] J. H. Freidman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.

[14] M. Girard and A. A. Maciejewski. Computational modeling for the computer animation of legged figures. *ACM Comput. Graph.*, 19(3):263–270, 1985.

[15] M. Gleicher. Motion editing with spacetime constraints. In *Proceedings of the 1997 symposium on Interactive 3D graphics*. ACM Press, 1997.

[16] M. Gleicher. Retargetting motion to new characters. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 33–42. ACM Press, 1998.

[17] M. Gleicher. Comparing constraint-based motion editing methods. *Graphical models*, 63(2):107–134, 2001.

[18] M. Gleicher, H. J. Shin, L. Kovar, and A. Jepsen. Snap-together motion: assembling run-time animations. *ACM Trans. Graph.*, 22(3):702–702, 2003.

[19] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popovič. Style-based inverse kinematics. *ACM Trans. Graph.*, 23(3):522–531, 2004.

[20] D. Gustafsson and M. Lysen. Meqon game dynamics sdk. http://www.meqon.com/, 2004.

[21] J. K. Hodgins and N. S. Pollard. Adapting simulated behaviors for new characters. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 153–162. ACM Press/Addison-Wesley Publishing Co., 1997.

[22] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien. Animating human athletics. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 71–78. ACM Press, 1995.

[23] Y. Koga, K. Kondo, J. Kuffner, and J. Latombe. Planning motions with intentions. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 395–408. ACM Press, 1994.

[24] E. Kokkevis, D. Metaxas, and N. Badler. User-controlled physics-based animation for articulated figures, 1996.

[25] L. Kovar and M. Gleicher. Flexible automatic motion blending with registration curves. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 214–224. Eurographics Association, 2003.

[26] L. Kovar and M. Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph.*, 23(3):559–568, 2004.

[27] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 473–482. ACM Press, 2002.

[28] L. Kovar, J. Schreiner, and M. Gleicher. Footskate cleanup for motion capture editing. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 97–104. ACM Press, 2002.

[29] D. Lam. Tokamak game physics sdk. http://www.tokamakphysics.com/, 2004.

[30] J. Laszlo. Controlling bipedal locomotion for computer animation. Master's thesis, University of Toronto, Toronto, Canada, 1996.

[31] J. Laszlo, M. van de Panne, and E. Fiume. Interactive control for physically-based animation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 201–208. ACM Press/Addison-Wesley Publishing Co., 2000.

[32] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 491–500. ACM Press, 2002.

[33] J. Lee and S. Y. Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 39–48. ACM Press/Addison-Wesley Publishing Co., 1999.

[34] J. Lee and S. Y. Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 39–48. ACM Press/Addison-Wesley Publishing Co., 1999.

[35] Y. Li, T. Wang, and H. Shum. Motion texture: a two-level statistical model for character motion synthesis. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 465–472. ACM Press, 2002.

[36] B. E. Maki and W. E. McIlroy. The control of foot placement during compensatory stepping reactions: Does speed of response take precedence over stability. In *IEEE Transactions on Rehabilitation Engineering*, 1999.

[37] B. E. Maki, W. E. McIlroy, and G. R. Fernie. Change-in-support reactions for balance recovery. In *IEEE Engineering in Medicine and Biology Magazine*, 2003.

[38] B. Mirtich and J. F. Canny. Impulse-based simulation of rigid bodies. In *Symposium on Interactive 3D Graphics*, pages 181–188, 217, 1995.

[39] M. Mizuguchi, J. Buchanan, and T. Calvert. Data driven motion transitions for interactive games. *In Eurographics 2001 Short Presentations*, September 2001.

[40] A. Moravánszky. Novodex physics sdk. http://www.novodex.com/, 2004.

[41] D. Mount and S. Arya. Approximate nearest neighbor queries in fixed dimensions. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1993.

[42] D. Mount and S. Arya. ANN: A library for approximate nearest neighbor searching. http://www.cs.umd.edu/ mount/ann, 1997.

[43] M. Oshita and A. Makinouchi. A dynamic motion control technique for human-like articulated figures. In *Computer Graphics Forum 20*, 2001.

[44] S. I. Park, H. J. Shin, and S. Y. Shin. On-line locomotion generation based on motion blending. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 105–111. ACM Press, 2002.

[45] K. Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, 1995.

[46] R. Playter. Physics-based simulation of running using motion capture. In *Course notes for SIGGRAPH 2000*, 2000.

[47] Z. Popović and A. Witkin. Physically based motion transformation. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 11–20. ACM Press/Addison-Wesley Publishing Co., 1999.

[48] K. Pullen and C. Bregler. Motion capture assisted animation: texturing and synthesis. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 501–508. ACM Press, 2002.

[49] P. S. A. Reitsma and N. S. Pollard. Perceptual metrics for character animation: sensitivity to errors in ballistic motion. *ACM Trans. Graph.*, 22(3):537–542, 2003.

[50] H. Reynolds and S. Collins. Havok 2: Game dynamics sdk. http://www.havok.com/, 2004.

[51] S.N. Robinovitch, E. Hsiao, M. Kearny, and V. Frenk. Analysis of movement strategies during unexpected falls. In *20th Annual Meeting of the American Society of Biomechanics*, 1996.

[52] C. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–40, 1998.

[53] C. Rose, B. Guenter, B. Bodenheimer, and M. F. Cohen. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 147–154. ACM Press, 1996.

[54] C. Rose, F. Sloan, and M. Cohen. Artist-directed inverse-kinematics using radial basis function interpolation. *Computer Graphics Forum*, 20(3):239–250, 2001.

[55] A. Safonova, J. K. Hodgins, and N. S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. In *Proceedings of ACM SIGGRAPH 2004*. ACM Press, 2004.

[56] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison Wesley, 1990.

[57] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa. Video textures. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 489–498. ACM Press/Addison-Wesley Publishing Co., 2000.

[58] A. Shapiro, F. Pighin, and P. Faloutsos. Hybrid control for interactive character animation. In *Pacific Graphics*, pages 455–461, 2003.

[59] M. Sherman and D. Rosenthal. Sd/fast. http://www.sdfast.com/, 2001.

[60] H. Shin, L. Kovar, and M. Gleicher. Physical touch-up of human motions. In *Pacific Graphics*, 2003.

[61] K. Shoemake. Animating rotations with quaternion curves. In *Proceedings of SIGGRAPH 1985*, pages 245–254. ACM Press, 1985.

[62] K. Sims. Evolving virtual creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 15–22. ACM Press, 1994.

[63] R. Smith. Open dynamics engine (ODE): A rigid body dynamcs library. http://www.ode.org/, 2004.

[64] L. M. Tanco and A. Hilton. Realistic synthesis of novel human movements from a database of motion capture examples. In *Proceedings of the Workshop on Human Motion (HUMO'00)*. IEEE Computer Society, 2000.

[65] R. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing 1*, pages 146–160, 1972.

[66] M. Unuma, K. Anjyo, and R. Takeuchi. Fourier principles for emotion-based human figure animation. *Computer Graphics*, 29(Annual Conference Series):91–96, 1995.

[67] J. Wang and B. Bodenheimer. An evaluation of a cost metric for selecting transitions between motion segments. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 232–238. Eurographics Association, 2003.

[68] D. J. Wiley and J. K. Hahn. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications*, 17(6):39–45, /1997.

[69] A. Witkin and M. Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, 1988.

[70] A. Witkin and Z. Popović. Motion warping. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 105–108. ACM Press, 1995.

[71] W. L. Wooten and J. K. Hodgins. Animation of human diving. *Computer Graphics Forum*, 15(1):3–14, 1996.

[72] J. Zhao and N. I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Trans. Graph.*, 13(4):313–336, 1994.

[73] V. B. Zordan and J. K. Hodgins. Motion capture-driven simulations that hit and react. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 89–96. ACM Press, 2002.