Hybrid Artist- and Data-driven Techniques for Character Animation

Leslie Kanani Michiko Ikemoto

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-54
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-54.html

May 15, 2007

Acknowledgement

I am indebted to David Forsyth and Okan Arikan, who encouraged, supported, and helped develop the ideas presented here. I also thank Maneesh Agrawala, Carlo Séquin, James O'Brien, Ken Perlin, Greg Niemeyer, and Jonathan Shewchuk for their guidance.

Berkeley is a magical place for me, because of Okan Arikan (especially), Ashley Eden, Tamara Berg, Andrea Frome, Adam Kirk, Jaety Edwards, Tony Lobay, Deva Ramanan, Adam Bargteil, Bryan Feldman, Pushkar Joshi, Hayley Iben, Tolga Goktekin, Alex Berg, Mike Maire, and Ryan White.

When the rest of the world walks out, my family always walks in. Words cannot express my gratitude to them.

Hybrid Artist- and Data-driven Techniques for Character Animation

by

Leslie Kanani Michiko Ikemoto

B.S. (Stanford University) 2002

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

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor David Forsyth, Co-Chair
Professor Carlo Sequin, Co-Chair
Professor Greg Niemeyer
Professor Maneesh Agrawala

Spring 2007

The dissertation of Leslie Kanani Michiko Ikemoto is approved.

Co-Chair Date

Co-Chair Date

Date

Date

University of California, Berkeley

Spring 2007

Hybrid Artist- and Data-driven Techniques for Character Animation

by

Leslie Kanani Michiko Ikemoto

Abstract

Hybrid Artist- and Data-driven Techniques for Character Animation

by

Leslie Kanani Michiko Ikemoto

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor David Forsyth, Co-Chair

Professor Carlo Sequin, Co-Chair

This thesis describes methods for automating the repetitive parts of character animation using semi-supervised learning algorithms. In our framework, we observe the output the artist would like for given inputs. Using these observations as training data, we fit input-output mapping functions that can generalize the training data to novel input. The artist can provide feedback by editing the output. The system uses this feedback to refine its mapping function. This iterative process continues until the artist is satisfied.

We apply this framework to three important character animation problems. First, sliding foot plants are a common artifact resulting from almost any attempt to modify character motion. We describe an on-line method for fixing this artifact that requires no manual clean-up. Using an artist-trained oracle, we demonstrate that we can accurately annotate animation sequences with foot plant markers. We then use an off-the-shelf inverse kinematics solver to determine the position of each foot plant.

Second, to our knowledge, all motion synthesis algorithms sometimes produce character animation that looks unnatural (i.e., contains artifacts or is otherwise distinguishable as synthesized motion by a human observer). We describe methods for automatically evalu-

ating synthesized character animation. We demonstrate that we can successfully use these methods as the testing parts of hypothesize-and-test motion synthesis algorithms. Our first method uses an SVM-based classifier trained on joint position data. We discovered that it is difficult to train a reliable classifier using this feature space, because natural and unnatural looking motion can lie close together. In our follow-on work, we discovered that using features known to be perceptually important to human observers yields a classifier that is more reliable than the current state-of-the-art.

Third, artists can create compelling character animations by manipulating the details of a character's motion. This process is labor-intensive and repetitive. We show that we can make character animation more efficient yet still controllable by generalizing the edits an animator makes on short sequences of training data to other sequences. Using Gaussian process models, our method predicts the pose and dynamics of the character at each frame (i.e., time instance in the animation), then combines these estimates using probabilistic inference. Our method can be used to edit motion for an existing character, or it can be used to map motions from a control character to a very different target character. Finally, we present data from interviews with professional animators which suggest that generalizing edits can save artists significant time and work.

Professor David Forsyth
Dissertation Committee Co-Chair

Professor Carlo Sequin
Dissertation Committee Co-Chair

Contents

List of Figures

List of Tables

Acknowledgements

I am indebted to David Forsyth and Okan Arikan, who encouraged, supported, and helped develop the ideas presented here. I also thank Maneesh Agrawala, Carlo Sequin, James O'Brien, Ken Perlin, Greg Niemeyer, and Jonathan Shewchuk for their guidance.

Berkeley is a magical place for me, because of Okan Arikan (especially), Ashley Eden, Tamara Berg, Andrea Frome, Adam Kirk, Jaety Edwards, Tony Lobay, Deva Ramanan, Adam Bargteil, Bryan Feldman, Pushkar Joshi, Hayley Iben, Tolga Goktekin, Alex Berg, Mike Maire, and Ryan White.

When the rest of the world walks out, my family always walks in. Words cannot express my gratitude to them.

Chapter 1

Introduction

One of the most powerful applications of computer graphics is in showing us worlds that exist only as ideas. Such worlds may be fanciful, as in many animated cartoons. Some may be exciting, like first-person shooter video games. Others, like emergency-training simulations, serve practical (perhaps even life-saving) purposes.

Often the creators of these worlds populate them with moving characters and objects, for which there are many animation techniques. This thesis focuses on character animation.

Producing high-quality character animation is generally time-consuming and expensive. Consider the work involved in animating Templeton, the rat character in the 2005 movie version of *Charlotte's Web*. Animating him using the traditional process of *keyframing* involves specifying poses he should assume at particular times. Pose changes determine Templeton's dynamics. Keyframing is difficult because a main character like Templeton will typically have hundreds of controls, each of which can be changed many times per second, depending on the sampling rate of the animation. Therefore, creating a one-minute long animation of Templeton can easily involve working with tens of thousands of controls.

Motion capture — recoding a live actor's movements — can also produce high-quality animation. Although motion capture is usually faster and cheaper than keyframing, it still requires costly equipment and skilled technicians. Furthermore, it may not be suitable for all characters. For example, animating Templeton using motion capture requires either

recording the movements of a real rat, or recording the movements of a human performer acting like a rat. In either case, the captured data will probably need editing. While editing an animation can be easier than creating one, it can still entail working with tens of thousands of controls.

Hence, creating and editing motion can be difficult. It is also repetitive, because motion has repeating structures, like walk cycles. Adding more spring to a character's walk can require making largely identical changes to potentially hundreds of walk cycles in a large motion database.

Despite the difficulty, character animation is widely used in movies, video games, and training simulations. One can also use character animation as training and testing data for computer vision applications such as people-trackers, or for medical applications that require close examination of movement. Because the demand for character animation is high but it is difficult to produce, there has been considerable interest and research into reliable techniques for generating high-quality character motion more efficiently.

1.1 Approach

This thesis focuses on automating the repetitive parts of character animation. We observe the output the artist would like for a given input. Then, upon seeing similar input, our goal is to generate the corresponding output automatically. We cast this problem in a probabilistic inference framework, using established machine learning techniques to build input-output mapping functions.

An appealing aspect of this framework is that it is highly controllable because the artist defines the behavior of the system. In fact, an important distinction between the goals of our work and many other motion synthesis algorithms is that we do not aim to fully automate animation. Fully automated systems usually expose a limited number of controls to the animator, making these systems easy to use. However, because of the limited control, the artist has few options to change the output.

In contrast, the artist defines the behavior of our system by providing examples of what the system should do. If the artist does not like the result, the artist can provide feedback by correcting the output. The system uses the feedback to refine itself and produce better results.

A second appealing aspect is that our framework makes producing character animation more efficient. We show that we can automatically generalize the training data the artist gives us to novel inputs, reducing the work that the artist must do by hand.

1.2 Overview

We demonstrate the application of this framework to three important motion synthesis problems. First, *footskate*, where a character's foot slides on the ground when it should be planted, is a common artifact resulting from almost any attempt to modify motion data. In Chapter 3, we describe an online method for fixing footskate that requires no manual clean-up. An important part of fixing footskate is determining when the feet should be planted, at which animators seem skilled. However, it is an onerous task to label every pose of an animation with foot plant markers. We describe our method for automatically determining a per-frame labeling of foot plants using an artist-trained oracle. Our method is more accurate than baseline methods that check the height or speed of the feet. These baseline methods perform especially poorly on noisy or imperfect data, requiring manual fixing. Once trained, our oracle is robust and can be used without manual clean-up, making it suitable for large databases of motion. After the foot plants are detected, we use an off-the-shelf inverse kinematics method to maintain ground contact during each foot plant. Our system can be treated as a black box that produces natural-looking motion of the feet, making it suitable for interactive systems. We demonstrate several applications which would produce unrealistic motion without our method.

Chapters 4 and 5 describe our techniques for using an artist-trained classifier to automatically label motions as natural (i.e., indistinguishable from high-quality hand animation

or motion capture data) or unnatural. Such a classifier can be used as the testing part of a hypothesize-and-test technique for motion synthesis.

In Chapter 4, we demonstrate that we can train a classifier to label natural/unnatural motions produced by a simple motion synthesis technique. We can use this technique to expand a pre-existing library of motion clips. We evaluate the method by obtaining motion demands from an application, using the expanded library to create motions to meet those demands, and then scoring the synthesized motions. Motions synthesized using our expanded library are generally somewhat better than those synthesized using the original library. Furthermore, we show classifier errors tend to have relatively little impact in practice. Finally, we show that the expanded library can be used to synthesize motions of a group coordinated in space and time without producing motions that share frames.

Then in Chapter 5, we show that we can build an improved classifier by using physical and data-driven features of motion to which humans seem sensitive. We demonstrate that our technique is significantly more accurate than current alternatives. Using this classifier, we can build a mechanism that can quickly provide an application with a transition of user-specified duration between any two frames in a motion collection. During pre-processing, we search all possible 2-, 3-, and 4-way blends between representative samples of motion obtained using clustering. The blends are automatically evaluated, and the recipe (i.e., the representatives and the set of weighting functions) that created the best blend is cached. At run-time, we build a transition between motions by matching a future window of the source motion to a representative, matching a past window of the target motion to a representative, and then applying the blend recipe recovered from the cache to source and target motion. People seem sensitive to poor contact with the environment like sliding foot plants. We determine appropriate temporal and positional constraints for each foot plant using a novel technique, then apply an off-the-shelf inverse kinematics solver to enforce the constraints. This synthesis procedure yields good-looking transitions between distinct motions with low online cost.

Finally, in Chapter 6, we describe the application of this framework to editing character motion. We show we can make editing more efficient by generalizing the edits an animator

makes on short sequences of motion to other sequences. Our method predicts frames for the motion using Gaussian process models of kinematics and dynamics. These estimates are combined with probabilistic inference. Our method can be used to propagate edits from examples to an entire sequence for an existing character, and it can also be used to map a motion from a control character to a very different target character. The technique shows good generalization. For example, we show that an estimator, learned from a few seconds of edited example animation using our methods, generalizes well enough to yield minutes of high-quality character animation. Learning is interactive: an animator who wants to improve the output can provide small, correcting examples and the system will produce improved estimates of motion. We make this interactive learning process efficient and natural with a fast, full-body IK system with novel features. Finally, we present data from interviews with professional character animators that indicate that generalizing and propagating animator edits can save artists significant time and work.

Chapter 2

Background

There are three primary threads in the current character animation literature. First, we can create animations manually. We will call this the *artist-driven* approach. Second, using *data-driven* techniques, we can create animations by sampling a database of pre-existing animations. Third, with *procedural* approaches, we can produce animation from a generative model. In this chapter, we examine each of these three major areas of research, and discuss their advantages and disadvantages.

These three threads are often treated as separate methodologies. At the end of this chapter, we argue that there is benefit in creating systems which draw from more than one thread. These hybrid techniques can combine the strengths and mitigate the weaknesses of their parent threads.

2.1 Artist-driven approaches

Traditionally, artists create animation using a process known as *keyframing*. An animator specifies poses the character should strike at specific times. Usually, the *keyframes*, or extreme poses, are specified first, followed by the *break-down* poses. Break-down poses are spatially halfway between the keyframes. The artist sets the time each break-down pose occurs, controlling the dynamics of the movement. For example, if a break-down pose is set

three-quarters of the way in time past the first keyframe to the second, the first part of the motion will be slower than the second.

Once the animator specifies the keyframes and break-down poses, the rest of the poses must be filled in. (In some cases, each pose is a keyframe, but these cases are rare.) Filling in poses is called *in-betweening*, and is usually done by an apprentice animator for drawn animation, and by a computer for computer animation. Animation programs like Autodesk Maya interpolate the in-between poses automatically. Often the result is not what the animator wanted, so animation programs also usually offer mechanisms to edit the in-between frames.

Another popular technique is *stop-motion* animation. Animators have successfully used this technique in a number of movies, such as *Nightmare Before Christmas*. The artist poses a physical model of each of the characters, then takes a picture with a camera. The artist then poses the characters for the next frame and takes another picture. Typically, the pose changes between subsequent frames is slight because natural-looking motion is usually fairly smooth.

Hand animation produces beautiful, compelling results, mainly because it is extremely controllable. In the hands of a master animator, keyframing and stop-motion are powerful tools. However, because of the many controls, hand animation is extremely difficult and time-consuming. A main character will typically have hundreds of controls, each of which can be set 10 to 60 times per second (depending on the sampling rate of the animation). Hence, a one-second long animation of a single character can easily have tens of thousands of controls. When characters interact, these controls can couple in complex ways, making it even more difficult for an artist to achieve the envisioned animation.

Many animation systems feature helpful mechanisms for reducing the animator's workload. Commonly used devices include grouping animation controls hierarchically, visualizing and providing widgets to manipulate the trajectories of the characters' joints, sticking end-effectors such as hands and feet to objects in the environment, and facilitating direct manipulation of the character's skeleton and skin. Moving the character directly is an intu-

itive way to edit motion [21, 43]. One way to implement direct manipulation is with inverse kinematics (IK). Full-body IK is well studied. (See for example, [20, 8].)

An intuitive interface can make animation more natural. One option is to tie the user's movements directly to the character's movements [70, 15]. It is easy to control the character using this scheme. However, these systems usually require a special input device with many degrees of freedom, because the user controls many degrees of freedom of the character simultaneously. Igarashi et al. describe an alternative system that requires only a standard mouse. Their method works by having the user specify the pose and timing of the character in separate steps [28].

2.2 Data-driven approaches

One of the reasons artist-driven approaches are time-consuming is that the artist starts every animation afresh. For some animations, starting from scratch may be unnecessary. For example, after the artist animates a few walk cycles, animating more cycles is mostly redundant because they will all be very similar.

How can we avoid this redundant work? One approach is to synthesize new walk cycles by applying slight modifications to our existing walk cycles. This is the core idea behind data-driven approaches, which synthesize new animation by sampling and modifying pre-existing sequences. New frames individually resemble examples from the database, but together they form novel animations.

For example, by applying appropriate warp functions to one of our walk cycles, we can make the character turn left, turn right, slow down, and speed up. Popovic and Witkin show how to derive a smooth warp function from a set of keyframe-like constraints [80]. Motion warping belongs to the more general class of motion editing techniques, which adapt a single pre-existing clip to satisfy user-specified constraints. Editing techniques can handle constraints which are kinematic [11], physical [47, 71, 68, 55], stylistic [57], emotional [1], or a combination. Grochow et al. describe an IK system that uses a version of a Gaussian Process Latent Variable Model [39] to bias output poses toward pre-existing

captured poses [25]. Another important class of constraints comes from the environment. For example, a character's feet are usually constrained to be stationary when planted [37, 29].

Other data-driven techniques assemble pieces of multiple pre-existing sequences. These methods search for assemblies that satisfy user demands (e.g., "Run to the treasure chest and open it") and that join without noticeable seams. Motion graphs encode seamless assemblies of motion clips [36, 3, 42]. While motion graphs produce novel content, other aspects, such as the style of the motion, are fixed to be the same as the collection.

Another class of data-driven methods models both the content and style of motions. Perhaps the simplest example is motion blending [62, 35]. Motion blending yields a parameterized blend space that varies over the differences in the blend targets. Careful choice of targets produces a blend space over, for example, content, style, or both. Other work models the space of content and style as a parameterized set of Hidden Markov Models [9] or as a set of linear dynamical systems [44]. Modeling content and style is an attractive goal, but since there are usually many parameters, one must perform dimensionality reduction [9, 44] or contend with a high-dimensional solution space [45].

Assuming the animation database contains high-quality animation, the major advantage of data-driven techniques is that they generally produce high quality animation with little user effort. The user simply sets a few constraints, and the system automatically computes an animation from the database. Often, these techniques show poor generalization, in that the output animations will look similar to the animations in the database. For example, one can get a variety of cartwheels out of a database that contains examples of cartwheels, but no one has yet demonstrated a method to get cartwheels out of a database with no examples. In the next section, we discuss generative models of motion which try to overcome this difficulty. Currently however, data-driven methods tend to produce more realistic-looking motion than procedural methods.

2.2.1 Motion capture

Data-driven approaches typically require a large database of motion, and it is usually infeasible to hand animate enough sequences. A popular alternative approach — called *motion capture* — is to copy motion from a real actor. Motion capture systems work by

Figure 2.1. **Motion capture actor:** In optical motion capture systems, such as the one shown here, multiple cameras record the movement of markers attached to the actor's suit. Once the markers are triangulated, the system can reconstruct the pose of the actor and map it onto a virtual character. The red circle visible in the upper right hand corner is a light source attached to one of the cameras. The light bounces off the retroreflective markers on the actor's suit, aiding tracking by making the markers appear as bright spots against the dark fabric.

tracking markers glued to the body or to a tight-fitting suit (Figure 2.1). At every time sample, the system uses the positions of the markers to reconstruct the pose of the actor. The system then maps the pose onto a virtual character. The actor and virtual character ideally should not differ much, since mapping procedures work best when the actor and character have the same skeletal topology and similar limb proportions.

Motion capture systems are usually expensive because they require many cameras. To reconstruct the position of a marker, it must be visible by at least two cameras. Also, the distance between the cameras should be large relative to the distance from the cameras to the markers. (See Forsyth's and Ponce's discussion of stereo imaging [18].) In addition, skilled

technicians are usually required to man the equipment and post-process the data. Post-processing typically includes fitting a rigid skeleton to the marker data, filling in occluded markers, smoothing noisy marker data, solving for character pose at every time sample, and eliminating footskate.

2.3 Procedural approaches

The goal of procedural approaches is to create animation using generative, parametric models. This class of methods encompasses controllers and optimization.

A controller generates forces and torques that make the character move. The challenge is in finding forces and torques that make the character move in a convincing way. Perlin demonstrated that a designer can fit a controller by introspection and testing [54]. As Perlin demonstrated, hand-designed controllers can produce compelling animations. However, this technique is generally limited to simple controllers, which can sometimes produce unnatural looking motion. Designing a more reliable controller is an open research question because such controllers can be considerably more complex and have many parameters. Faloutsos et al. approach this problem by learning a controller automatically from motion data [16].

A second popular procedural method is to formulate motion as the optimum of an objective function. Metabolic energy is often used as a term in the function, because there is evidence in the biomechanical literature that terrestrial animals adopt gaits that minimize expended metabolic energy. (See [64] for a brief review of relevant literature.) One can approximate the consumption of metabolic energy with muscle usage, which can be approximated further as the forces and torques acting on the joints to produce motion. Optimization approaches have been used successfully to produce cartoon-like motion [79], and to generate human motion [45, 81]. For complex characters like humans which have many degrees of freedom, the global optimum can be difficult to find because the solution space is high-dimensional and bumpy [46]. Currently, controllers and optimization functions that generate realistic human motion are too complex to use in real-time applications.

Designing a good objective function is a challenging, open problem. Although many

objective functions include a metabolic energy term, biological entities do not always move in energy-optimal ways. For example, although the most energy-optimal way for a human to sit down is to fall on a chair, humans rarely do this. The biomechanics community is still actively investigating how real humans and animals control their muscles. Building an effective generative model for human and complex animal motion that can produce a wide variety of motions has proven difficult. In fact, so far, broad generalization has proven difficult for all automatic methods of motion synthesis.

2.4 Hybrid approaches

Each of the three motion synthesis methodologies we have reviewed is well suited for particular applications. Artist-driven approaches are a good match for movies. For example, Pixar produces effective character animation using skilled animators. While it is costly and time-consuming to produce movies this way, their focus is on giving the artist as much control as possible. Data-driven approaches are ideal for applications that require a particular person's motion. Electronic Arts sports games feature movements from well-known athletes, such as Tiger Woods' golf swing. Procedural methods are perfect for characters who will need to respond to unpredictable situations, like characters that interact with people.

However, many applications do not mesh well with any of these methodologies. These applications often require a combination of efficiency, control, and generality. For example, the ideal motion synthesizer for a video game like *The Sims* would be efficient enough to produce movement in real-time, and able to generalize the movement it has in memory to fit novel situations. Currently, The Sims' animation system can produce motion in real-time, but cannot generalize. For every action, the animation system uses the same animation clip. Hence, a Sim will always mount a bicycle the same way. If the mount-a-bicycle animation clip needs the Sim to start on the right side of the bike but the Sim is on the left, the Sim will move to the right side before getting on. The animation system also cannot cope with

a differently sized bike. If the designer creates a bike that has a higher seat, the designer must also create a new mount-a-bicycle animation clip.

How can we overcome this limitation? One strategy is to borrow ideas from both the data-driven and procedural camps, attempting to leverage the efficiency of data-driven techniques and the generality of procedural methods.

There is evidence that suggests that such an approach will work. Arikan et al. describe a hybrid data/procedural system that animates characters who are being pushed by an external force (such as another character) [5]. A purely data-driven approach is not practical for this application because it is infeasible to capture responses to all possible pushes. A procedural approach is also not ideal because the technique should generate realistic motion in real-time. Their hybrid technique begins by sampling a database to find motions that respond to the requested push as closely as possible. Then, using a spring-based physical model, they modify the response to exactly match to the user's push. This technique retains the efficiency and realism of a data-driven method, but generalizes to pushes not observed in the data.

Previous work shows that hybrid techniques also work well for other applications. We will discuss the different hybrids we can form, which applications they might be useful for, and what previous work has been done in this area.

2.4.1 Example/Artist hybrids

Artists can create beautiful animations by hand, but this process is time-consuming and expensive. How can we help artists be more efficient?

One idea is to record video of an actor performing the motion the animator wants to create. Then, the animator modifies the poses and timing to create the target animation. This technique is called *rotoscoping*, and has been used in many movies, including Disney's *Sleeping Beauty*. Although rotoscoping is difficult and also requires a great degree of skill, creating an animation by modifying an existing one can be easier than starting from scratch.

The artist can use the example animation to guide the kinematics and timing of the target animation.

Some applications require an even easier and more efficient animation method. For instance, animation prototype packages and programs geared towards novices should feature fast, easy mechanisms for creating animation.

For such applications, we may be able to use example motions more directly. Pullen and Bregler describe a system for making efficient keyframing systems [57]. Their system is based on the observation that artists can sparsely keyframe an animation quickly. Adding detail is the difficult and time-consuming part. Their method uses a database of example motions to fill in the details of a sparsely keyframed animation. Hence, the artist can create animations quickly at the cost of relinquishing control over the details of the animation. For some applications like prototyping packages, this trade-off is beneficial.

As these methods show, example/artist hybrid techniques can make an artist more efficient at the cost of losing some control over the output animation.

2.4.2 Artist/Procedural hybrids

Applications that require an easier and more efficient animation technique than artist-driven methods can provide may prefer artist/procedural hybrids to artist/example hybrids. One of the disadvantages of artist/example hybrids like Pullen's and Bregler's system [57] is that they will only work if the database has examples that are similar to what the artist wants.

Artist/procedural hybrids may not suffer from this problem. Liu and Popovic describe a system that uses procedural methods to fill in sparsely keyframed motions [47]. Their system system needs no examples. It optimizes for a motion that satisfies the keyframes and a small set of linear and angular momentum constraints. Because of their choice of constraints, they note that their technique works best with ballistic motions, like jumping. Creating a system that works well for any type of motion is an open, interesting research question.

Artists can also use procedural techniques to improve a keyframed animation. McCann et al. describe a technique for changing the timing of a motion sequence automatically so that the sequence obeys physical laws [49]. Such techniques may be useful for novice animators.

As with procedural approaches, one of the difficulties in building an effective artist/procedural hybrid is designing a reliable procedure.

2.4.3 Example/Procedural hybrids

The ideal motion synthesizer for sports video games would produce motion in real-time that responds realistically to the player and the environment. For instance, an animated football player should respond correctly to being tackled. Example-based motion synthesis techniques can only produce pre-recorded responses, so a football player animated with example-based techniques will have only a finite number of being-tackled motions. Procedural techniques can produce proper responses but generally not in real-time.

Producing a football player who can respond realistically to every tackle in real-time is an attractive goal. Zordan et al. [81] use a hybrid example/procedural method to animate characters responding to large external forces. Their system takes approximately one minute to generate an animation, so it is not real-time, but their technique highlights the promise of example/procedural hybrids for video games. Arikan et al.'s system can animate a character responding appropriately to arbitrary smaller forces (like shoves and pushes) in real-time [5].

One can also use examples to overcome some of the limitations of procedural techniques. In particular, it can be difficult to fit the parameters for a complex controller by hand because there may be hundreds of them. Liu et al. use data to learn the parameters for a character controller [45]. They solve for muscle and tendon parameters using motion capture data, then use optimization techniques to synthesize motion.

Another limitation of procedural techniques is that it is difficult to craft an objective function that will yield motions the animator wants. Instead of trying to design an objective function, one could try to define the space of valid motions, then optimize only within this

space. Safanova et al. describe a technique for performing optimization of human motion within a lower-dimensional subspace defined by example motion capture sequences [66]. Currently, their method can only construct subspaces that are specific to a motion type (like jumping). No one has yet discovered a method for determining the space of all human motions.

Chapter 3

Annotating Foot Plants

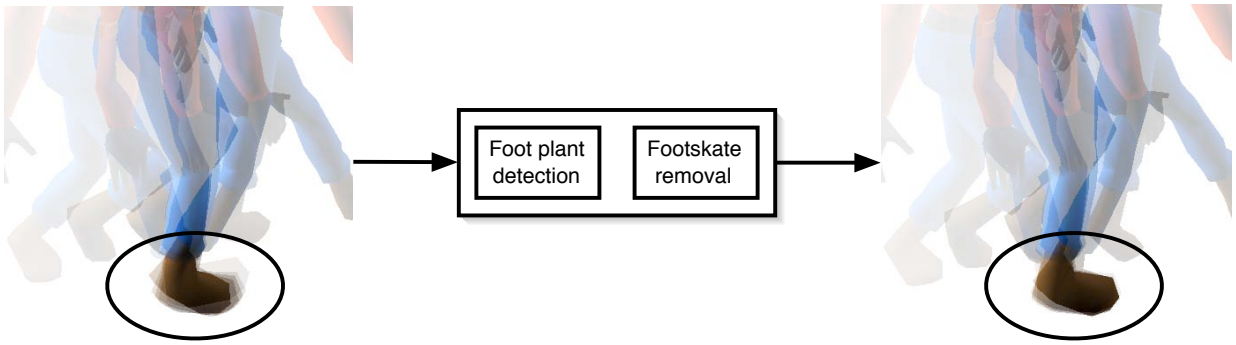

Figure 3.1. Motion editing can produce significant footskate. On the **left** is an edited motion capture sequence. We superimpose partially translucent renderings of frames spaced evenly in time. As a result, a slowly moving part of the body – like the skating foot plant in this image – shows up as a dark region with blurry outlines. In practice, people seem sensitive to footskate. We introduce a robust oracle for detecting foot plants. When coupled with an off-the-shelf footskate remover, our system behaves like a black box (**center**) that cleans up motion at interactive rates (**right**). The foot is now planted firmly, as one can see from the sharp outline around the toe. Notice that the mild blur at the heel on the right results from the way the heel and then the toes are planted.

Algorithms that modify motion are undoubtedly useful, but they sometimes introduce undesirable artifacts. One such artifact is known as *footskate*, in which a characters foot slides on the ground after the character plants it instead of remaining firmly in place. Maintaining ground contact is an important aspect of realistic motion, and in practice people seem to notice footskate and find it objectionable.

Many authors have previously noted this problem and introduced successful techniques

for fixing footskate (Section 3.1). Up to the date of this work, none of these methods were completely automatic. All required knowing *a priori* when feet should be planted. Typically, these methods used the height and speed of the foot to mark the frames, but none of these tests works reliably (Figure 3.2). One reason they do not work all the time is that motion capture data is usually noisy. In addition, a character's skeleton is not an accurate representation of a human skeleton. Because these tests do not work reliably, a user needs to check all frames by hand.

Clearly, checking frames by hand is not feasible for large datasets, which may contain many hundreds of thousands of frames. We present an automatic method for foot plant detection that is scalable and efficient (Section 3.2). By labeling a small set of frames, a user trains a classifier to detect when the foot should be planted. The classifier then automatically labels the remainder of the frames. Training time is short (our oracle required less than 3 minutes of labeled examples), and the classifier is accurate and efficient. We compare our results to results obtained using the simple tests commonly used to detect foot plants (Section 3.6). Once foot plants are detected, we can fix motion sequences containing footskate using an off-the-shelf real-time inverse kinematics solution. We use the technique of Kovar et al. [37], but other techniques could be used instead.

We show that the combination of our foot plant detector and an off-the-shelf footskate remover can be treated as a black box that cleans up motion in real-time (Section 3.6). As we demonstrate, this black box can produce plausible results from motion modifications that change the original motion drastically. In interactive applications, one cannot check the frames produced by an algorithm before they are displayed. Our method is robust, making it suitable for interactive systems.

After we did this work, Le Callenec and Boulic built a system which can robustly detect a stationary end-effector, even in the presence of noise [41]. Their system requires less training data than ours.

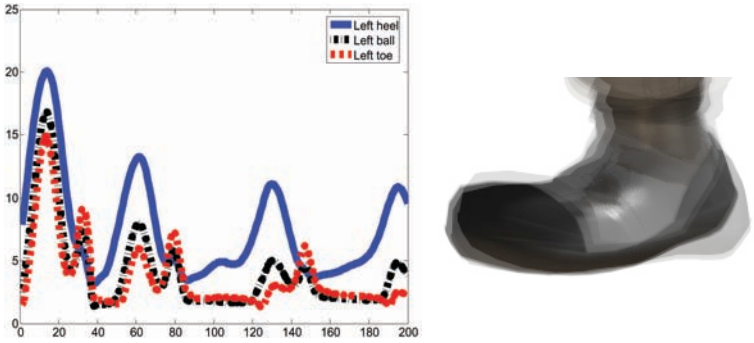

Figure 3.2. **Speed and height cannot reliably discriminate foot plants. Left:** A graph plotting the height of the joints in the left foot during 4 foot plants of walking motion in our original data set. There are actually double peaks when the foot is not planted, which may lead to mislabeling. **Right:** Despite what one might expect, foot plants are not perfectly stationary even in good motion capture data. For example, we show a left foot plant from our motion capture database. As in Figure 3.1, we overlay partially translucent renderings of frames spaced evenly in time. The foot is not stationary, so it appears with a blurry outline. If the foot was planted firmly, it would appear sharper (like the right-hand image in Figure 3.1).

3.1 Previous work

Synthesized motion must often meet constraints that define how it should look. The problem of fixing footskate can be viewed as the problem of computing positional and temporal constraints such that the feet stay planted when they should be.

Much previous work on footskate clean-up successfully attacks the problem of finding and enforcing positional constraints. Positional constraints can come from the original data, as in [8, 70]. Commercial motion processing packages often have the user set a parameter that specifies how well the end effectors should track the original data. Other papers describe techniques for computing positional constraints. Kovar et al. solves for the target position that results in minimum error over the window of frames in the foot plant [37].

Once positional constraints are determined, they can be enforced using a variant of inverse kinematics. Kovar et al. adjusts the joint angles in the legs, the root position, and the lengths of bones to satisfy foot plant constraints at each frame [37]. Lee and Shin use analytical IK solutions to reduce the number of parameters in their numerical IK computation [43]. Applying IK separately at every frame (as in [8, 37]) may produce

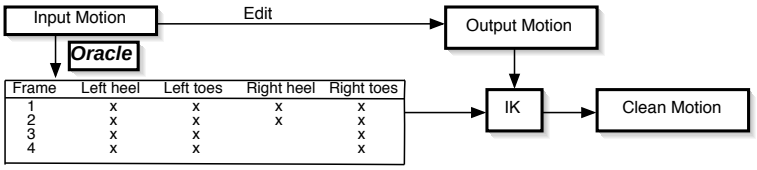

Figure 3.3. **Overview.** At run-time, an application edits an input motion in the motion database (e.g., using motion blending or warping). After editing, the output motion may contain footskate. To produce clean motion, an inverse kinematics solver needs to know in which frames the feet should be planted. Using such a labeling, the IK solver can modify the output motion to produce clean motion. This chapter describes a method for obtaining such a labeling automatically. During pre-processing, an oracle labels each frame as to whether the left heel, left toes, right heel, or right toes are planted. This labeling can be stored efficiently (requiring only 4 bits per frame). At run-time, we simply look up the labels.

visual discontinuities in the motion. Kovar et al. proposes blending the IK adjustments into surrounding frames. Other techniques optimize for the smoothest motion that meets all of the constraints [43, 21, 22].

Fewer techniques have addressed finding the proper temporal constraints for fixing footskate. A commonly used technique is to check the height of the foot, where it is assumed a foot plant occurs when the foot is close to the ground. However, this technique is easily fooled. For example, the test will probably return false results if the character skids to a stop. Another often used method is to check the speed of the foot. This technique is also unreliable. When motion capture data is taken, markers are placed on top of the feet, not on the bottom. Therefore, the markers usually have some speed even during foot plants. Furthermore, marker data is noisy, and skeletal fitting introduces further error. Therefore, joint speed is not a reliable indicator.

Liu and Popovic detect frames in which the feet are stationary [47]. Bindiganavale and Badler detect zero-crossings in acceleration space of end effectors [7]. Like the height and speed tests discussed previously, both methods work well for non-noisy data. However, motion capture data is usually noisy, so the results from both of these tests can be unreliable on motion capture data.

3.2 Chapter overview

Given a frame of motion, we would like an oracle to detect automatically if the feet are planted. Our system detects two types of foot plants — heel plants and toe plants. (However, it is easy to have the system detect other types of foot plants as well.)

We can formulate this as a classification problem. Given a feature vector describing the motion of the feet over a short period of time about a frame (Figure 3.4), we seek a labeling of the frame.

We describe the feature set we use in Section 3.3. To build the oracle, we train a k nearest-neighbors classifier interactively in an active learning process. The training procedure is described in Section 3.4, and the classifier is described in Section 3.5.

We use our foot plant oracle to label the frames in the motion database during pre-processing. At run-time, an application may edit the motions in the database, potentially introducing footskate. To clean up an edited motion, we look up the foot plant labels on the original frame. The edited motion and the labels are then given to the inverse kinematics solver, which outputs clean motion. Figure 3.3 illustrates our system.

Each frame requires only a 4-bit label (`left heel`, `left toe`, `right heel`, and `right toe`), so the labels can be stored efficiently. In addition, the oracle is fast and accurate, so pre-processing the database can be done rapidly and reliably, or the oracle can be used on-the-fly.

3.3 Features

The goal of the oracle is to decide whether a frame of motion contains a foot plant or not. The oracle makes its decision based on the skeletal configuration at the frame and a description of the dynamics. To encode the dynamics, we use the skeletal configurations in a window of nearby frames. (We use a 21 frame window.) The oracle is a function F such

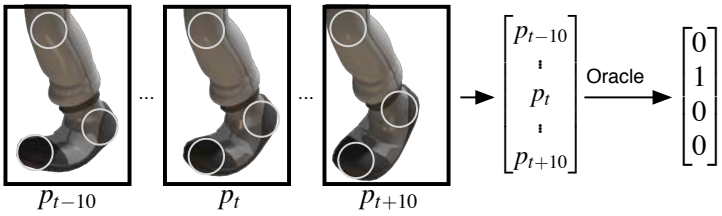

Figure 3.4. **The oracle.** In order to classify a frame, we compute the positions of the knees, ankles, and toes in a 21-frame window centered about the frame (**left**). (In the figure, p_t is the knee, ankle, and toe positions at time equals t.) We then stack these positions $(p_{t-10}, \ldots, p_t, \ldots p_{t+10})$ into a feature vector (**center**). The oracle accepts these feature vectors and outputs a 4-bit labeling corresponding to which parts of which feet are supposed to be planted in that frame (**right**). (For clarity, we show only the left foot in this figure, but positions from both feet are used in the feature vector.)

that:

$$F(p_{t-10}, \ldots, p_t, \ldots, p_{t+10}) = \begin{bmatrix} L_{left\_heel} \\ L_{left\_toe} \\ L_{right\_heel} \\ L_{right\_toe} \end{bmatrix}$$

where p_t is the position of the joints in the skeleton at time t. L_a is 1 if the oracle thinks that part of the foot is planted, and 0 otherwise. Figure 3.4 contains a schematic of the oracle.

In practice, we do not need to put all of the joints into the feature vector. The configuration of the upper body and upper parts of the legs is largely irrelevant. We use the positions of the knees, ankles, and toes, since these joints seem most useful for discriminating foot plants. The position and orientation of the figure on the plane is also irrelevant, so we transform the window of frames so that the central frame is at the origin with a zero orientation about the y-axis.

3.4 Training

We approximate the oracle F with a k nearest neighbors classifier (where k is 10). Other classifiers (such as an SVM) would probably also work well.

Figure 3.5. **Oracle training.** The oracle is approximated by a k nearest neighbors classifier which uses a small example pool. Imagine that the feature space is 2-dimensional. If we plot the feature vector for every frame in the motion database, it would look similar to the figure on the **left**. The solid circles denote frames which have already been hand-labeled, and form our example pool. Dotted circles denote the other still unlabeled frames. The system picks a frame which is maximally different from the already labeled frames (i.e., it picks the dotted circle which is most distant from the solid circles). The oracle labels the 200-frame sequence containing this frame and displays its results to the user (**center**). The user checks whether this labeling is correct. If it is not, the user corrects the labels, and the oracle puts the 200 labeled frames into the example pool (**right**). If it is correct and the user believes the oracle can correctly discriminate foot plants, the training can stop.

The classifier is trained from a set of hand-labeled examples. The system displays a motion, and the user annotates each frame with a combination of the labels `left heel plant`, `left toe plant`, `right heel plant`, and `right toe plant`.

The classifier is trained interactively (as in [4]). The system chooses a 200-frame motion sequence at random, which the user annotates. The system then trains the classifier on the annotated examples, and picks another sequence. The classifier labels this sequence and presents its results to the user. The user can check whether this labeling is correct. Errors typically show up as incorrectly localized foot plant boundaries. If the labeling is wrong, the user can correct the labels, and have the system retrain the classifier. Using on-line learning exposes the state of the classifier. The user can stop training whenever the classifier appears to discriminate correctly. The training procedure is illustrated in Figure 3.5.

We use importance sampling to select examples for classification. Once the oracle has seen an example frame annotated by the user, frames that are similar to that example are likely to be labeled correctly. However, dissimilar frames may be classified incorrectly. Therefore, one wants to choose examples which are different from those previously chosen and user annotated.

We can do this using an importance sampling function that assigns high importance to frames that are far away from the example frames in the feature space. Nearby frames get low importance. The motion database is first split into non-overlapping 200-frame sequences. The importance function assigns a value to each sequence based on the distance to the previously annotated examples. Let $d_{min}(q)$ be the minimum Euclidean distance between the feature vector of a query frame q to the example frames in the oracles training set. The importance sampling chooses the 200-frame sequence which contains the frame with the highest d_{min}.

We have found our sampling scheme effective. In particular, our example pool should contain examples from different types of motion (e.g., running, dancing, skipping). The dynamics of different types of motion are fairly distinct from each other. Therefore, we expect our sampling scheme to pick frames from each of the different types of motions in our database, and this occurs in practice. However, there are many other good schemes for selecting classification examples.

We double the size of the example pool by mirroring the example sequences and their labels, such that left foot plants become right foot plants and vice versa. The training set can include motions from multiple actors, provided that all the motions were fit onto the same skeleton.

Because it is very difficult for a user to precisely localize the frame in which a foot plant begins and the frame in which a foot plant ends, errors in the training data are unavoidable. This means that nearly identical frames occurring at the boundaries of foot plants are liable to have contradictory labels. We will address this problem later when we conduct queries (Section 3.5).

3.5 Classification

Once the oracle is trained, labeling a motion dataset is an automatic process. To get a labeling for a frame q, we examine the labels on each of the k closest samples. Let us first determine whether there is a left heel plant. Each of the k samples carries a binary label

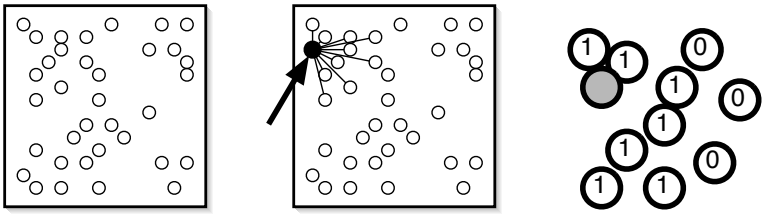

Figure 3.6. **Classifying a frame.** During training, we populate a k nearest neighbors classifier with examples of frames containing and not containing foot plants (**left**). Each example is labeled with whether it contains a left heel plant, left toe plant, right heel plant, and/or right toe plant. To classify a query frame, we locate its k nearest neighbors (**center**). Determining whether the query contains a left heel plant entails looking at the left heel plant labels on its neighbors (**right**), and averaging.

of 1 if it contains a left heel plant or a 0 if it does not. We add these labels together and divide by k to get the left heel plant label for q. This gives us a real-valued label between 0 (if the classifier is certain q does not contain a left heel plant) and 1 (if the classifier is certain q does). We do the same for the left toe, right heel, and right toe. See Figure 3.6 for a schematic diagram.

We would now like to obtain a binary labeling for q, but we cannot simply threshold the real-valued label. As discussed in Section 3.4, some of the training data of frames toward the beginnings and ends of foot plants will probably be contradictory. Therefore, the real-valued query labels tend to fluctuate rapidly toward the beginnings and ends of foot plants. The fluctuations may occur around a threshold value, so we cannot threshold the labels.

A key observation is that if a frame contains a foot plant, its neighboring frames are likely to contain foot plants as well. Therefore, if the threshold value in a frame is high indicating a foot plant, we can lower the threshold we use in the next frame.

We use two thresholds, t_{high} (0.6) and t_{low} (0.4). If a query value is greater than t_{high}, it is automatically labeled as a foot plant. If it is below t_{low}, it is labeled as not a foot plant. If a query value is between the two thresholds and the previous frame was classified as a foot plant, this frame is also classified as a foot plant. This technique successfully fixes the fluctuations at the boundaries of foot plants. It is commonly known as hysteresis [12].

Note that our method is robust to noise in the motion data. As long as the user classifies

the example set consistently, the classifier will label frames in the database consistently. Therefore, our method is sensitive to user noise, but not to data noise.

3.6 Results

Our oracle was trained on 9,410 user-labeled example frames (under 3 minutes of motion data sampled at 60 Hz), so training our classifier took little time. We double the number of examples by mirroring the example sequences and annotations (Section 3.4).

The oracle is efficient. It takes about 22.5 ms on a 3.2 GHz Pentium 4 to classify a single frame (i.e., compute the feature vector, locate the nearest neighbors, and determine the frames labels). While the nearest neighbor search dominates the running time, it is unlikely that the user will annotate many example frames. Therefore, we used a brute-force k-nearest neighbors classifier, where the query frame is compared against every other frame. However, the time required to find the nearest neighbors will increase linearly with the number of examples in the training set. If a large example pool is available, an approximate nearest-neighbors algorithm such as locality sensitive hashing [31] could speed up the search.

We compare our detection algorithm to two baseline algorithms that check the speed and height of the feet. These algorithms are commonly used to detect foot plants. During a foot plant, the foot should be stationary and touching the ground. Thus, these baseline algorithms threshold the speed and height of the feet. We fit these thresholds by trial-and-error.

We hand labeled 2000 frames of data against which to test the oracles. Each frame carries 4 labels (`left heel/toe` plant and `right heel/toe plant`), so there are a total of 8000 labels in our test set. The speed-based classifier determines 57.45% of them correctly (4596 out of 8000). The height-based classifier determines 57.00% of them correctly (4560 out of 8000). Our classifier determines 90.78% of them correctly (7262 out of 8000).

Some types of errors are more problematic than others (e.g., labels can fluctuate during

Varying the number of nearest neighbors					
1	**5**	**15**	**20**	**30**	**40**
91.00%	91.10%	91.06%	90.90%	91.03%	91.10%

Varying the window size					
3	**11**	**31**	**41**	**61**	**121**
91.03%	90.90%	90.79%	91.05%	90.76%	90.51%

Changing the joints in the feature vector	
Adding the root's position	**Removing the knee positions**
90.70%	90.80%

Table 3.1. **Varying parameters has little effect on accuracy.** This table contains accuracy rates for our classifier on the same test set used in Section 3.6. We varied parameters independently. The first row of each table corresponds to the value of the parameter being changed, and the second row reports accuracy rates. These rates vary little between each other, and are very similar to the accuracy rate of our classifier (90.78%) using 10 nearest neighbors, a 21-frame window, and using the knee, ankle, and toe positions of each leg in the feature vector. Therefore, our method does not require careful parameter tuning.

a foot plant, or a foot plant can be too short or too long). While our classifier does not have perfect accuracy, it does not make these problematic errors. Our imperfect accuracy score is because the boundaries of the foot plants differ by a few frames from the hand labeled examples. As discussed in Section 3.4, it is difficult to precisely localize the boundaries of foot plants, and the first and last 1-5 frames of a foot plant will usually appear ambiguous. Our labels differ from the user's labels only within these ambiguous regions.

Varying the parameters to our oracle does not change the accuracy rates very much (Table 3.1).

3.7 Chapter conclusion

One of the limitations of our method is that the feature set we use is tied to a particular skeleton. When motion capture data is taken, markers are placed at different positions on the body, so skeletal joint positions usually vary from dataset to dataset. It would be interesting to change the feature set to one that was skeleton independent, but we have not yet explored this point.

It is natural to try scaling the features. We have tried using our oracle on another dataset, applying a uniform scaling parameter to the feature vectors to account for the difference in the sizes of the legs. Our original data set contained motion of a graduate student.

We used our classifier — without additional training — to label the foot plants of a professional football player performing specialized football maneuvers. The two data sets were taken at different studios and fit onto different skeletons.

Our test set contained 595 hand-labeled frames. Out of the 2380 foot plant labels in the set, our classifier marked 2253 correctly, giving us 94.66% accuracy. However, even though our accuracy rate is higher than on our original dataset, our classifier actually performed worse. Our classifier made some problematic labeling errors toward the boundaries of foot plants — the labels sometimes fluctuated, and they sometimes began a few frames too early or extended a few frames too far. Errors like these are relatively unsurprising, since the relative scales of the joints between the two skeletons differ (e.g., the heel joint on the football dataset is significantly higher than on the graduate student dataset). It seems probable that non-uniform scaling would perform better.

Our method has two potential sources of latency: classification and footskate removal. The classifier requires the configuration of the legs 10 frames beyond the current frame ($\frac{1}{6}$ of a second at 60 frames per second), and the footskate removal technique we use (from [37]) requires 30 frames of look-ahead to locate a target position for the foot. We fix the first problem by doing classification as a preprocessing step. The oracle can label every frame in the database during pre-processing; at run-time, we simply retrieve the labels for the current frame. Since each frame only requires a 4-bit label, the storage cost is low. The second source of latency — footskate removal — is more problematic, and is the second limitation of our approach. However, we believe that with appropriate regression techniques we could locate a suitable target position for the foot without requiring a long lookahead.

We have described a fast and reliable method for detecting foot plants. Our technique requires only a small labeled training set and does not need careful parameter tuning. When

combined with an inverse kinematics solver, it can robustly clean-up footskate artifacts in edited motion, making it suitable for interactive applications.

Chapter 4

Scoring Motion

This chapter describes a hypothesize-and-test method that can significantly increase the size of a collection of motion observations by cutting limbs from one motion sequence and attaching them to another. Not all such transplants are successful, because correlations across the body are a significant feature of human motion. The method uses randomized search based around a set of rules to generate transplants that are (a) likely to be successful and (b) likely to enrich the existing motion collection. The resulting frames are annotated by an artist-trained classifier to tell whether they look like human motion or not.

4.1 Chapter overview

Human motion is compositional: movements can be composed across the body, so that, for example, one can walk and scratch one's head. The range of available compositions is vast (lending a "combinatorial explosion" flavor to the problem). No currently conceivable collection of data will contain examples of each possible composition. As a result, a successful motion synthesis algorithm, whether rooted in combinatorial or smoothing methods, must require some encoding of legal motion compositions (as opposed to simply looking up all motions in a set of examples).

It is natural to attack composition by cutting and pasting across the body. However,

many of the resulting motions do not look human, a result of cross-body correlation. Motion at some points on the body is often correlated with motion at other parts of the body. There are two phenomena, one active and one passive. The active phenomenon occurs in such movements as relaxed walking, where arms swing out of phase with the legs for energetic reasons. This is a gait that is chosen by the actor, and can be broken at will for example, to scratch or to reach. However, if a cut-and-paste results in a motion where these correlations are, say, out of phase, the motion will look forced and may not look human at all. Passive correlations occur as a result of generating large torques at some joints; other parts of the body may experience reaction torques. For example, a very fast reaching movement of the arm, launched while walking, may result in a passive twitch of the opposite arm. If the arm that is reaching is replaced with some other arm movement, the resulting motion will not look human because the passive twitch will be unexplained.

The main limit on the capacity of combinatorial motion synthesis and its applications is the richness of the pool of available motions. In this chapter, we demonstrate that it is possible to enhance a collection of human motions substantially by cutting and pasting across the body — a process we call transplantation (Section 4.2). Not all cut-and-paste motions look natural, but we demonstrate the we can automatically assess motions using a technique that can easily generalize to evaluate other techniques (Section 4.3). In Section 4.4, we use our enhanced pool of motions to demonstrate that with an enriched motion graph, we can synthesize multiple motions organized in time and space that do not share frames.

4.2 Enriching a motion collection by transplantation

It is currently difficult to predict precisely which components of motion are used to determine whether a motion looks human or not. However, this decision almost certainly involves observing correlations. In turn, while we cannot currently predict precisely whether a particular cut-and-paste will result in an acceptable motion, we can identify rules that are likely to yield a high percentage of successful results. This suggests using these rules to

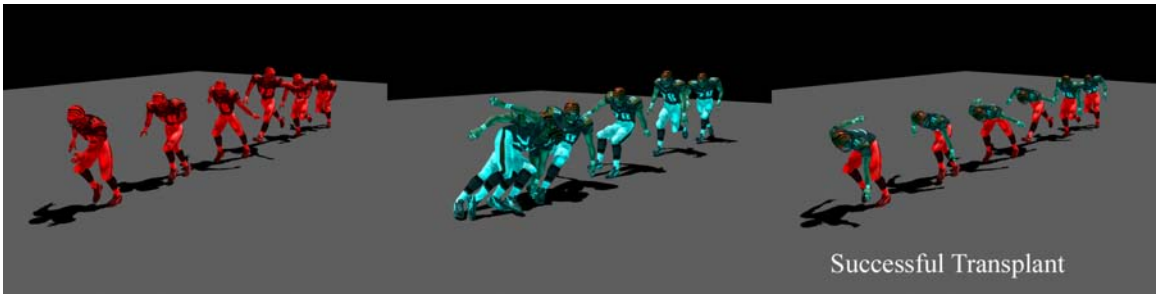

Figure 4.1. **Left** and **center** are frames from motion sequences observed using motion capture equipment. **Right** shows a sequence constructed using one of our rules; the upper body from the top sequence has been attached to the lower body from the bottom sequence. This leads to a substantial change in the qualitative structure of the motion — we have no head-butting sequences in our original collection — but retains the human character of the motion. In this figure, the frames and colours correspond in the natural way.

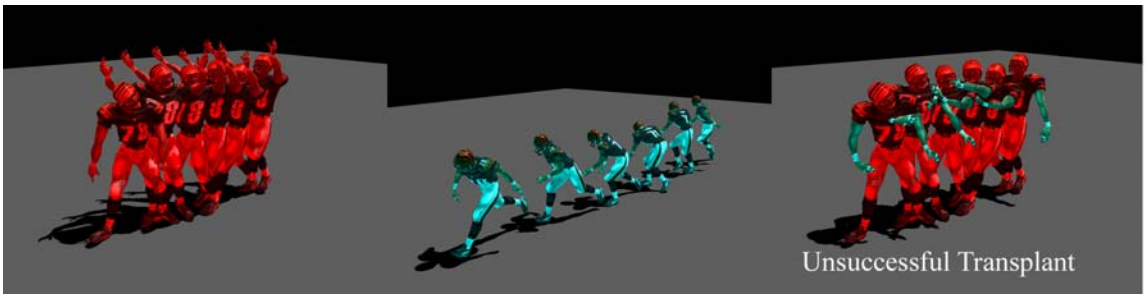

Figure 4.2. Not every transplant is successful. **Left** and **center** are frames from motion sequences observed using motion capture equipment. **Right** shows a sequence constructed using one of our rules; both arms from the top sequence has been attached to the rest of the body from the center sequence. In this case, we have produced bizarre behaviour in the arms; such frames are usually annotated `not looks human` by our classifier.

perform a randomized search through the available cut-and-pastes, with resulting motions accepted or rejected by a classifier. We have found this strategy to be successful.

4.2.1 Proposing a transplant

Our approach to transplantation is simple. We invoke one of four rules uniformly and at random, and apply that rule to two sequences (chosen randomly, using parameters determined by the rule). Each rule determines whether the pair of sequences to which it is applied is acceptable. It then prepares a set of new sequences obtained by transplanting various parts of the upperbody from the first sequence to the second, and vice versa. Our

current set of rules does not transplant any signals from below the waist, though additional rules could easily be added.

The transplants are rule-dependent. For rule 1, we transplant the upper body of the first motion with the lower body of the second motion, and vice-versa. For rules 2 and 3, we transplant:

- The right arm severed just above the shoulder from sequence 1 to the right arm of sequence 2.
- The left arm severed just above the shoulder from sequence 1 to the left arm of sequence 2.
- The shoulders and arms severed at the top torso joint from sequence 1 to sequence 2.
- The left arm, shoulder and torso severed at the top torso joint from sequence 1 to sequence 2 (preserving sequence 2's right arm and shoulder joint).
- The right arm, shoulder and torso severed at the top torso joint from sequence 1 to sequence 2 (preserving sequence 2's left arm and shoulder joint).
- The uppercarriage (torso, shoulders and arms) from sequence 1 to sequence 2.

We then transpose sequences 1 and 2, and repeat the process, resulting in twelve new sequences. For rule 4, we do not transplant. Instead, we pick one of the two motions at random, and select joint angles on the arms to perturb. We generate a sequence where we perturb the selected angles only on the right side of the body, then a sequence where we perturb only the angles on the left, and finally a sequence where we perturb both.

Our rules are designed to balance the goal of obtaining successful transplants with that of searching a wide range of possible transplants. One can reasonably believe that transplantation will work best if the source sequences are similar. However, confining transplants to similar sequences means that one is unlikely to obtain any significantly new compositions. This suggests that one should try a broad range of transplants. An attractive feature of

randomized search is that one can incorporate new rules as the process is better understood, perhaps using reinforcement learning to choose the rule applied.

Rule 1 searches for general compositions by choosing two sequences at random; we start at the first frame of each sequence and clip the longer sequence to the length of the shorter. Our observed motions tend to be in phase, meaning there is no particular advantage to be obtained by searching for a better alignment.

Rule 2 again tries to find pairs of sequences that are likely to work well, but now uses the criterion that sequences with similar footplants are likely to allow successful composition. This criterion is appropriate for finding motions where the correlation across the body is unlikely to be disturbed by transplantation. For some motions, one can regard correlation within the motion as being governed by an internal clock (e.g. arms and legs swinging in counter phase in a walk). The footplants are a rough guide to the internal clock; this suggests that pairs of motions whose footplants are sufficiently similar share a similar clock, and so may admit successful transplantation. The attraction of this criterion is that, while the footplants may be similar, the upper body may engage in quite different activities, meaning that transplantation could lead to quite significantly different motions. We implement this criterion by taking two sequences uniformly and at random, and forming for each a string representing whether and which foot is planted at each frame. The start of the aligned subsequences is found by running along time until the two sequences have a simultaneous footplant; we then continue advancing through time until the footplants are poorly aligned.

Rule 3 searches for transplants that are likely to work well, by using the distance matrix of Kovar et al. [36] to obtain a joint probability matrix. The rule then uses importance sampling to choose according to these probabilities. We write the distance matrix from that paper as $\mathbf{D}$; now zero the diagonal to obtain $\hat{\mathbf{D}}$. We then form the matrix $\mathbf{J}$ whose i, j^{th} element is $e^{-d_{ij}/(2\sigma_d^2)}$, where d_{ij} is the i, j^{th} element of $\hat{\mathbf{D}}$. The joint probability matrix is obtained by normalizing $\mathbf{J}$ to sum to one. This rule selects sequences preferentially if the sequences are "similar". We now use Dijkstra's algorithm to determine a possible time alignment between the sequences (as in [34]). If this time alignment is sufficiently similar to the identity (i.e. its slope is everywhere close to 1), we accept the pair of sequences and

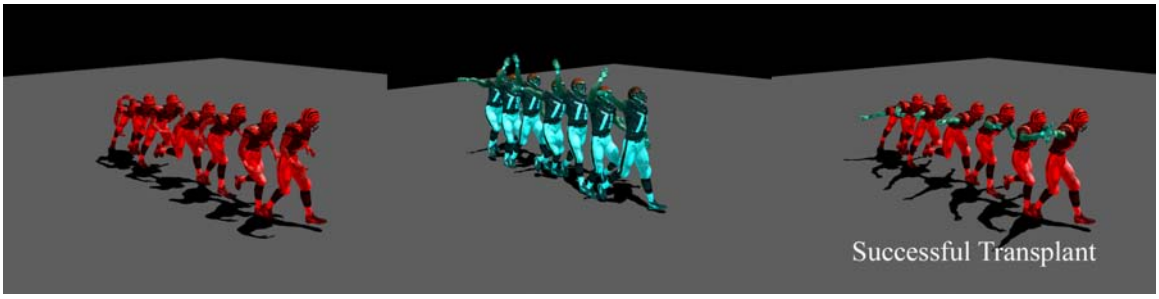

Figure 4.3. **Left** and **center** are frames from motion sequences observed using motion capture equipment. **Right** shows a sequence constructed using one of our rules; an arm and shoulder from the top sequence has been attached to the rest of the body from the center sequence. As in this case, this quite commonly leads to relatively small changes in the qualitative structure of the motion, and retains the human character of the motion. In this figure, the frames and colours correspond in the natural way.

transplant. We choose a random start point within each sequence to begin the alignment, and the endpoint is provided by the time alignment procedure.

Rule 4 first picks one of the two motions at random, then randomly selects whether to perturb the shoulder angles, the elbow angles, or both. The rule then chooses an angle offset between 20 and 90 degrees, favoring smaller angles by sampling according to a skewed distribution. Note that one can easily expand Rule 4 to include other angles (such as those on the torso), and that it could be applied to a transplant created by Rules 1, 2, or 3.

4.2.2 Determining which motions look human

None of the transplants proposed by any rule is guaranteed to look human. We must check each frame to determine whether it does. It is infeasible to check large quantities of motion by hand. In addition, no algorithm currently exists to check automatically. Our solution is to regard `looks human` as yet another annotation within the Arikan et al. framework [4]. As in that framework, we use a support vector machine (SVM) to determine whether a motion appears human or not. Using a classifier to distinguish human-looking motion is an appealing option because a classifier generalizes its training labeling without concerning itself with the underlying semantics. It is also a feasible solution because in practice motions that do not look human typically fail to do so as a result of either fast

twitches uncorrelated with the rest of the motion (Figure 4.2), or self-interpenetration. These are both phenomena relatively easily picked up on by a classifier.

We used a radial basis function kernel SVM, from the public domain library libsvm [14]. (Note that the SVM is one of a small number of classifiers that tend to perform well on a wide-range of problems, but one could choose other classifiers such as logistic regression, neural nets, or a naive Bayes classifier and expect comparable results.) Every frame of our observed motions is annotated with `looks human`. We classify every frame of the transplanted motion sequences; if the frame is accepted by the classifier it is annotated with `looks human`, otherwise it is annotated with `not looks human`. The classifier uses joint positions in the frame to be classified, the 30 frames before, and the 30 frames after as a feature vector.

The classifier is trained using two pools of examples. First, every frame of each observed human motion we possess is used as a positive example. Second, we classified a pool of 16,127 frames of transplanted motion by hand. One can create an accurate training set for the classifier because humans are very skilled at discriminating human-looking motion from other candidates. Also, as we said above, in practice motions that appear non-human often feature uncorrelated fast twitches or self-interpenetration. Both phenomena are easy for a person training the classifier to spot. One can also create a large training set quickly because the annotation flag tends to change relatively seldom for a sequence, so that it is possible to label many frames efficiently.

4.2.3 Annotating transplanted motions

In much of what follows, we use annotations attached to motions. This means that transplanted motions must be annotated. We use the annotation scheme of Arikan et al. [4], which we sketch briefly here for the reader's convenience. In particular, we follow Arikan et al. by choosing to focus on annotations that describe the qualitative properties of motion, and are using a database of 7 minutes of American football motions. The vocabulary used to annotate this database consisted of: `Run`, `Walk`, `Wave`, `Jump`, `Turn Left`, `Turn`

`Right`, `Catch`, `Reach`, `Carry`, `Backwards`, `Crouch`, `Stand`, and `Pick up`. Any combination of annotations is allowed, though some combinations may not be used in practice. Motions are then annotated using an active learning method based around an SVM to set each flag (i.e. `Run` vs. `Not Run`) independently. The classifier uses the joint positions for one second of motion centered at the frame being classified as a feature vector. The out of margin cost for the SVM is kept high to force a good fit within the capabilities of the basis function approximation.

We obtained this annotated collection from Arikan et al. [4]. The question now is to annotate transplanted motions with minimal extra effort. In practice, we have found it sufficient use nearest neighbours. We take a 17 frame sequence centered around the transplanted frame to be annotated, and find the nearest such sequence in the original annotated motions by blank search. The search cost is relatively insignificant, and is an off-line cost. The annotation sequence is then smoothed by one pass of opening and one pass of closing (as in [17]). As figure 4.1 suggests, our transplantation procedure may result in motions that are best controlled with an expanded annotation vocabulary (the figure might well be a `head-butt`). We have not explored this point in detail.

4.2.4 Transplantation results

We use a test set of transplants to establish the classifiers total error rate. The annotation bit supplied by the classifier tends to change seldom on test sequences, too. This means that it is difficult to check large numbers of individual frames, and our estimates of total error rate are rough. We arrive at these estimates by looking at short runs of motion, and count errors in terms of these runs. If the whole motion looks human to the observer, but the classifier marks only a quarter of the frames as `human`, we count a quarter motion as true positive and three-quarters as false negative. This gives a rough estimate of the error rates. On 102 test motions of a total of 14,108 frames, the total error rate is approximately 13%. The false positive rate is approximately 12%.

We apply our transplant strategy in batch mode, meaning that transplanted sequences

are not available for retransplantation. We have not attempted every possible application of the rules, but can estimate how many successful transplants are available because the rules are applied uniformly at random. From 118 observed motions, we applied the rules a total of 106 times, obtaining 234 motions and a total of 27,491 frames. It is not possible to classify the first and last 30 frames of each motion (because features are incomplete), and so we have 13,451 annotatable frames, of which the classifier marked 10,226 frames as human.

Rule 1 is applied 29 times. Of a total of 6903 possibilities, all applications find pairs that could be transplanted, yielding 3904 new frames. Approximately 35% of the resulting frames are (a) annotatable and (b) annotated as `looks human`. This suggests that rule 1 could yield approximately a quarter million good frames annotated as `looks human` if applied exhaustively. Rule 2 is applied 41 times. Of a total of 6903 possibilities, 19 applications found pairs of sequences from which motions could be transplanted, yielding 12660 new frames. Approximately 44% of the resulting frames are (a) annotatable and (b) annotated as `looks human`. This suggests that rule 2 could yield approximately a million new frames annotated as `looks human`. Rule 3 is applied 36 times. Of a total of 6903 possibilities, 18 applications found pairs of sequences from which motions could be transplanted, yielding a total 15432 new frames. Approximately 28% of the resulting frames are (a) annotatable and (b) annotated as `looks human`. This suggests that rule 3 could yield approximately three-quarters of a million new frames annotated as `looks human`. Rule 4 is applied 30 times. Of a total of 118 possibilities, all applications find angle perturbations, yielding 2,841 new frames. Approximately 35% of the resulting frames are (a) annotatable and (b) annotated as `looks human`. This suggests that rule 4 could yield approximately 4000 new frames annotated as `looks human`. There is no particular reason to believe that the rules lead to sets of frames that are distinct, but it is reasonable to believe that from half-a-million to a million new, good frames are available via this route. We have no statistics on multiple applications of the transplant process (i.e. transplanting a transplant), but believe much new motion is available this way. Comprehensive application of the rules in this manner is currently computationally impractical, however.

These results suggest that transplants lead to many frames that look acceptable, at least to the classifier. Figure 4.3 shows a successful transplant (one that looked good to the classifier). In this case, the motion has changed, but not greatly. Figure 4.2 shows a transplant that has produced a poor motion that was identified as such by the classifier. Transplants can potentially produce motions quite different in character from those in the original collection of observations, as figure 4.1 indicates.

To date, we have used our technique to generate 340,596 frames of motion data accepted by the classifier as human (from a pool of 477,362 annotatable candidate frames). At 60 frames per second, this means we have generated slightly over an hour and a half of new data from approximately 7 minutes of original motion capture.

4.3 Assessing motion modification strategies

One can evaluate a motion modification strategy only in the context of motion demands. The demands should come from a relevant application. A strategy can then be evaluated by generating many demands, synthesizing a large quantity of motion sequences, and scoring them. We have implemented a system that evaluates our motion transplantation technique in this manner, and believe it is easily extensible to evaluate other synthesis techniques.

Creating a large, canonical set of motion demands is not trivial. The types of demands typically needed for useful applications are not random, so we cannot simply place annotations randomly on a timeline. Additionally, creating demands by hand is not feasible at a large scale. Instead, we use the popular video game Unreal Tournament 2004 to generate motion demands. At every one-fifth of a second, we query the game state for position, velocity, rotation, and an annotation supplied by the game of either running or falling for every player. We enrich this set of annotations by examining the velocity vector at every tick the game describes as running. We annotate the tick with `standing`, `walking`, or `running` by setting thresholds on the magnitude of the velocity vector. In this manner, we generated over 1200 annotation streams of slightly less than 300 frames apiece from less than an hour of logging 3-person games. The streams generated in this manner do not

test all possible annotations for our particular motion database of American football. We created an additional 40 streams by hand that include annotations from the full annotation set.

To evaluate a motion graph, we need a system that takes a set of demands and a motion graph, and tries to generate motion sequences that meet the demands. It should also score how well the generated motion sequence meets its demand. We use the system described by Arikan et al. [4] to perform these tasks.

For each demand, the system generates two motion sequences. It generates the first using our original motion capture database as the motion graph, and the second using our transplantation-enriched database. The system is permitted to perform 300 iterations of search to solve for each demand. Arikan et al.'s procedure [4] assigns a score to each motion sequence. Though this score is not canonical, we believe it is a reasonable proxy because it reflects how well a motion sequence meets its demand; and, in general, better scores tend to indicate better motions.

We expect to see the enriched motion graph produce sequences whose scores are better than the original motion graph for some demands, and it does (Figure 4.4). We also expect that the motion sequences generated seldom use frames which are labelled `not human`. They rarely do (of 28,800 frames synthesized to meet our 100 demands, only 980 — or 3.4% — are annotated `not human`). Lastly, we verify that the motion sequences use frames that were newly generated, and do not solely contain frames from the original motion capture collection.

4.3.1 Examples and results

To assess the impact of mislabelled frames, we need to check all the frames by hand. We subsampled 14,108 frames from our transplantation-enriched database, annotated them with the classifier, and then had a human check the annotations. Of 7988 annotatable frames, the classifier labelled 5044 of them as `human`. For this experiment, we test our original motion capture graph against one that contains all of the original motion capture

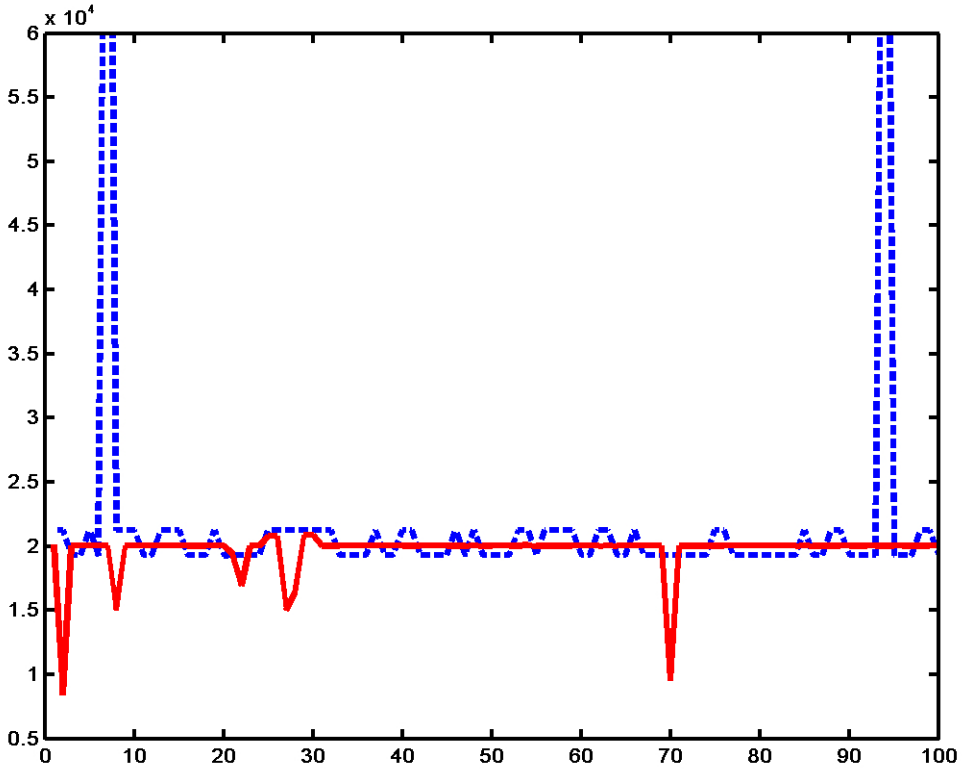

Figure 4.4. This plot shows the scores for 100 sequences generated from for the original motion graph (in **dashed blue**) and the transplantation-enriched motion graph (in **solid red**). This figure indicates that transplantation can create significantly better motions for certain demands, and avoid catastrophes for others.

data plus the subsampled version of our enriched graph. This gives us a total of 36,753 frames. The original graph contains 22,645 frames. Our set of motion demands consists of 40 annotation streams created by hand, and 60 gathered from Unreal Tournament 2004. For each graph, we generate 100 motion sequences from these demands. Then we compare the scores for the motion sequences. A lower score indicates that the sequence met its demand better than a sequence with a higher score.

Is the transplantation-enriched motion graph better than the original? As Figure 4.4 shows, the enriched graph generates significantly improved motion sequences for some demands. This is because adding even a small number of frames from transplantation makes the graph easier to search. Transplantation tends to locally explore the space around existing examples, creating frames that are similar to original frames. Thus, the search used

in Arikan et al.'s system [4] can choose between several similar frames, allowing small refinements that can increase continuity between motion blocks. The figure also illustrates that the enriched graph avoids catastrophic synthesis results for motion demands that are difficult to meet with the original graph.

Now we assess the impact of frames that do not look human. We first examine the effect of mislabelled frames — frames that the classifier labels as human looking, but a human thinks are not. Such frames, which form about 12% of the graph, are potentially dangerous because their inclusion in a sequence can make the motion look implausible. For each sequence generated from the enriched database, we count the number of falsely labelled frames the motion contains. Only 7 out of the 28,800 frames used were false positives. Therefore, one can safely ignore their presence. Recall that frames that do not look human usually contain fast twitches that are uncorrelated with the rest of the motion, or interbody penetration. We conjecture that our success at avoiding false-positive frames may be because such frames are not well-connected to the rest of the motion graph.

Secondly, we determine how often we visit frames which the classifier labelled `not human`. 3.4% of the frames used (or 980 of the 28,800 total frames in the sequences) were classified `not human`. However, this percentage is controllable. We left the frames labeled `not human` in the database because such frames can potentially improve continuity. We demand motions in which every frame has a `looks human` annotation, and we will get such motions unless inserting frames that are not so annotated produces a significantly smoother motion. Our experience is that the tendency to produce smoother motions significantly outweighs the presense of the occasional frame — which is usually kinematically acceptable — from a problem sequence. However, the frames labeled `not human` can be stripped before synthesis if the user wishes.

Lastly, we gauge the number of times we visit frames that were not in our original motion collection. 20.1% of the frames (or 5,984 of 28,800) were generated from transplants. This figure indicates that the frames generated by transplantation were beneficial in meeting the motion demands.

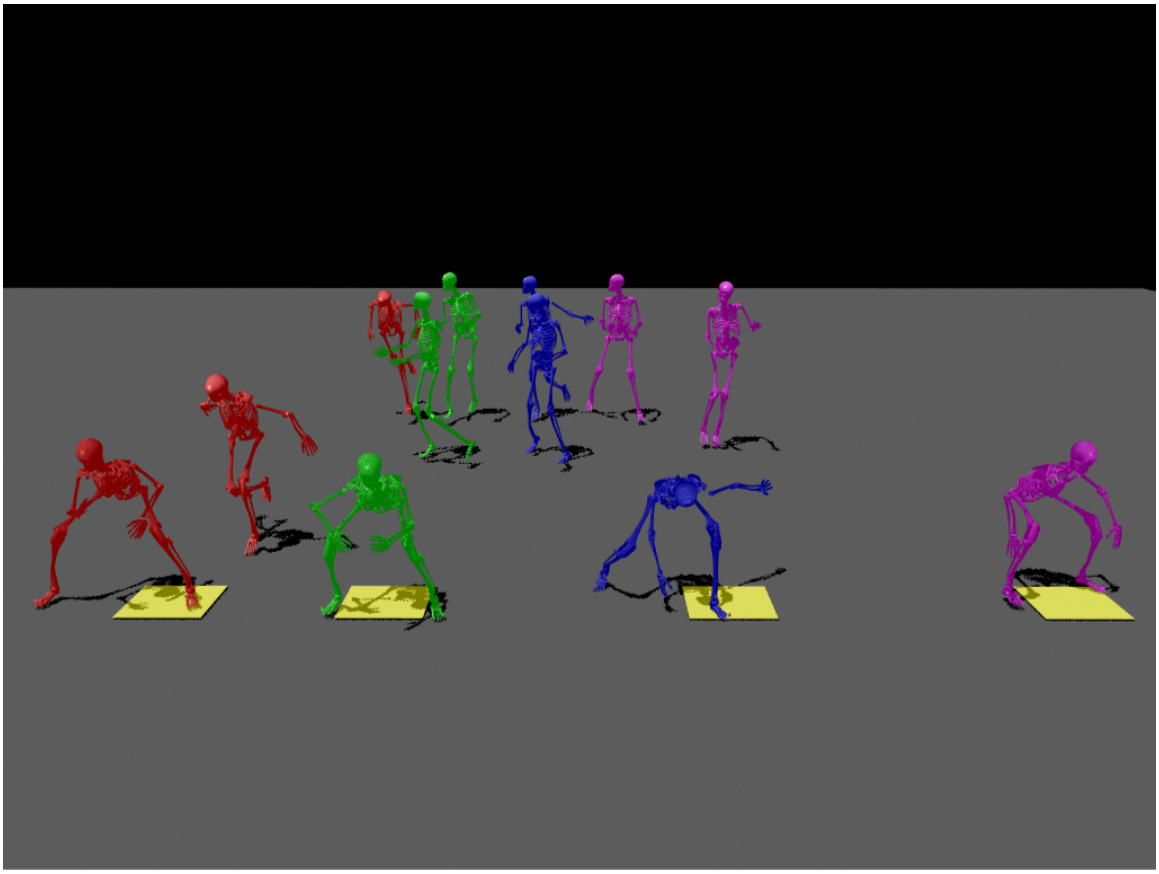

Figure 4.5. A motion synchronised in time and space. The motion demand specifies actors run from the start point, reach their specified end points and crouch. If one simply synthesizes four motions with these demands, the resulting motions are likely to share frames. By expanding, then splitting, the motion collection using our transplantation strategy, we are able to produce coordinated motions that do not share frames.

4.4 Generating multiple unique sequences from a single motion demand

One application of our technique is the generation of multiple sequences from a single motion demand that do not share frames in detail. This is an interesting problem in crowd synthesis, for example. We concentrate on getting a small group of actors to move in a fashion coordinated across space and time. This means that each should engage in the same type of motion at the same time. This creates difficulties for current motion synthesis algorithms, because, while the actors should engage in the same type of motion, they should not use exactly the same motions. Doing so produces extremely distracting effects.

In principle, Arikan et al.'s framework [4] allows this type of synthesis, but does so poorly in practice. This is because the search tends to be quite efficient at finding motions that meet the particular annotation and geometric demand: different actors tend to share the same motion example. In turn, this suggests we can finesse the difficulty with shared motions by synthesizing each actor with a distinct component of the original collection of motions. Of course, this applies only if we have sufficient motion examples to split up the collection safely.

We are able to significantly increase the size of our motion collection with transplantation. Furthermore, a natural property of our process is that it tends to generate many slightly different examples of similar motions. All this suggests that the result of transplantation should allow this type of synthesis in Arikan et al.'s framework [4]. This turns out to be successful. As a base motion collection, we use all the results of transplantation whether annotated with `looks human` or not. As in the experiment in Section 4.3, we do this because such frames can potentially improve continuity.

We have animated sequences involving four actors in this way. We used only the result of transplantation — 204 sequences in total. For each possible annotation string, we ensure that each actor has at least 30 example frames with that string, and then choose motions uniformly at random so that each actor has approximately a quarter of the total pool of frames. Motions are then synthesized with the techniques of Arikan et al. [4], resulting in sequences where actors (a) obey start and end constraints, (b) obey frame constraints, and (c) can be synchronised in time without sharing frames. Figure 4.5 shows an example of a synchronised motion.

The compositional nature of human motion means that a spanning set of motion examples will probably never be available. Instead, we must search for acceptable methods for deriving good new motions from observed motions. We have shown that good motion sequences can result from removing part of one body and attaching it to another. We used a simple randomized search to increase the size of our pool of motion examples nine times, obtaining both more examples of existing types of motion as well as novel motions in the process.

A motion synthesis technique, however, can only be evaluated in the context of relevant motion demands. We describe a methodology for evaluating our technique using demands generated by hand and from a popular video game. This methodology can also be used to evaluate other synthesis strategies.

There is a substantial advantage to possessing enough motion examples. We have demonstrated that a sufficiently large pool of examples makes it possible to synthesize motions of groups that are qualitatively synchronized in both space and time without obtaining motions that share frames; if one has too small a pool of motion examples, this strategy fails.

A great deal of work also remains on motion composition. Attractive goals include: obtaining a (near) complete set of rules; understanding what aspects of correlations are passive and what active; and determining when motions should be seen as new motions and when as composites of existing motions.

Chapter 5

Perceptual Features for Scoring

Applications commonly demand sequences that transition between types of motion (such as skipping to running) or between particular frames in a motion collection. There may be several constraints on the transition, but typically the transition must look realistic and be of a particular duration.

Motion graphs [50, 36, 42, 3] are an effective motion synthesis technique, and can be used to generate such transitions. The major difficulty with doing so is that the motions may not be responsive, because the shortest path between two frames may be too long or look bad. This problem is common in practice (Section 5.8).

In this chapter, we present a method that can generate transitions of a user-specified length from any frame in a motion collection to any other frame or type of motion. As in the previous chapter, our technique is a hypothesize-and-test method, which depends on an accurate scoring mechanism to automatically evaluate large numbers of transitions. We demonstrate that our scoring mechanism is more reliable at recognizing natural-looking transitions than current alternatives on our data sets. Although we cannot generate a natural-looking transition for every frame pair, we compute a score that indicates each transition's quality. Hence, applications know before display whether the transition will look natural or not.

5.1 Previous work

Natural looking transitions are generally difficult to synthesize. Wang and Bodenheimer show a careful choice of blend schedule and duration can significantly improve transitions between similar frames [76]. Transitions between widely differing motions — from a walk to a stand, for example — are difficult to construct in the absence of observations, because the considerations that apply (such as avoiding awkward body configurations) are more than physical. Wang and Bodenheimer describe some success with a metric for selecting transitions [75]. Spacetime methods can be used to generate dynamically plausible constraints [63]. Arikan et al. combine physical models with motion data to create transitions [5]. Transitions are important, because one wants a motion graph of **small diameter** — it should be possible to get from one frame to another with a relatively short path. One trivial and unwise way to get a motion graph with a short diameter is by inserting edges between dissimilar frames (or in the most extreme case, between all frames). Transitioning between dissimilar frames will likely result in displeasing visual artifacts, where the character appears to move unnaturally. Currently, no method can automatically yield a motion graph with a short diameter that also produces natural human motion. We demonstrate some difficulties in Figure 5.4; Gleicher et al. describe a method to engineer a motion graph with small diameter that also produces high quality motion by hand [23].

It is natural to **blend** motion clips to create transitions. This approach is known to be effective for similar motions, and a sensible choice of blending procedure can produce physically correct motion [65]. Sequences can be warped in time and space to create visually pleasing blends [34]. Wang and Bodenheimer demonstrate that the length of a linear interpolation is important [76]. One may obtain a better blend by blending more than two sequences. The methods of [11, 77, 62] generate visually pleasing multi-way blends for many blend sources, but produce unnatural-looking blends for others. Our method attacks the problem of determining which were successful. Kovar and Gleicher describe how to find multiple motions that can be blended, and how to create parameterized blend spaces within these multiple examples [35]. Other work shows how to create a parameterized blend space

for each of walking, running, and standing; their method can generate realistic transitions between these three types of motion [53, 52, 38].

Unlike previous techniques that use multi-way blending, we do not restrict our blend space to contain only motions of a homogeneous type. Instead, our method searches a very large number of blends for good results, and so we can find transitions between significantly different motions.

Our method relies on **automatic evaluation of motion**, so we require a reliable, automatic scoring technique. Work done previous to that described in this chapter demonstrates that supervised classifiers can discriminate between natural and unnatural motions produced by particular synthesis techniques [30, 5]. Ren et al. demonstrate and evaluate several unsupervised techniques that identify natural-looking motions regardless of the synthesis technique used [61]. In section 5.5, we demonstrate a novel motion evaluation method that improves on current practice.

Our blended motions may be subject to **footskate**. There are numerous footskate cleanup methods that involve some manual intervention: one marks footplants and plant locations, then uses inverse kinematics to ensure the constraints are met (e.g. [7, 70, 37, 47]). Because we must clean up a large collection of blends, we require a fully automatic method such as [29] or [41]. Our method is based on [29].

5.2 Chapter overview

Our goal is to create a compact motion synthesis mechanism that can meet transition demands in real-time with a natural-looking sequence of user-specified duration.

Interpolation is a popular technique for creating transitions. We can build a transition from frame s to t by interpolating between S (the motion clip that starts with s), T (the motion clip that ends with t), and one or more other motion clips (which we call *intermediaries*) once all of the clips have been time-aligned (Section 5.7). Some multi-way

blended transitions look natural and some do not. We need a method for deciding which intermediaries produce the most natural-looking blended transition.

We use a hypothesize-and-test procedure that automatically searches over the intermediaries to find the most natural-looking transition available, scoring motions with a novel evaluation technique (Section 5.6). We demonstrate that this evaluation technique is more accurate for our datasets than current alternatives.

There are too many motion clips to search over, so we reduce the search space by clustering them (Section 5.4). We define the *representative* of a motion clip to be the medoid of the cluster to which the clip belongs. If the clustering is effective, the motion clip's representative will be similar to the clip. Searching over only the representatives greatly reduces the number of multi-way blends we need to evaluate, but still retains much of the variation we would encounter if we conducted the entire search.

5.3 Run-time mechanism

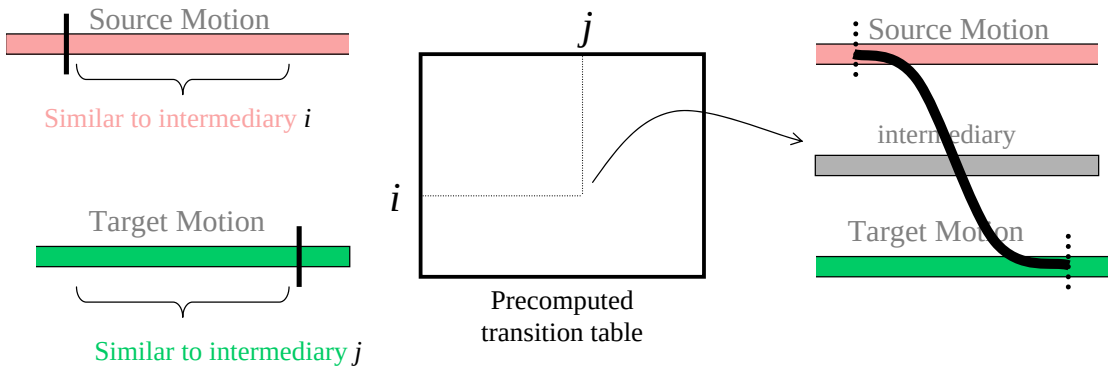

Figure 5.1. To synthesize a frame-to-frame transition online, we find the representative i that is closest to the source motion and the representative j that is closest to the target motion. We use i and j as indices into a precomputed transition table that indicates which motion clips to blend with the source and target to create the transition. We call these motion clips intermediaries.

To synthesize a frame-to-frame transition online, we find the representative that is most similar to the motion beginning at the source frame (S), and the representative that is most similar to the motion ending at the target frame (T). We then retrieve the intermediaries

required for these two representatives. We perform the blend on S, T, and the cached intermediaries after time-warping the motion clips. Figure 5.1 illustrates our mechanism.

Before display, we clean up footskate automatically. We first decide when and where the feet should be planted using the method described in Section 5.6.1. We then modify the character's degrees of freedom to meet the foot plant constraints using inverse kinematics (IK). Since IK may produce awkward poses if the target location is too far from the original position of the foot, we attempt to move the foot only if the target is less than a small distance away (we use 6-12 inches). Moving the feet may introduce discontinuities in the blended motion. We smooth these discontinuities over neighboring frames.

5.4 Clustering

We first split the motion dataset into overlapping clips with length equal to the user-specified transition duration, then use k-means clustering in spectral embedded space [56] to separate the clips by similarity. We use the cluster medoids as the representatives.

Spectral embedding is a method developed in the computer vision community for producing features that tend to cluster data points consistently with an affinity matrix. To compute the affinity between two clips, we first align their center frames in the global coordinate system, then compute the sum of squared differences of the joint positions over the motion (similar to the method used in [36]). The affinity matrix is then used to embed all clips in a low-dimensional subspace by computing the principal components of the affinity matrix, then using the ones with largest variance to project each clip into a lower-dimensional subspace. We use 10 principal components. k-means clustering can then be used to cluster the clips. Setting k equal to 50 gave good results for our dataset.

Computing an affinity between all pairs of clips is expensive because the computation time is proportional to the number of motion clips squared, and the number of clips can be large. Using a dense affinity matrix to compute an embedding is also very expensive. We use the approximate spectral embedding method of [19], which uses the Nystrom approximation to reduce the computational burden considerably. Briefly, the approximation computes

affinities between only a small random subset of motion clips and all other clips. This partial affinity matrix is used to approximate an embedding.

5.5 Scoring baseline method

Hypothesize-and-test is a natural algorithm for constructing human motions. There are now many possible methods for producing hypotheses, but no wholly reliable scoring methods. Generalization — giving an accurate score to motions very different from the training motions — is a notoriously difficult problem. In the previous chapter, we described a method that used a classifier to evaluate motions produced by a cut-and-paste method, but we found that the classifier is significantly less accurate on novel motions [30]. The classifier is trained using both positive and negative examples.

There is some advantage to not using negative examples, which can be difficult to obtain. Ren et al. fit an ensemble of generative models to positive examples; motion is scored by taking the lowest likelihood over all models to obtain a conservative score [61]. While the combined generative model gives the best behavior in practice, their combined Hidden Markov model (ensemble of HMMs) is almost as accurate. There is no information on generalization behaviour. However, if negative examples are available, we expect that models trained discriminatively are likely to perform better, because they possess more information about the location of the boundary between good and bad motion.

Training set: We picked 400 transitions generated from multi-way blends of different source-target pairs and intermediaries at random. These were annotated as natural-looking or unnatural-looking motions by hand.

Baseline: We fit an ensemble of HMMs (one for the body, one for each limb, and one for each pair of limbs) to positive motion examples. Each example is represented by a feature vector containing joint position, velocity, and acceleration data over the motion. Motions are scored with their likelihood under this model.

5.6 Scoring transitions

Several papers have observed that humans find footskate noticeable and objectionable [33, 2]. While there are many effective techniques for fixing footskate, large amounts of footskate where the foot slides a great deal usually cannot be fixed in a natural-looking way. This motivates using an error metric based on footskate to determine whether a motion looks natural (Section 5.6.1).

Unnaturalness may also stem from other parts of the body. In a physically valid motion, the *zero moment point* (the point on the ground plane at which the moment of the ground reaction forces is zero) is always within or on the boundary of the *support polygon* (the convex hull of the points at which the character is contacting the ground). People seem to find motions which do not obey this constraint objectionable, as the physical inconsistency often appears as loss of balance (though it is important to note that the zero moment point constraint holds even if the character is falling) [69]. We can use the distance between the character's zero moment point and support polygon as another error metric, which we explain further in Section 5.6.2.

We use these two error metrics to automatically evaluate the naturalness of the transitions we generate between between two frames. The best transition is the one with the lowest combined error. We obtained good results by simply summing the foot plant error and the zero moment point error.

5.6.1 Foot plant error

To determine footskate error, we first need to decide when and where foot plants should occur in the blend. Then, we can measure how much the foot moves relative to where it should be planted. If the foot does not move very much, the IK module will only need to make small changes to the motion, which will probably look natural. However, if the foot moves a great deal when it is supposed to be planted, the motion may still look objectionable if IK is used because the model will make large changes to the motion. Hence, we can use

the extent to which the foot moves as an approximate error metric for the naturalness of the motion.

Determining foot plants: We wish to identify when and where foot plants should occur in the blended motion. One solution is to identify the frames in the blended motion where one or both of the feet are close to the ground and roughly stationary. Kovar and Gleicher used such a scheme to compute input constraints for their IK algorithm (though they note that this scheme required some manual clean-up to get good results) [33]. While this technique can be successful for fixing footskate, it may not work well for determining foot plant error because the error estimated will always be small.

An observation we make is that since the blended motion is a combination of known blend sources $\mathcal{B}$, we assume that the blended motion's foot plants are a combination of the blend sources' foot plants.

Given this observation, a possible solution is to blend the source foot plants using the same weighting scheme we used to blend the source motions. However, this technique will yield fractional values for the foot plants. These fractional values could be thresholded to obtain binary values, but it is unclear how to set the threshold to obtain good results.

Instead, we consider the entire set of source foot plants, and choose the non-overlapping combination that is closest to the trajectories of the character's feet in the blended motion.

More specifically, we first compute the position of each footplant for each motion in $\mathcal{B}$ relative to the blended character's torso coordinate frame. We now take each right footplant in each motion in $\mathcal{B}$. Over the period of the footplant, we average the absolute value of the displacement error between the trajectory of the right foot in the blended motion, and the location of the right footplant. We put each right footplant in a priority queue. We then follow the same procedure to put the left foot plants in a second priority queue.

We now dequeue the right footplant with lowest error. We label the corresponding frames in the blend to indicate that they contain a footplant and where the footplant should be located. We then dequeue another right footplant. If the footplant does not overlap the first, we again label the corresponding frames to indicate that they contain a

footplant and where the footplant should be located. We stop once the queue is empty. We do the same for the left footplants. Figure 5.2 illustrates this algorithm.

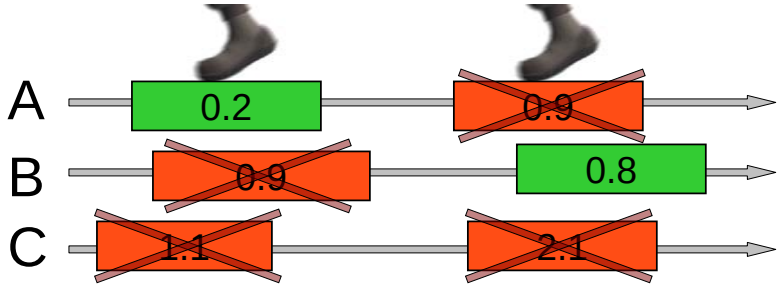

Figure 5.2. **Choosing when and where foot plants should occur in the blended motion:** Consider three blend sources A, B, and C. We would like to choose foot plants for the left foot of the blended motion. The frames containing left foot plants in each of the source motions have been labeled. In this example, all three motions contain two left foot plants. For each left foot plant, we compute the displacement error between the plant's position and the position of the blended motion's left foot at the corresponding frames. Then we choose non-overlapping left footplants in order of lowest error. We first choose the first plant in A. This means we cannot choose the first plant in B or the first plant in C. The next valid footplant with lowest error is the second plant in B. We choose this one, which invalidates the rest of the footplants.

Computing footskate error: Now that we have determined when and where footplants should occur, we can compute footskate error. We compute the displacement between the foot's actual position and the target foot plant location at every frame labeled as a foot plant. The displacement indicates how much the IK module will move the foot. IK can generally produce natural-looking motion for small displacements but not for large ones. Also, a large displacement indicates that the blended transition is very dissimilar from the blending sources, which means that it probably looks unnatural. We use the average displacement as our measure of footskate error.

5.6.2 Zero moment point error

At every point at which the character contacts the ground, the character is subject to ground reaction forces. The zero moment point (ZMP) is the point on the ground plane where the moment of these forces is zero. In a physically valid motion, the ZMP lies within

or on the boundary of the support polygon. This concept was first introduced by [73] and has been used previously in robotics and graphics (e.g., [72, 69]).

We compute the distance between the ZMP and the support polygon at every frame in a transition we wish to evaluate, then use the maximum distance as our error metric.

We calculate the ZMP at each frame using the equations from Shin, Kovar, and Gleicher [69]. These equations depend on knowing the mass of each segment of the character's body. As Shin et al. suggest, we compute the character's mass distribution using optimization with an average mass distribution (reported by [78]) as the starting point. The optimization yields a mass distribution close to the starting point that gave plausible ZMP locations for our original motion collection. We believe that this is because our motion collection is large and contains varied types of motion.

5.7 Multi-way blending

In this section, we describe how we create a multi-way blended transition of a particular duration between a source motion, a target motion, and intermediaries. Multi-way blending is not new (see for example [62, 77, 11, 35]) and has been shown to produce high-quality motion for many blend sources. We describe our time-warping and weighting techniques for completeness.

Kovar and Gleicher demonstrate that aligning motion sequences in time so that similar frames correspond is an important precursor to blending [34]. They show how to use dynamic timewarping to time-align two motions by constructing a distance matrix. Each row in the distance matrix corresponds to a frame in the source sequence, and each column corresponds to a frame in the target sequence. Each cell corresponds to a possible frame correspondence, and stores the error between the source and target frame. Paths through the matrix are possible time alignments. Dynamic programming can find a minimum cost path.

Call the set of motions to blend $\mathcal{B}$ and the i^{th} motion in the set B_i. To form a multi-way

blend, we time-align all of the motions in $\mathcal{B}$, then blend. Because we search large pools of possible blends, we may need to blend very dissimilar motions. This means that we may not be able to follow the dynamic timewarp constraints recommended in [34] and [35]. In these cases, we simply use the time-alignment that traverses the diagonal of the distance matrix. Timewarping squashes and stretches time slightly, and so the blends that we produce may not be exactly 1 second (60 frames) long. We allow time to stretch at maximum by a factor of 2, but in practice 99% of our cached transitions are fewer than 80 frames in length.

Now that we have computed a time alignment for the motions in $\mathcal{B}$, we can interpolate them. To do so, we need to devise blend weights. We create a weighting function W, where $W(t)$ gives us the weights for the frames we are interpolating. Our blending weights are Bezier functions. We chose Bezier functions because they seem to produce visually good blends, and because they extend linear blending. For example, a quadratic Bezier function is a linear interpolation of two linear interpolations. Therefore, choosing a quadratic Bezier function for a three-way blend is equivalent to interpolating B_0 and B_1, then B_1 and B_2, and then the two resulting blends. However, there are several plausible possibilities for weighting functions. It is an easy extension to our method to search over alternative weighting functions.

Once we have obtained the time alignment and the weighting functions, we can interpolate the motions. Each frame contains the 2D position of the root, the rotation of the root about the vertical axis, and the 3D position of each joint. We interpolate each joint's position using W. We cannot simply interpolate the root's position because doing so can cause the root's path to collapse on itself [34]. Instead, we interpolate the differential movement of the root of each B_i. The position of the root at the t^{th} frame of the blend is:

$$p(t-1) + \sum_{i=0}^{N} W(i,t)(p_{B_i}(t) - p_{B_i}(t-1))$$

where p is the position of the blended root, p_{B_i} is the position of B_i's root, $W(i,t)$ is the current blend weight on B_i, and N is the total number of motions in B. The position of the blended root at time 0 is taken from the first frame in the source motion.

5.8 Results

All transitions we generated were one-second long (or 60 frames in our motion collection, which was captured at 60 Hz). We can generate high-quality transitions which are normally considered difficult, such as a transition from a walk to a skip (Figure 5.3).

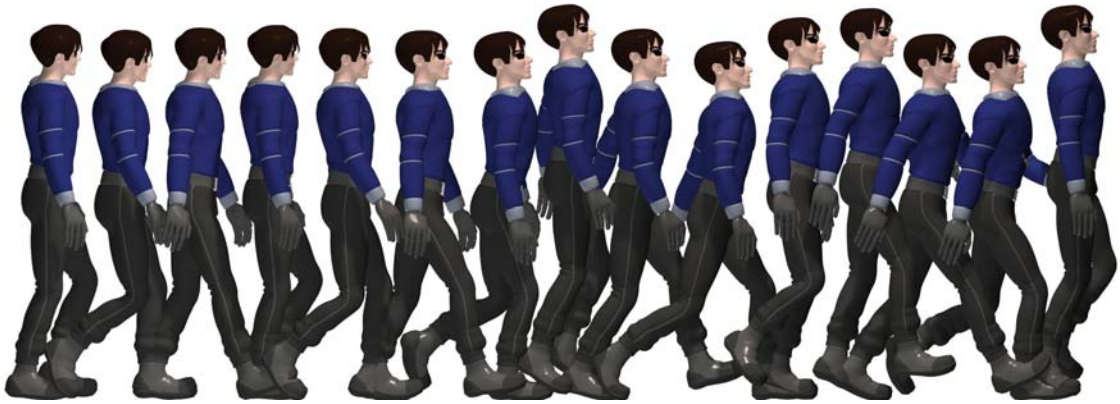

Figure 5.3. **Walk to skip transition:** This figure is a time-lapsed shot of a transition synthesized in real-time using our method. The character transitions in one second from walking to skipping in a seamless, natural way.

Our motion synthesis mechanism (Section 5.3) operates in real-time. We randomly sampled 200 source frame/target frame pairs. The average time the synthesizer required to create a one second long blend and cleanup footskate in the sequence was 0.012 seconds. Amortized over 60 frames, this cost is negligible. We also randomly sampled 200 source frame/target annotation pairs. The average time the synthesizer required to create the blend and cleanup footskate was 0.014 seconds, which means the extra time to search for a good target cluster is very small.

Our mechanism is also compact. We can compute the upperbound on the size of our transition mechanism. A maximum size entry (i.e., intermediaries and timewarp) contains a 4-way blend and the worst case timewarp, which is 120 frames long. Such an entry requires 1.9 KB. There are 50 clusters, so there are 50 members in the representative set. Thus, there are 50×50 entries in the transition mechanism, so the upperbound on the size of the matrix is less than 5 MB. Storing the differences between all one-second clips in the

database to all cluster medoids consumes 200 bytes per frame (4 bytes per float $\times$ 50 cluster medoids).

Figure 5.4 contains a scatter plot comparing the performance of motion graphs to our synthesizer. Each point represents a random source frame/target frame pair (there are 500 total). The horizontal axis is the number of frames required to make the transition in a motion graph. The vertical axis is the cost our scoring mechanism estimated for making a 60-frame transition in our synthesizer. This figure demonstrates that our synthesizer can synthesize many transitions that motion graphs cannot. We believe that the two mechanisms can nicely complement each other. Our plot is separated into quadrants. The bottom lefthand quadrant contains the frame pairs for which the motion graph can synthesize a motion that is less than 1 second long. For these demands, it is best to use a motion graph, but our synthesizer still produces good results. The upper lefthand quadrant contains the pairs for which our synthesizer cannot generate a natural looking transition, but a motion graph can easily generate one, so for these demands also, it is best to use a motion graph. The bottom righthand quadrant contains the frame pairs for which the transition synthesized by the motion graph is long, but we can produce a natural 1 second long transition. These demands are best served with our synthesizer. The upper righthand quadrant contains frame pairs for which the motion graph and our synthesizer cannot produce a good transition.

We compare our scoring mechanism to the state-of-the-art HMM baseline described in Section 5.5. Figure 5.6 demonstrates that our method outperforms the baseline. This is consistent with the general belief that natural behavior at the feet and whether the character appears in balance are important tests for the goodness of a motion.

Because our scoring mechanism is reliable, if a natural-looking blended transition exists in our search space, we are likely to find and cache it. Therefore, our aggressive hypothesize-and-test strategy can produce natural-looking transitions for many demands. For some demands, there may not be a natural-looking transition in our search space. This problem could be alleviated by conducting more search, but in general, transitioning to a very different activity over a short time period is difficult to synthesize convincingly (and may in fact be hard for a human to perform).

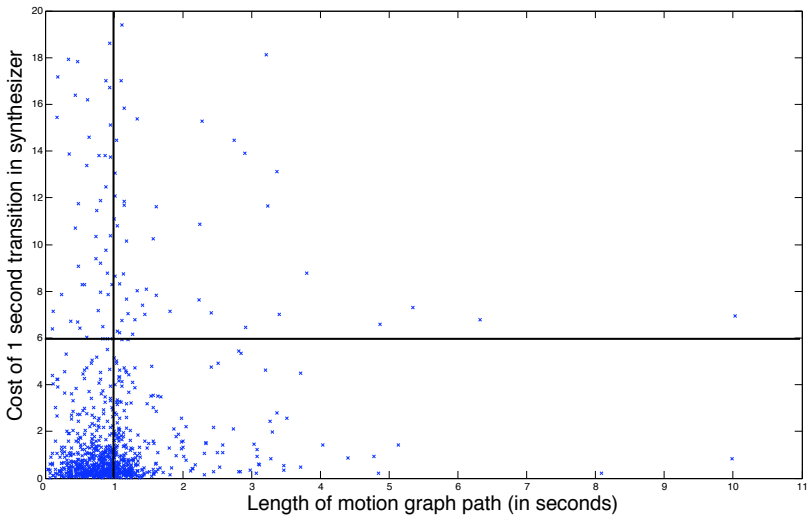

Figure 5.4. **Comparison to motion graphs.** This scatter plot compares the performance of a motion graph to our synthesizer. Each point represents a random source frame/target frame pair (there are 1000 total). The x-axis represents the length of the shortest transition between this pair of frames using motion graph, and y represents the cost of the 1-second long transition between the frames using our synthesizer. A cost under the horizontal line generally corresponds to a natural-looking motion. Note that there are many points in the bottom righthand quadrant (302 points), indicating that there are many frame pairs for which our synthesizer can generate a natural-looking, 1-second long transition while the shortest transition in a motion graph would take longer than 1 second. (Some of the motion graph paths are in fact quite long, reaching 10 seconds in one case.) Note also that there are few points in the upper righthand quadrant (57 points), which means there are few frame pairs for which both our synthesizer and a motion graph cannot produce a good transition. There are 581 points in the lower left hand quadrant (where both our method and motion graphs perform well), and 60 points in the upper left (where motion graphs outperform our method). These results indicate that our technique and motion graphs nicely complement each other, and that together there are few transition demands the combination cannot meet.

5.9 Chapter conclusions

Our work on detecting foot plants reinforced our belief that environmental contacts are an important cue as to whether or not a motion looks natural. This belief motivated us to revisit our early work on scoring motion (Chapter 4). One of the difficulties with the classification technique described there and in follow-on work [61] is that natural and unnatural looking motion may lie close together in joint data space. Using perceptually important features, such as environmental contacts, we were able to develop a more reliable

Figure 5.5. We extended our method to synthesize transitions between a source frame and a target annotation. In this figure, we show a screen shot of a real-time application that uses this extension. The user chooses one of the annotations shown on the buttons in the lower left corner, and within one second, the character will begin performing the desired activity. In this screen shot, the user has just selected skipping, and the character is beginning to skip.

classifier. However, as Figure 5.6 demonstrates, our classifier can still be improved. Hence, motion scoring remains a rich area of future work.

We believe that experimenting with other perceptually important features may be a promising approach. For example, Reitsma and Pollard describe a perceptually based metric for measuring errors in ballistic human motion that could be adapted for this application [60]. There is in fact compelling evidence that humans need little perceptual stimulation and little time to infer a great deal about human motion. In landmark work, Johansson discovered that human observers could infer the number of people, their genders, their activities, and other features about a scene from a *moving lights display* [32]. (In a moving lights display, LEDs are attached to objects and people in a dark room so that all the observer sees is moving point lights.) These experiments suggest that there may be a small number of informative features that are important to human observers. These features may be useful for motion scoring.

Our method can be easily extended to synthesize transitions between a source frame and a target annotation by simply picking a target frame with the desired annotation. We demonstrate an application that uses this extension in Figure 5.5.

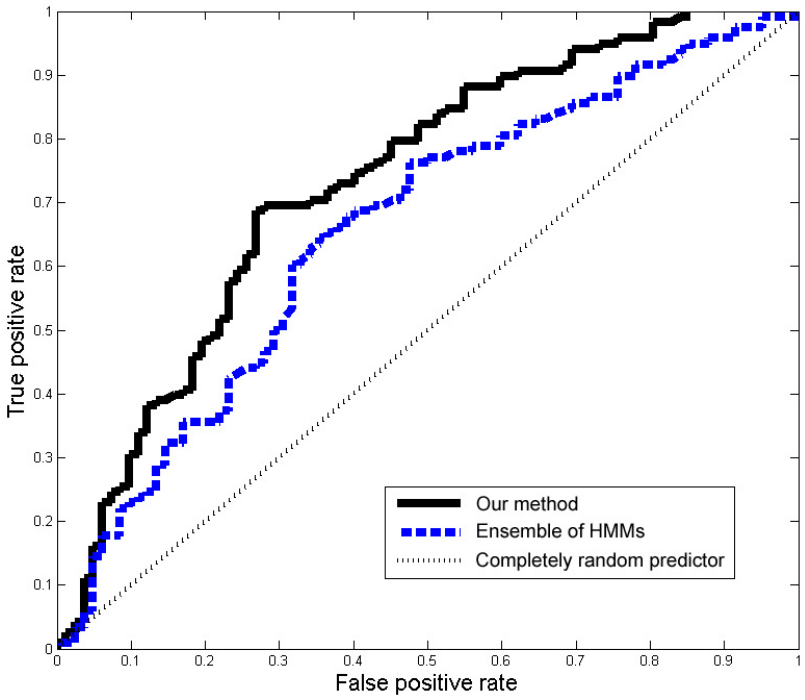

Figure 5.6. We compare the performance of our method for scoring the naturalness of motion with HMM's. The most natural way to compare is to classify good from bad motion using a threshold, then plot the receiver operating curves (ROC). A completely random classifier's ROC would lie along the black dotted line from bottom left to top right (sometimes called the no-discrimination line), and a perfect classifier's ROC would be a point in the upper left corner. The area between a classifier's ROC and the no-discrimination line is commonly used as an indicator of the performance of the classifier, where a larger area indicates a more discriminative classifier. We used a test set of 200 multi-way blended transitions chosen at random which were not in the training set. A user labeled each as *natural* (118 total in our test set) or *unnatural* (82 total). We compared each classifier's labels against the user's labels. As the figure shows, our method clearly outperforms the HMM's.

Chapter 6

Generalizing Motion Edits

Expressive character animation is compelling, because careful manipulation of subtle details of a motion can reveal a great deal about a character — from physical parameters like body weight, to movement style, to personality. Character animation can be difficult and expensive to produce, because these details must be created and manipulated with great skill and care.

This chapter describes a system that uses generalization properties of a learning system to propagate changes created in short training sequences to a full animation. Such propagation is possible because motion generally has repetitive and predictable features.

To use the tool, the artist first selects a pre-existing animation clip and edits it. Our system creates a model of the edits. Then, given another animation clip, the system automatically generalizes and propagates the artist's edits. Important properties of our system include:

- **Incremental model refinement:** The artist can change parts of the automatically generated sequence, and the system uses the changes to refine its internal model and produce a new sequence. The artist can then again make changes which the system again uses to improve its model, and thus improve the edited sequence. The ability to incrementally refine the model until it produces results the artist likes makes the system controllable.

- **Generalization to new characters:** By editing the motion to fit a new character, the artist can use our method for retargeting. Unlike previous retargeting methods, our technique is controllable. Furthermore, we show that our system works even for new characters that differ significantly in physical properties and movement styles from the original character. (In the rest of this chapter, we call the original character the source, and the new character the target.)

- **Few training examples needed:** Examples are precious, because an artist must create them by hand. Our system needs few of them. For example, after an artist edits a few cycles of walking motion, our system can automatically propagate the edits to a much longer walking sequence. In the results section (see Section 6.8), we demonstrate that using three cycles of hand-animated walking, our system can automatically generate a high-quality, five minute long walking sequence that exhibits distinctive features of the hand-animated example frames. For some sequences, the system can also generalize the edits it has learned to motions that are not in the training set (see Figure 6.6, for example).

Our method works by merging the estimates from a function that generates poses for the new character and a function that estimates the dynamics. The estimation technique we use is known as Gaussian process regression, which is a supervised learning technique that develops a probabilistic model using input-output examples.

Our method benefits greatly from an intuitive motion editing system. We present a direct-manipulation interface for editing character animation that uses a fast, full-body inverse kinematics (IK) system for interactively posing the character. Our IK algorithm is at least as fast as current solutions, and the formulation fits well with our problem.

Finally, we interviewed professional character animators about their work practices. These interviews indicate that generalizing and propagating animator edits can save an artist significant time and work.

6.1 Previous work relating to motion editing

Our method is most closely related to the work in [27]. They encode the transformation between two motions having matching content but differing style as a linear time invariant system (for a reference, see [48]). Once they fit the system, they show that they can effectively apply the transformation to a new motion faster than real-time, producing a corresponding motion in the learned style. Because their method requires one to two minutes of training data per transformation, their technique is not suitable for the problem we consider here. Also, their encoding restricts the output motion to be a linear transformation of the input motion, whereas our encoding facilitates an arbitrary transformation. We also describe an improved method for computing confidence values for the translations our model produces.

One can also encode a complex mapping between a source motion and target motion by storing examples of the relationship between them. Hsu et al. showed they can use this technique to generate compelling animations of partner dance [26]. Their method focuses primarily on motion control, and is not as well-suited for the motion editing problem we consider here, because their technique does not have a strategy for handling input motions that were not in the training set.

Our technique also draws from work in retargeting and computer puppetry, in which the motion of one character is adapted for another. The characters may, for example, differ in proportion [22, 70], in which case kinematic constraints should be identified and enforced. The characters may differ so much that an artist specifies corresponding poses [10]. Other techniques adapt the motion of a person to drive an animated character [13]. Retargeting and computer puppetry fall under the broader class of motion editing techniques.

Other character animation methods use techniques based on Gaussian processes. The scaled Gaussian process latent variable model used in [25] learns the mapping between a low-dimensional latent space and a high-dimensional character pose space. In [74], the Gaussian process dynamic model learns the mapping between a low-dimensional latent space and a

high-dimensional character dynamics space. Mukai and Kuriyama used kriging, a technique closely related to Gaussian processes, for interpolating motion [51].

Generalization — automatically producing natural-looking motion very different from training motions — is a crucial, but notoriously difficult problem. As far as we know, all systems perform unreliably in this regard. One mitigating strategy in motion transfer is to use a heuristic to detect whether an input frame is outside the space of training data, then blend the output frame with the input [27]. We use an analogous technique. Nevertheless, as we show in our results (Section 6.8), our technique shows some generalization to motions not in the training examples.

Our system incrementally builds a model of artist edits that is refined through additional examples. In a similar vein, Arikan et al. described a system for incrementally learning motion annotations [4].

6.2 Methods

We describe a general, directly controllable method for learning the edits an artist makes to an animation. Our method can be used to edit an animation, or it can be applied after traditional retargeting techniques to yield a controllable retargeting system.

First the artist selects a pre-existing animation clip, then edits it to fit the target character. The artist may edit the target's configuration in any frame, or manipulate timing between frames. The source and target clips are assumed to have the same number of frames, but the frames may appear with different timing. In Section 6.5, we describe our animation editing system, which features a direct-manipulation interface.

The artist then picks another source animation. Using the technique described in Section 6.3, the system estimates the corresponding target animation. The artist can then inspect the new target animation. If the artist does not like any part of the animation, the artist can fix the estimate. The system incorporates these fixes into the training data and produces a new estimate. Hence, the method is directly controllable because an artist can

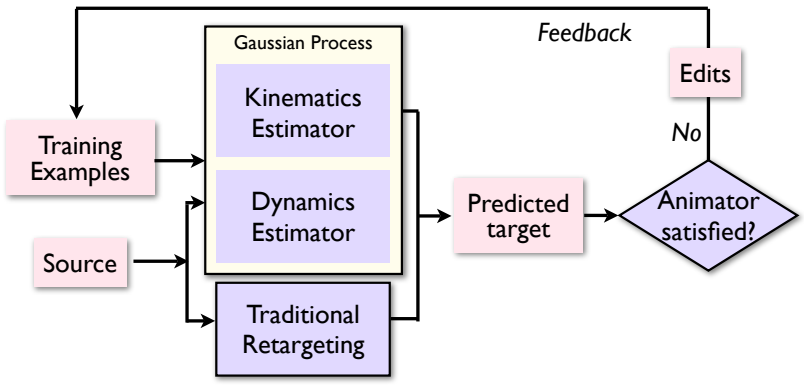

Figure 6.1. **System Overview:** After the artist picks a source animation, the system estimates the kinematics and dynamics of the target using the source and training examples. The system also produces a regularizing estimate using a traditional retargeting technique. Combining these estimates forms the predicted target, which the artist can inspect. The artist can change any parts of the predicted target, and the system learns from these fixes and incorporates them into the training set to produce an improved target animation. Until the artist is satisfied, this loop continues.

refine the system by providing feedback. After updating its internal model, the system produces a new target animation, which the artist again examines. Until the artist is satisfied, this estimate-inspect loop continues. Figure 6.1 shows a high-level schematic of the system.

Assume we have an animation for the source character, and the artist supplies the corresponding animation for the target character. We compute features from the animations to produce source features X^* and target features Y^*. Our goal is to build a function $Y = f(X)$ that estimates a target animation given a source animation. We can treat this as a regression problem. Section 6.3 discusses our regression procedure, and Section 6.4 describes the features we use.

6.3 Regression

There are several predictors we can build to produce good frames. In this work, we derive two predictors that build probability distribution functions over the target kinematics, and two that build probability distribution functions over the target dynamics. The predictors produces confidence values for each of their estimates. We solve for the frame that is closest

to all of the estimates, weighting the distances by the confidence values. Further in this section, we lay out the probabilistic justification for our approach.

The kinematic predictor defines the probability distribution function of the target kinematics at the i^{th} frame ($\tilde{Y}_i$) given the source kinematics and dynamics at the same frame (X_i), the source and target kinematics in the training set (X^*, Y^*), and parameters (θ):

$$P_k(\tilde{Y}_i|X_i, X^*, Y^*, \theta_k). \tag{6.1}$$

The dynamics predictor defines the probability distribution function of the target's second derivative at the i^{th} frame ($\ddot{\tilde{Y}}_i$), given the source's second derivative and the second derivatives of the training set:

$$P_d(\ddot{\tilde{Y}}_i|\ddot{X}_i, \ddot{X}^*, \ddot{Y}^*, \theta_d). \tag{6.2}$$

We build the kinematic and dynamic predictors using Gaussian process regression. Gaussian process regression is an elegant, nonparametric technique which yields a Gaussian probability distribution function over the space of output vectors given an input vector. The technique is non-parametric because the kernel parameters θ are computed automatically using maximum likelihood. For a good reference, see [58]. There are several publicly available implementations of Gaussian processes. We use the GPML package [59].

Because we use Gaussian processes, the probability distribution functions defined in (6.1) and (6.2) are multi-dimensional Gaussian distributions:

$$P_k(\tilde{Y}_i|X_i, X^*, Y^*, \theta_k) = c_k e^{(\tilde{Y}_i - \tilde{Y}_i^\mu)^T \Sigma_i^{-1} (\tilde{Y}_i - \tilde{Y}_i^\mu)} \tag{6.3}$$

$$P_d(\ddot{\tilde{Y}}_i|\ddot{X}_i, \ddot{X}^*, \ddot{Y}^*, \theta_d) = c_d e^{(\ddot{\tilde{Y}}_i - \ddot{\tilde{Y}}_i^\mu)^T (\Sigma_i^D)^{-1} (\ddot{\tilde{Y}}_i - \ddot{\tilde{Y}}_i^\mu)} \tag{6.4}$$

c_k and c_d are normalizing constants. $\tilde{Y}_i^\mu$ is the predicted mean pose at frame t, $\ddot{\tilde{Y}}_i^\mu$ is the predicted mean second derivative, Σ_i is the variance of the pose estimate, and Σ_i^D is the variance of the dynamics estimate. Low variance means that the Gaussian probability distribution function has a narrow peak, indicating high confidence in the prediction. Conversely, high variance indicates low confidence.

For some frames, P_k and/or P_d return an estimate with high variance. High variance means that the source lies outside the space represented by the training data or that the

training data is ambiguous, so the solution to the regression problem is ill-defined. Hsu et al. describe a heuristic for handling this problem [27], which essentially regularizes the solution to look like the source animation: when their system detects an input frame that lies outside the space of source configurations in their training set, they blend the estimated target frame with the source frame.

We adopt a similar approach. To get a regularizing estimate, we copy the joint angles of the source motion onto the target character, which is rigged the same way as the source. (Alternatively, we could use other techniques, such as one similar to Filmbox's retargeting system, in which positions of the end-effectors are determined procedurally before computing joint angles using IK.) We use the source's second derivative as a regularizing estimate for the dynamics. Call the regularizing estimates $\hat{Y}_i$ and $\ddot{\hat{Y}}_i$ respectively. We can write the probability distribution functions over these regularizing estimates as:

$$P_{rk}(\hat{Y}_i|X_i) = c_{rk}e^{(\hat{Y}_i - Y_i^R)^T(\Sigma_i^{RK})^{-1}(\hat{Y}_i - Y_i^R)} \tag{6.5}$$

$$P_{rd}(\ddot{\hat{Y}}|X_i) = c_{rd}e^{(\ddot{\hat{Y}}_i - \ddot{Y}_i^R)^T(\Sigma_i^{RD})^{-1}(\ddot{\hat{Y}}_i - \ddot{Y}_i^R)} \tag{6.6}$$

where Y_i^R is the i^{th} frame of the source sequence mapped onto the target character, $\ddot{Y}_i^R$ is the i^{th} frame of the source's second derivative, and Σ_i^{RK} and Σ_i^{RD} are the covariance matrices, which will be explained shortly. c_{rk} and c_{rd} are normalizing constants.

Now we combine the estimates to infer the target at the i^{th} frame, Y_i. We want to solve for:

$$P(Y_i|X_i, X^*, Y^*, \theta) \tag{6.7}$$

We assume that the estimates of $\tilde{Y}_i$, $\ddot{\tilde{Y}}_i$, $\hat{Y}_i$, and $\ddot{\hat{Y}}_i$ are conditionally independent conditioned on X_i. (This assumption is safe for most regression procedures, including Gaussian processes.) Now, we marginalize over the estimates to form an expression for $P(Y_i|X_i, X^*, Y^*, \theta)$:

$$\int_G P(Y_i|G)P(\tilde{Y}_i|X_i)P(\ddot{\tilde{Y}}_i|X_i)P(\hat{Y}_i|X_i)P(\ddot{\hat{Y}}_i|X_i)dG, \tag{6.8}$$

where G represents the estimates $\tilde{Y}_i$, $\ddot{\tilde{Y}}_i$, $\hat{Y}_i$, and $\ddot{\hat{Y}}_i$. (For notational clarity, we omit the training data and parameter terms in the probability distribution functions.)

Assuming an improper, uniform prior on Y_i (because we have no reason to prefer certain Y_i), $P(Y_i|\tilde{Y}_i, \ddot{\tilde{Y}}_i, \hat{Y}_i, \ddot{\hat{Y}}_i)$ is proportional to $P(\tilde{Y}_i, \ddot{\tilde{Y}}_i, \hat{Y}_i, \ddot{\hat{Y}}_i|Y_i)$. Using the Naive Bayes assumption that the estimates are conditionally independent when conditioned on Y_i, then we can model $P(\tilde{Y}_i, \ddot{\tilde{Y}}_i, \hat{Y}_i, \ddot{\hat{Y}}_i|Y_i)$ as:

$$P(\tilde{Y}_i|Y_i)P(\ddot{\tilde{Y}}_i|Y_i)P(\hat{Y}_i|Y_i)P(\ddot{\hat{Y}}_i|Y_i) \tag{6.9}$$

Model each of the probability terms in equation 6.9 with a delta function, so that Equation 6.9 is equal to $\delta(\tilde{Y}_i - Y_i)\delta(\ddot{\tilde{Y}}_i - \ddot{Y}_i)\delta(\hat{Y}_i - Y_i)\delta(\ddot{\hat{Y}}_i - \ddot{Y}_i)$, which we will call $\Delta(G)$. Now the integral in Equation 6.8 is proportional to:

$$\int_G \Delta(G)P(\tilde{Y}_i|X_i)P(\ddot{\tilde{Y}}_i|\ddot{X}_i)P(\hat{Y}_i|X_i)P(\ddot{\hat{Y}}_i|X_i)dG \tag{6.10}$$

Naturally, we would like to determine the target sequence with the highest posterior probability. The integral in Equation 6.10 turns out to be the product of the last four probability terms, so maximizing it is equivalent to the maximizing the product of the last four probability terms.

Therefore, the sequence we seek is the one that maximizes the product of the probabilities. Finding the maximum product is equivalent to finding the minimum of the negative logarithm.

The kinematic and kinematic regularizing log terms of the minimization for a single frame are:

$$(Y_i - \tilde{Y}_i)^T\Sigma_i^{-1}(Y_i - \tilde{Y}_i) + (Y_i - \hat{Y}_i)^T(\Sigma_i^{RK})^{-1}(Y - \hat{Y}_i), \tag{6.11}$$

and the dynamics and dynamics regularizing log terms are:

$$(\ddot{Y}_i - \ddot{\tilde{Y}}_i)^T(\Sigma_i^D)^{-1}(\ddot{Y}_i - \ddot{\tilde{Y}}_i) + (\ddot{Y}_i - \ddot{\hat{Y}}_i)^T(\Sigma_i^{RD})^{-1}(\ddot{Y}_i - \ddot{\hat{Y}}_i) \tag{6.12}$$

Note that we dropped the logs of the normalizing constants for clarity.

We can combine the equations and turn them into a minimization over Y alone by approximating $\ddot{Y}$ with finite differencing. We set $\ddot{Y}$ equal to the matrix product LY, where L is a linear operator that performs the finite differencing. Now the minimization over the

whole sequence in matrix form becomes:

$$\begin{aligned}\min_{Y} \quad & [(Y-\tilde{Y})^T\Sigma^{-1}(Y-\tilde{Y}) + (LY-\ddot{\tilde{Y}})^T\Sigma_D^{-1}(LY-\ddot{\tilde{Y}}) \\ + \quad & (Y-\hat{Y})^T\Sigma_{RK}^{-1}(Y-\hat{Y}) + (LY-\ddot{\hat{Y}})^T\Sigma_{RD}^{-1}(LY-\ddot{\hat{Y}})],\end{aligned}$$

where the i^{th} column of Y corresponds to Y_i, the i^{th} column of $\tilde{Y}$ corresponds to the mean pose at the i^{th} frame, and Σ^{-1} is the diagonal inverse covariance matrix obtained by stacking the Σ_i^{-1} matrices in order. Σ_D^{-1} is obtained correspondingly.

Recall that we left the covariance matrices Σ_{RK} and Σ_{RD} unspecified. How do we compute them? We would like to downweight the source when the Gaussian process has low variance, and upweight it when the Gaussian process has high variance. Thus, we set the weights on the regularizing estimates to be inversely proportional to the weights on the Gaussian process estimates. Therefore, we set Σ_{RK}^{-1} to be Σ and Σ_{RD}^{-1} to be Σ_D. For brevity, we will write the first term of the minimization as K, the second D, the third R_K, and the fourth R_D from here on.

The minimization is in quadratic form, so we can solve it by computing the partial derivative with respect to Y and solving the sparse linear system that results using least squares.

Finally, the variances associated with the kinematics and dynamics estimates may need to be scaled with respect to each other. In our system, the artist sets four scalar weights that control how much the kinematics, dynamics, kinematics regularizer, and dynamics regularizer affect the solution. The final form of the minimization is:

$$\min_{Y}[w_kK + w_dD + w_{rk}R_K + w_{rd}R_D] \tag{6.13}$$

The weights have fairly intuitive meaning. If w_{rk} and w_{rd} are set to zero, then the prediction depends only on the Gaussian processes regression. This choice of weights can be dangerous because the estimates with high variance may be unreliable. Setting w_k and w_d to zero roughly yields the same result as traditional retargeting methods. In our experiments, we have found tuning the weights typically requires 2-3 iterations. Since solving the sparse linear system is fast, these iterations can be done quickly. We expect that the weights could alternatively be set automatically.

An Alternative: While we found these estimators to work well, other estimators may exist. In particular, we tried using an alternative scattered data interpolation technique called moving least squares. Moving least squares has been shown to produce good results for other graphics applications such as surface reconstruction (see for example, [67]). The moving least squares formulation is:

$$W_p A x = W_p B$$

where W_p is a diagonal weight matrix that depends on the source point we want to evaluate p, A is the source training data, and B is the target training data. x maps source vectors onto target vectors. We computed W using a Gaussian weighting function. We used the mean of the three highest weights as the confidence value. (Confidence can be considered a measure of inverse variance.)

This regression method gave us good kinematic predictions, and was also faster than using Gaussian processes. However, we found the variance values from Gaussian processes more reliable.

6.4 Features

In the previous section, we explained our procedure for predicting a target animation given kinematics predictors and dynamics predictors. Now we consider the input-output data for the predictors.

Recall from the last section that the kinematics predictor produces $\tilde{Y}_i$ given X_i. $\tilde{Y}_i$ is an estimate of the skeletal configuration. It also contains an estimate of the time elapsed since the last frame, and the ground plane translation and vertical rotation of the root relative to the root at the last frame. Integrating the differential time and root estimates yields estimates for the global time and global root path, respectively.

We encode the skeletal configuration as joint angles parameterized by exponential maps [24]. Alternatively, one could represent the configuration as joint positions in 3D space, but the regression can produce configurations in which the bones stretch/squash to

noticeably incorrect lengths. We encode the orientation of each bone relative to the orientation of its parent in the skeletal hierarchy. Additionally, we tried encoding the orientation relative to the skeleton's global orientation, which yielded mostly identical results.

X_i is represented using a feature vector that encodes the kinematics and dynamics at each frame of the source animation. The feature contains the skeletal configuration currently, 0.125 seconds ago, and 0.125 seconds in the future. (We could alternatively use a window of frames, but we found there was not much difference in the results.) Inclusion of past and future frames exposes the dynamics. We also tried using the current skeletal configuration only, but found that disambiguating frames with similar kinematics and different dynamics yielded more reliable confidence estimates. We represent the three skeletal configurations as joint positions in 3D space. (Notice that this is different from the skeletal configuration encoding we use for Y_i. We tried using joint angles, but got more reliable variance estimates using joint positions.) All three configurations are rigidly transformed so that the current skeletal configuration is at the origin with zero global orientation.

The dynamics predictor outputs an estimation of the second derivative of the target's skeletal configuration (represented as exponential maps), the second derivative of the root ground-plane position and vertical orientation relative to the root at the last frame, and the second derivative of the time elapsed since the last frame. The input to the dynamics predictor is the finite difference approximation to the second derivative of the source skeletal configuration (encoded as joint positions, factoring out the root position and orientation). Because the animation data may be noisy, we smooth the source's second derivative by averaging over a 0.125-second window before estimating the dynamics predictor.

6.5 Interactive motion editing

For our method to be useful, it is beneficial to have a system that helps artists to quickly and naturally create animate the target.

Our system allows the user to modify a particular pose $x(\hat{t})$ to obtain an edited pose $y(\hat{t})$ using inverse kinematics, where $\hat{t}$ is the time of the pose that the user is modifying.

Traditional inverse kinematics systems may be too slow or may only allow the user control over a limited set of end effectors at a time. Our system allows the user to change the joint locations as if they were independent.

Our system also allows the user to move joints individually or as a group. By painting weights onto joints, the user controls how much a joint will be affected by mouse movement. The user then drags the mouse to edit the character's configuration.

Let us say $p_i^\star$ is the 3D position i^{th} joint after the user manipulation. We first find the set of joint positions that are as close to the edited joint positions as possible while preserving bone lengths. This can be formulated as a constrained optimization:

$$\begin{aligned} \min_{p_i} \quad & \textstyle\sum_i |p_i - p_i^\star|^2 \\ s.t. \quad & |p_i - p_j| = d_{ij} \end{aligned}$$

where for every bone, d_{ij} is the length of the bone between joints i and j. The objective function and the constraints are quadratic. We solve for the joint positions p_i using Lagrange multipliers. Since the Hessian is sparse (and linear with respect to the degrees of freedom), we can solve this optimization efficiently with few iterations of Newton Raphson. Notice that we can easily constrain end effectors to be at a particular location by introducing additional quadratic constraints.

Now that we have computed joint positions in which the bone lengths are preserved, we can compute the joint angles. Unfortunately joint angles cannot be uniquely determined from joint positions (because of the twist degree of freedom around the axis of the bone). Since we already have the joint angles before the user manipulated the joint positions, we can compute the minimum rotation required for each bone to align with the new joint positions (Figure 6.2).

This way of performing inverse kinematics is efficient: we perform this computation and compute the joint angles every time the user drags the mouse. From the edited pose $y(\hat{t})$ and the original pose $x(\hat{t})$, we obtain the a kinematic delta key frame to construct the offset function $d(t)$ where $y(t) = d(t) + x(t)$.

We also allow the user to drag a particular pose in time to modify the dynamics. This

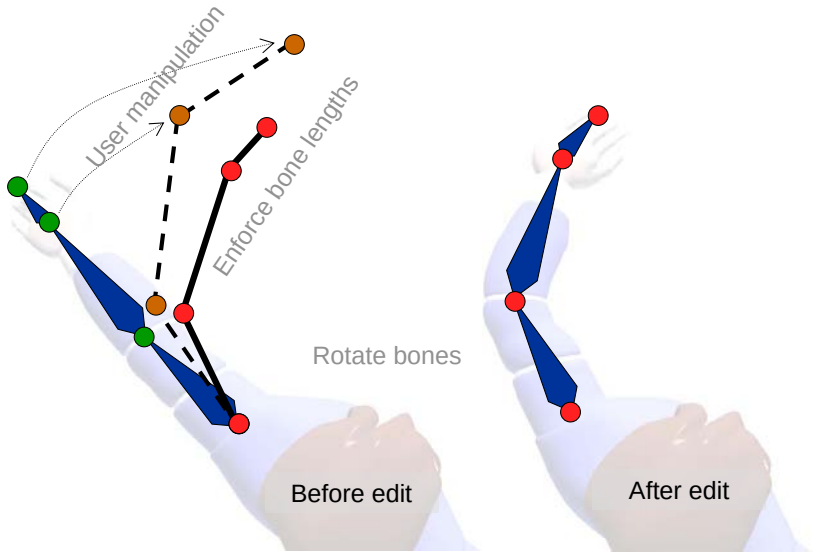

Figure 6.2. **Inverse Kinematics: Left:** The joints in the character's arm start out at the positions indicated by the green dots. The user drags the arm to the right so the new joint positions correspond to the brown dots, which makes the bones the wrong lengths. The system performs a constrained optimization to meet the user's request as closely as possible while maintaining bone lengths. The output of the optimization are the new joint positions shown in red. **Right:** The system computes the minimal rotation required to align the character's arm with the new joint positions to yield the new bone configuration shown in blue.

gives us the timing delta function $k(t)$ where $y(t) = d(t) + x(t + k(t))$. This way, the user can directly author kinematic offsets as well as timing differences between the source and target animations interactively (Figure 6.3).

6.6 Animator workload

To examine our model and better understand how it might be used in practice, we conducted structured interviews [6] with four professional animators (three from major movie studios, and one with over 20 years of experience). We asked them about their work practices, and about which aspects of editing character animation are challenging. From these interviews, we distilled two areas they deemed important and difficult that our technique can help with.

- **Degree of expressiveness:** A traditional retargeting technique is to map the source

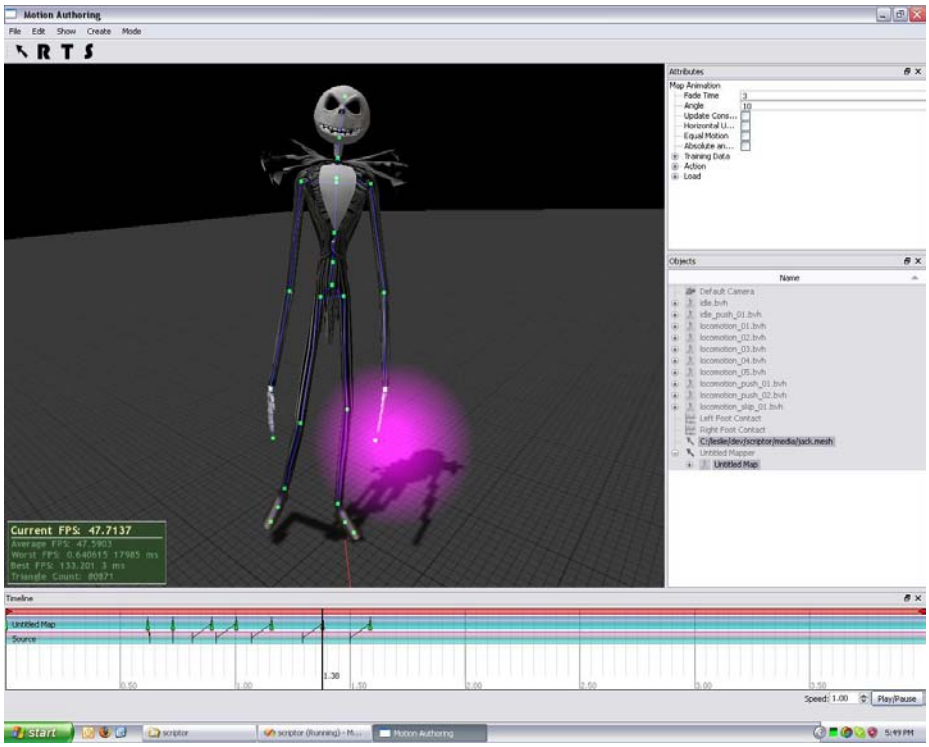

Figure 6.3. **Interactive motion editing environment:** The animator can pose the character by dragging individual green joints on the blue skeleton, or can move multiple joints by first painting influence weights onto the skeleton using the Gaussian-shaped brush shown in purple. The green arrows on the bottom part of the figure mark keyframes on the timeline. The positions of the arrows (on the timeline at the **bottom**) means that the animator has stretched time between the second and third keyframes.

animation onto the original character. However, while the target character moves like the source, the motion may not fit the target character, because the target character may have different physical properties, movement style, etc. Getting highly individualized, expressive motion usually requires hand-editing. Our technique can propagate an artist's edits, so the artist can create expressive animation without having to create entire sequences by hand.

- **Graphic strength:** An animator is often careful to produce frames that look good kinematically and dynamically. High kinematic quality is sometimes referred to as *graphic strength*. Producing graphically strong frames can be difficult because of the kinematic detail involved. Therefore, producing long animations with graphic strength

can be hard. Our system can help the artist by propagating poses that are graphically strong.

Our results suggest that the combination of an intuitive editing tool and a system to generalize and propagate edits can reduce the work associated with both of these aspects (see Section 6.8).

6.7 Limitations

One limitation of our method is that Gaussian processes are not very scalable. For training, they demand $O(n^3)$ time and $O(n^2)$ storage, where n is the number of training frames. While there are methods like the Informative Vector Machine [40] which speed up training considerably, such techniques may not actually be necessary for our problem. It generally takes a great deal of time to produce edits. (Otherwise artists could do them all by hand and there would be no need for automatic techniques.) Therefore, we will usually face limited training data.

A second limitation is that we do not directly handle environmental contacts like footstrikes. Having a method that keeps contacts from slipping is desirable. Like many other techniques, we clean up contacts like foot strikes with an automated post-processing step [37, 29].

Third, our system assumes a single model of artist edits. Hence, our system may not be useful for characters whose movements need to be specialized per-scene to look good from the camera, for example.

Fourth, our method can fail to generalize the artist's edits to motions that are completely different from the ones in the training set. The artist can adjust the weights controlling how much each of the estimators influences the regression, but the artist has more direct control over the result by adding more training examples.

A fifth limitation is that currently, our implementation of our regression system is not real-time. Estimating the parameters for the Gaussian processes consumes 10-20 seconds,

and producing frame estimations from the Gaussian processes proceeds at 2-3 frames per second. We did not optimize our code for speed, so we assume that these times could be considerably improved by a careful implementation. However, even these running times are fairly acceptable for our application.

6.8 Results

Using our method, we created animations for a skeleton, a space ranger, a giant, and Elvis. Even though these characters have different physical properties, personalities, and styles, our technique was able to drive all of them with the same pre-existing motion capture database.

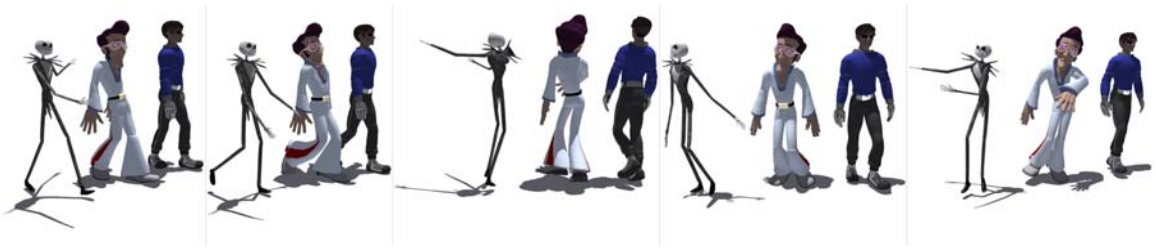

Figure 6.4. Our technique can animate a wide variety of characters. During training, an artist modified existing animations to make the skeleton move in a loose, swinging way, and to make Elvis move like a rock star. Our technique learns and generalizes the artist's modifications. Now, given new frames of motion capture (shown on the character with the blue shirt on the right side of each frame), our method can create new poses for the skeleton and Elvis automatically. Both characters move in unique ways that are faithful to the artist's modifications.

Interactive motion editing: All of the artist-generated examples exhibited here were generated using the interactive motion editing system described in Section 6.5. We found it to be an intuitive way to fluidly edit animation.

Characteristic properties: Characteristic properties of the example animations transfer well to the target motion. For example, the skeleton has different proportions than a person, and is likely to move differently because he is modeled in a cartoon fashion. We show that not only can we make him move like the examples the artist provided, but that distinctive features of the training data manifest in the target animation. For exam-

ple, the training data shows idiosyncratic marks, such as bowed legs. (The artist we used is a computer science graduate student, not a professional animator, so these distinctive features are expected.) The bowed legs and other idiosyncrasies are clearly visible in the target animation.

Similarly, Elvis is modeled in a cartoon style, so he will likely move differently from a real human. We show that we can make him move like the examples provided, so that he struts instead of walks, for example.

Length of animation: The work involved in editing frames is generally linear in the number of keyframes needed, which is usually proportional to the total number of frames. However, motion shows a great deal of repetition. Our method essentially encodes repeating structures in the animations so that the artist does not have to deal with every instance.

Hence, the work involved in editing frames using our system can be considerably less than traditional editing methods. The number of training frames needed for each example we generated is on the order of a few seconds. For example, we created ten-minute long, high quality walking animations for Elvis, the giant, and Jack using our method. The Elvis example required 7.5 seconds of training data, the giant required 22.2 seconds of data, and Jack took 14.5 seconds of data. To make Jack and Elvis skip, we required 5.92 seconds of training data for Jack, and 5.8 seconds for Elvis. Recall that previous work required a minimum of one to two minutes of training data [27].

Extreme physical differences: Our method works even when the source and target characters move very differently. The artist animates our giant character with a slow gait, wobbly stomach, and short steps. These properties are preserved in the target animation we generated, even though it is being driven by motion capture of an average-sized person (Figure 6.5).

Controllability: Our method is directly controllable. We animated a space ranger character directly from our existing motion capture database. We show that by providing a simple example that is 9.2 seconds long, we can make him lean back while walking. By providing another example (1.2 seconds long), we can remove the lean while he is standing.

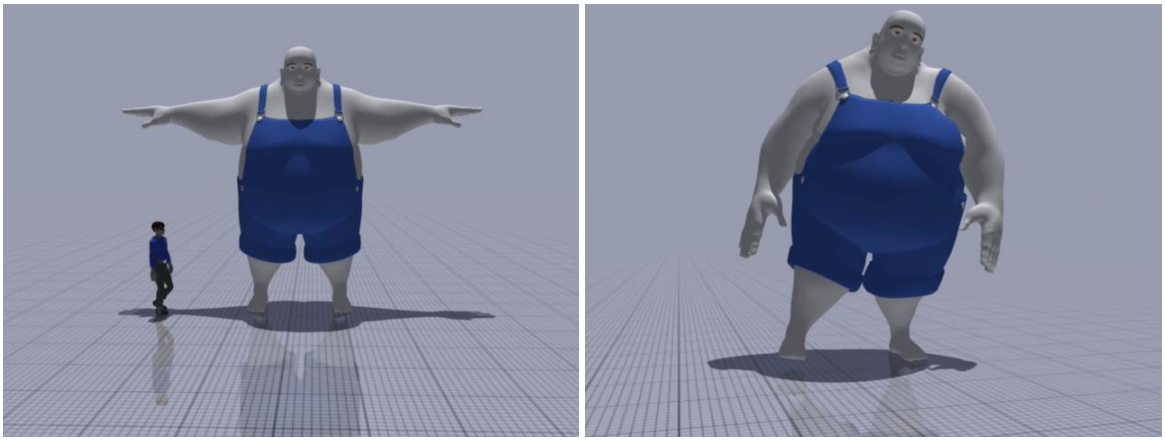

Figure 6.5. Our method works even when there are extreme physical differences between the source and target, like between the human and giant on the **left**. On the **right**, we show a frame from our results.

And by replacing the walking example with another example that is also 9.2 seconds long, we can make him walk like a zombie (Figure 6.6).

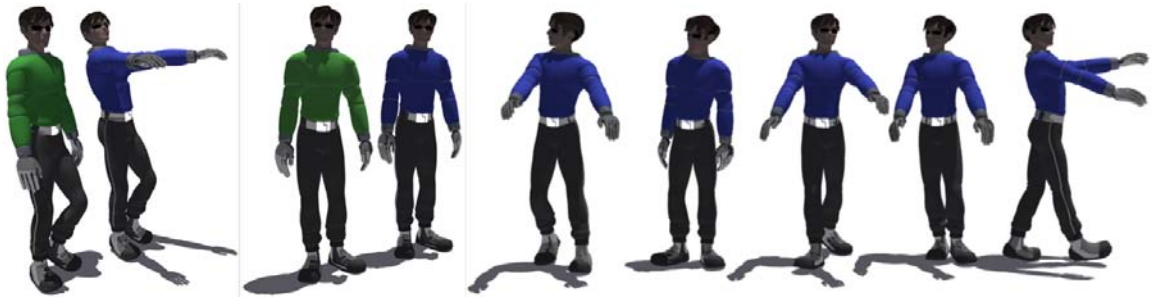

Figure 6.6. An artist wants the space ranger character shown here to walk with his arms extended, but to stand with his arms hanging down. In the **left** and **center** frames, we show the source frames in green and the target frames in blue. **Right:** Here we show frames automatically generated from our system. Note that the frames do not represent a time-lapsed shot. They were chosen to demonstrate that the character walks and stands like the artist wanted, and that the method generalizes the artist's edits well, so that transitions and turns look natural. None of the frames shown in this panel are from the training examples.

Chapter 7

Conclusion

As we discussed in Chapter 2, there are three major methodologies in the current motion synthesis literature. *Data-driven* techniques are easy to use because they require the user only to set a few constraints. The user generally does not need to know how to animate by hand. Data-driven techniques are also usually efficient, capable of synthesizing motion in real-time or faster. The major drawback of data-driven techniques is that they can only generate motions similar to the examples they are given.

Artist-driven techniques are the most controllable and flexible. However, these techniques require a great deal of skill and are time-consuming because the artist must control many degrees of freedom. Artist-driven techniques are thus not suitable for real-time applications.

Procedural techniques show the most promise to be both automatic and general. The user designs controllers or optimization functions to produce motion. The biggest drawbacks are that procedural techniques usually require a large amount of computation. For example, Liu et al. report that it takes 8-20 minutes to compute a 40-60 second animation using their procedural framework [46]. Also, currently, procedural methods produce less realistic motion than example-based methods. It is an open and challenging research question to write procedures that always generate realistic human motion. Procedural methods encode physical laws well, but human motion is the result of more than physical considerations. The

motor planning mechanism humans use remains mysterious. That is, we still do not know how the body translates a demand like "Pick up the cat" into muscle activation signals, despite a great deal of research and interest.

The motion synthesis requirements of some applications fall clearly into one methodology. However, other important applications do not. These applications typically need a combination of generality, efficiency, and control. For these applications, hybrid techniques may be suitable.

This thesis presents three example/artist hybrid techniques. All of these methods strive to make creating animation easier and more efficient for artists. We describe methods for labeling frames of motion with foot plant labels, scoring synthesized motion, and editing motion. These techniques are most useful for applications which use large databases of motion containing several variants of the same motion, as is common in motion capture databases.

We believe that the space of hybrid techniques should be explored further. There are few methods which can be classified as hybrids, and the work in this thesis as well as other work suggests that hybrid techniques may be useful. They may meet the motion synthesis needs of applications that are ill-served by the three current methodologies. For example, video games require reactive, real-time motion synthesis, which is difficult to achieve with current techniques. Hybrid procedural/example-based techniques show promise to fill this need [5, 81].

Notably, to our knowledge, there are no artist/procedure/example hybrid techniques. Clearly each methodology offers unique strengths. In fact, one of the grand challenges of motion synthesis is to create a mechanism that has all of these strengths: control, efficiency, and generality. The ideal mechanism would power a character with the full motor capabilities of a complex biological entity, such as a human. The character's movements would look like high-quality keyframed animation. However, the character would be capable of performing a wide range of movements without (or without a large amount of) training data, as we expect will be the case from procedural algorithms. The ideal character would

have high-level controls like "Dance the samba to this music," as well as low-level controls, enabling artists to easily manipulate subtle nuances, as in traditional artist-driven systems.

The solution may or may not be a hybrid of all three methodologies. However, it seems reasonable to believe that further exploration of the space of hybrid techniques will yield useful methods.

Bibliography

[1] Kenji Amaya, Armin Bruderlin, and Tom Calvert. Emotion from motion. In *Graphics Interface 1996*, 1996.

[2] Okan Arikan. Compression of motion capture databases. In *SIGGRAPH 2006*, 2006.

[3] Okan Arikan and David A. Forsyth. Interactive motion generation from examples. In *SIGGRAPH 2002*, 2002.

[4] Okan Arikan, David A. Forsyth, and James F. O'Brien. Motion synthesis from annotations. In *SIGGRAPH*, 2003.

[5] Okan Arikan, David A. Forsyth, and James F. O'Brien. Pushing people around. In *SCA*, 2005.

[6] Hugh Beyer and Karen Holtzblatt. *Contextual Design: A Customer-Centered Approach to Systems Designs*. Morgan Kaufmann, 1998.

[7] Rama Bindiganavale and Norman I. Badler. Motion abstraction and mapping with spatial constraints. In *CAPTECH 1998*, 1998.

[8] Bobby Bodenheimer, Chuck Rose, Seth Rosenthal, and John Pella. The process of motion capture: Dealing with the data. In *Computer Animation and Simulation 1997*, 1997.

[9] Matthew Brand and Aaron Hertzmann. Style machines. In *SIGGRAPH 2000*, 2000.

[10] Christoph Bregler, Lorie Loeb, Erika Chuang, and Hrishi Deshpande. Turning to the masters: motion capturing cartoons. In *SIGGRAPH 2002*, 2002.

[11] Armin Bruderlin and Lance Williams. Motion signal processing. In *SIGGRAPH 1995*, 1995.

[12] John Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1986.

[13] Jinxiang Chai and Jessica K. Hodgins. Performance animation from low-dimensional control signals. In *SIGGRAPH 2005*, 2005.

[14] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[15] Mira Dontcheva, Gary Yngve, and Zoran Popovic. Layered acting for character animation. In *SIGGRAPH 2003*, 2003.

[16] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. In *SIGGRAPH 2001*, 2001.

[17] D. A. Forsyth and M. M. Fleck. Automatic detection of human nudes. *Int. J. Comput. Vision*, 32(1):63–77, 1999.

[18] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[19] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the nystrom method. In *PAMI*, 2004.

[20] Michael Girard and A. A. Maciejewski. Computational modeling for the computer animation of legged figures. In *SIGGRAPH 1985*, 1985.

[21] Michael Gleicher. Motion editing with spacetime constraints. In *Symposium on Interactive 3D graphics 1997*, 1997.

[22] Michael Gleicher. Retargetting motion to new characters. In *SIGGRAPH*, 1998.

[23] Michael Gleicher, Hyun Joon Shin, Lucas Kovar, and Andrew Jepsen. Snap-together motion: assembling run-time animations. In *I3D*, 2003.

[24] F. Sebastin Grassia. Practical parameterization of rotations using the exponential map. *J. Graph. Tools*, 3, 1998.

[25] Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popovic. Style-based inverse kinematics. In *SIGGRAPH 2004*, 2004.

[26] Eugene Hsu, Sommer Gentry, and Jovan Popovic. Example-based control of human motion. In *SCA 2004*, 2004.

[27] Eugene Hsu, Kari Pulli, and Jovan Popovic. Style translation for human motion. In *SIGGRAPH 2005*, 2005.

[28] T. Igarashi, T. Moscovich, and J. F. Hughes. Spatial keyframing for performance-driven animation. In *SCA 2005*, 2005.

[29] Leslie Ikemoto, Okan Arikan, and David Forsyth. Knowing when to put your foot down. In *I3D*, 2005.

[30] Leslie Ikemoto and David A. Forsyth. Enriching a motion collection by transplanting limbs. In *SCA*, 2004.

[31] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC 1998: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 1998.

[32] Johansson. Visual motion perception. 1976.

[33] Lucas Kovar and Michael Gleicher. Footskate cleanup for motion capture editing. In *SCA*, 2002.

[34] Lucas Kovar and Michael Gleicher. Flexible automatic motion blending with registration curves. In *SCA*, 2003.

[35] Lucas Kovar and Michael Gleicher. Automated extraction and parameterization of motions in large data sets. 2004.

[36] Lucas Kovar, Michael Gleicher, and Frederic Pighin. Motion graphs. In *SIGGRAPH 2002*, 2002.

[37] Lucas Kovar, John Schreiner, and Michael Gleicher. Footskate cleanup for motion capture editing. In *SCA 2002*, 2002.

[38] Taesoo Kwon and Sung Yong Shin. Motion modeling for on-line locomotion synthesis. In *SCA 2005*, pages 29–38, New York, NY, USA, 2005. ACM Press.

[39] Neil Lawrence. Gaussian process models for visualisation of high dimensional data. In *NIPS 2004*, 2004.

[40] Neil D. Lawrence and John C. Platt. Learning to learn with the informative vector machine. In *ICML 2004*, 2004.

[41] Benoît Le Callennec and Ronan Boulic. Robust Kinematic Constraint Detection for Motion Data. In *SCA*, 2006.

[42] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. In *SIGGRAPH 2002*, 2002.

[43] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *SIGGRAPH 1999*, 1999.

[44] Yan Li, Tianshu Wang, and Heung-Yeung Shum. Motion texture: a two-level statistical model for character motion synthesis. In *SIGGRAPH 2002*, 2002.

[45] C. Karen Liu, Aaron Hertzmann, and Zoran Popovic. Learning physics-based motion style with nonlinear inverse optimization. In *SIGGRAPH 2005*, 2005.

[46] C. Karen Liu, Aaron Hertzmann, and Zoran Popovic. Composition of complex optimal multi-character motions. In *SCA 2006*, 2006.

[47] C. Karen Liu and Zoran Popovic. Synthesis of complex dynamic character motion from simple animations. In *SIGGRAPH*, 2002.

[48] Lennart Ljung, editor. *System identification (2nd ed.): theory for the user*. Prentice Hall PTR, 1999.

[49] J. McCann, N. S. Pollard, and S. Srinivasa. Physics-based motion retiming. In *SCA*, 2006.

[50] L. Molina-Tanco and A. Hilton. Realistic synthesis of novel human movements from a database of motion capture examples. In *Workshop on Human Motion (HUMO'00)*, pages 137–142, 2000.

[51] Tomohiko Mukai and Shigeru Kuriyama. Geostatistical motion interpolation. In *SIGGRAPH 2005*, 2005.

[52] Sang Il Park, Hyun Joon Shin, Tae Hoon Kim, and Sung Yong Shin. On-line motion blending for real-time locomotion generation: Research articles. *Comput. Animat. Virtual Worlds*, 15(3-4):125–138, 2004.

[53] Sang Il Park, Hyun Joon Shin, and Sung Yong Shin. On-line locomotion generation based on motion blending. In *SCA 2002*, pages 105–111, New York, NY, USA, 2002. ACM Press.

[54] Ken Perlin and Athomas Goldberg. Improv: a system for scripting interactive actors in virtual worlds. In *SIGGRAPH 1996*, 1996.

[55] Zoran Popovic and Andrew Witkin. Physically based motion transformation. In *SIGGRAPH 1999*, 1999.

[56] A. Pothen, H.D. Simon, and K.P. Liou. Partitioning sparse matrices with eigenvectors of graphs. In *SIAM Journal of Matrix Anal. Appl.*, 1990.

[57] Katherine Pullen and Christoph Bregler. Motion capture assisted animation: texturing and synthesis. In *SIGGRAPH 2002*, 2002.

[58] Carl Rasmussen and Christopher Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[59] Carl Rasmussen and Christopher Williams. Gaussian processes for machine learning software, http://www.gaussianprocess.org/gpml/code/matlab/doc/, 2006.

[60] Paul S. A. Reitsma and Nancy S. Pollard. Perceptual metrics for character animation: sensitivity to errors in ballistic motion. In *SIGGRAPH 2003*, New York, NY, USA, 2003. ACM Press.

[61] Liu Ren, Alton Patrick, Alexei A. Efros, Jessica K. Hodgins, and James M. Rehg. A data-driven approach to quantifying natural human motion. *SIGGRAPH 2005*, 2005.

[62] Charles Rose, Michael F. Cohen, and Bobby Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Comput. Graph. Appl.*, 18(5), 1998.

[63] Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. Efficient generation of motion transitions using spacetime constraints. In *SIGGRAPH 1996*, pages 147–154, New York, NY, USA, 1996. ACM Press.

[64] Jonas Rubenson, Denham B. Heliams, David G. Lloyd, and Paul A. Fournier. Gait selection in the ostrich: mechanical and metabolic characteristics of walking and running with and without an aerial phase. *Proc Biol. Sci*, May 2004.

[65] Alla Safonova and Jessica K. Hodgins. Analyzing the physical correctness of interpolated human motion. In *SCA 2005*, 2005.

[66] Alla Safonova, Jessica K. Hodgins, and Nancy S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. In *SIGGRAPH 2004*, 2004.

[67] Chen Shen, James F. O'Brien, and Jonathan R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. In *SIGGRAPH 2004*, 2004.

[68] Hyun Joon Shin, Lucas Kovar, and Michael Gleicher. Physical touch-up of human motions. In *Pacific Graphics 2003*, 2003.

[69] Hyun Joon Shin, Lucas Kovar, and Michael Gleicher. Physical touch-up of human motions. In *Pacific Conference on Computer Graphics and Applications*, 2003.

[70] Hyun Joon Shin, Jehee Lee, Sung Yong Shin, and Michael Gleicher. Computer puppetry: An importance-based approach. *ACM Trans. Graph.*, 20(2), 2001.

[71] Seyoon Tak and Hyeong-Seok Ko. A physically-based motion retargeting filter. *ACM Trans. Graph.*, 24(1), 2005.

[72] Seyoon Tak, Oh young Song, and Hyeong-Seok Ko. Motion balance filtering. *Eurographics*, 2000.

[73] M. Vukobratovic and D. Juricic. Contributions to the synthesis of biped gait. In *IEEE Transactions on Biomedical Engineering*, 1969.

[74] Jack Wang, David Fleet, and Aaron Hertzmann. Gaussian process dynamic models. In *NIPS 2005*, 2005.

[75] Jing Wang and Bobby Bodenheimer. An evaluation of a cost metric for selecting transitions between motion segments. In *SCA 2003*, 2003.

[76] Jing Wang and Bobby Bodenheimer. Computing the duration of motion transitions: an empirical approach. In *SCA 2004*, 2004.

[77] Douglas J. Wiley and James K. Hahn. Interpolation synthesis for articulated figure motion. In *Virtual Reality Annual International Symposium (VRAIS '97)*, 1997.

[78] David Winter. *Biomechanics and Motor Control of Human Movement, Third edition.* John Wiley and Sons, 2005.

[79] Andrew Witkin and Michael Kass. Spacetime constraints. In *SIGGRAPH 1988*, 1988.

[80] Andrew Witkin and Zoran Popovic. Motion warping. In *SIGGRAPH 1995*, 1995.

[81] Victor Zordan, Anna Majkowska, Bill Chiu, and Matthew Fast. Dynamic response for motion capture animation. *SIGGRAPH*, 2005.