AUTOMATED AUTHORING OF QUALITY HUMAN MOTION FOR INTERACTIVE ENVIRONMENTS

by

Rachel Marie Heck

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

Doctor of Philosophy

(Computer Science)

at the

UNIVERSITY OF WISCONSIN–MADISON

2007

ACKNOWLEDGMENTS

Graduate school has been a wonderful adventure, one that I could not have completed without the support of many. In particular, my advisor Dr. Michael Gleicher has been a constant source of help and inspiration. Since I started graduate school with almost no knowledge of Computer Graphics, I can honestly credit him with teaching me much of my area of expertise. He is also a phenomenal source for networking, a skill that is beneficial in all walks of life. I must also thank the other members of my committee: Dr. Charles Dyer, Dr. Nicola Ferrier, Dr. Darryl Thelen, and Dr. Benjamin Liblit. The comments and questions I have received from my committee members over the years have helped me to improve my work considerably.

Thank you also to the members of the Computer Graphics and Vision Lab, both past and present. My conversations with the other students in the lab have been very helpful for working out technical issues; these conversations have also been some of the most enjoyable moments I have experienced in graduate school. In particular, thank you to Dr. Lucas Kovar, Alex Mohr, Eric McDaniel, and Matthew Anderson for your help with my work and your acceptance of me when I joined the group. Thank you also to Greg Cipriano for agreeing to be the subject for my gaze experiments, despite the cold capture room, and to Dr. Hyun Joon Shin, whose insightful conversations helped me through a number of stumbling blocks. And thank you to Michael Wallick, who started graduate school along with me; your immense help in all things, from virtual videography to motion capture shoots, and wonderful friendship made a huge impression on my graduate school career.

Because my work depends heavily on having access to motion capture data, I also wish to thank those who have contributed to my database in some way. Thank you to Dr. Jehee Lee for his donated boxing data and to the folks at CMU for putting together the CMU Graphics Lab Motion

Capture Database. And a big thank you to Dr. Bryan Heiderscheit for allowing us direct access to motion capture technology, as well as for the time he spent supervising our shoot.

I also sincerely thank my amazing group of family and friends. You all mean so much to me. In particular, my mother (Carol Heck), father (Daniel Heck), sister (Shaina Heck), brother (Cory Heck), and grandmother (Audrey Holzschuh) have supported me for many years, helping and encouraging me to grow and learn. Finally, thank you to my best friend and dearest companion, my husband, Justin Rose. Your kindness, sense of humor, and love mean everything.

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

Appendix

Page

AUTOMATED AUTHORING OF QUALITY HUMAN MOTION FOR INTERACTIVE ENVIRONMENTS

Rachel Marie Heck

Under the supervision of Associate Professor Michael Gleicher

At the University of Wisconsin-Madison

Many computer applications depend on the visual realism of virtual human character motion. Unfortunately, it is difficult to describe what makes a motion look real yet easy to recognize when a motion looks fake. These characteristics make synthesizing motions for a virtual human character a difficult challenge. A potentially useful approach is to synthesize high-quality, nuanced motions from a database of example motions. Unfortunately, none of the existing example-based synthesis techniques has been able to supply the quality, flexibility, efficiency and control needed for *interactive applications*, or applications where a user directs a virtual human character through an environment. At runtime, interactive applications, such as training simulations and video games, must be able to synthesize motions that not only look realistic but also quickly and accurately respond to a user's request. This dissertation shows how *motion parameter decoupling* and *highly structured control mechanisms* can be used to synthesize high-quality motions for interactive applications using an example-based approach. The main technical contributions include three example-based motion synthesis algorithms that directly address existing interactive motion synthesis problems: a method for splicing upper-body actions with lower-body locomotion, a method for controlling character gaze using a biologically and psychologically inspired model, and a method for using a new data structure called a parametric motion graph to synthesize accurate, quality motion streams in realtime.

Michael Gleicher

ABSTRACT

Many computer applications depend on the visual realism of virtual human character motion. Unfortunately, it is difficult to describe what makes a motion look real yet easy to recognize when a motion looks fake. These characteristics make synthesizing motions for a virtual human character a difficult challenge. A potentially useful approach is to synthesize high-quality, nuanced motions from a database of example motions. Unfortunately, none of the existing example-based synthesis techniques has been able to supply the quality, flexibility, efficiency and control needed for *interactive applications*, or applications where a user directs a virtual human character through an environment. At runtime, interactive applications, such as training simulations and video games, must be able to synthesize motions that not only look realistic but also quickly and accurately respond to a user's request. This dissertation shows how *motion parameter decoupling* and *highly structured control mechanisms* can be used to synthesize high-quality motions for interactive applications using an example-based approach. The main technical contributions include three example-based motion synthesis algorithms that directly address existing interactive motion synthesis problems: a method for splicing upper-body actions with lower-body locomotion, a method for controlling character gaze using a biologically and psychologically inspired model, and a method for using a new data structure called a parametric motion graph to synthesize accurate, quality motion streams in realtime.

Chapter 1

Introduction

Many computer applications prominently feature virtual human characters. For example, in the entertainment industry, virtual humans appear in movies as digital body doubles to perform super-human feats, and video games often depict a virtual human figure interacting with an environment. Virtual human characters also appear frequently in training simulations to supplement on-the-job training or in-class learning in a wide range of industries: the military uses training simulations to help train soldiers in combat, athletes use simulations to improve their game, construction workers can practice using fork lifts and bulldozers in a simulation, and there is talk now of using training simulations to better teach medical students bed-side manner.

It is often important that the virtual human figures in these applications appear lifelike to help create a sense of immersion. While there are a number of different characteristics that contribute to a character's believability, one important aspect is the way a character moves. Unfortunately, human motion is a complicated process driven by a person's subconscious goals. Bones, muscles, and other tissues work together to make a human's body perform a specific task. Creating animations for a computer character that mimic these complicated processes is a challenge. This challenge is further complicated by the fact that people are good at recognizing when a motion looks fake, even if they cannot pinpoint the exact reason why. This dissertation is concerned with the automated

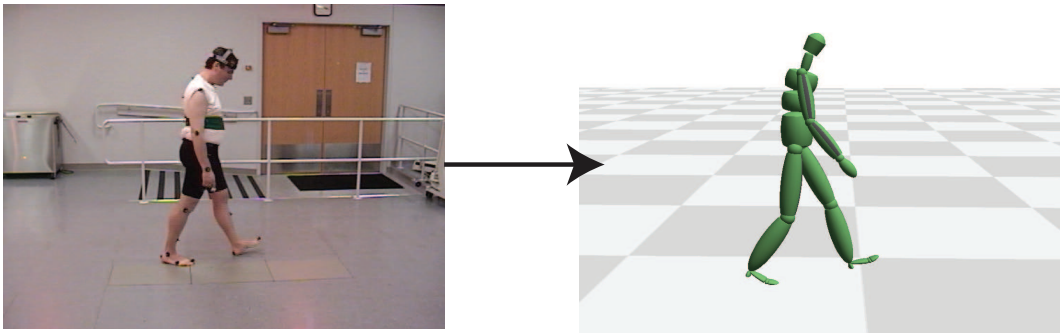


Figure 1.1 Motion capture is the process by which observed human motion is translated into a digital character motion.

synthesis of quality human motion for *interactive applications*, or applications like training simulations and video games where a virtual human character is directed through an environment by a user.[1]

Traditionally, talented and patient artists tackled the challenge of authoring believable animations for human characters for movies and video games. More recently, these artists have been aided by the introduction of *motion capture*. Motion capture is a set of techniques that use sensors to observe and record the detailed motion of an object or set of objects (see Figure 1.1). While the quality of the recorded motion and the process of capturing that motion vary from one motion capture technology to another, motion capture is capable of producing accurate representations of the way a person moves [Men00].

Motion capture is a potentially useful technique for creating believable human motion as it produces animations with all of the subtle nuances of real motion. The problem with motion capture data is that it is only directly useful for reproducing a specific motion. For example, the motion of a person reaching toward the front of a shelf is different in small but perceivable ways from the motion of that same person reaching toward the back of that same shelf; modifying a motion by hand in order to capture all of these subtle changes can be difficult.

[1] For the purposes of this dissertation, I am concerned with synthesizing motion for the character that is under the direct control of the user. However, the techniques discussed in this document can also be applied to non-player characters (NPCs) found in an interactive application.

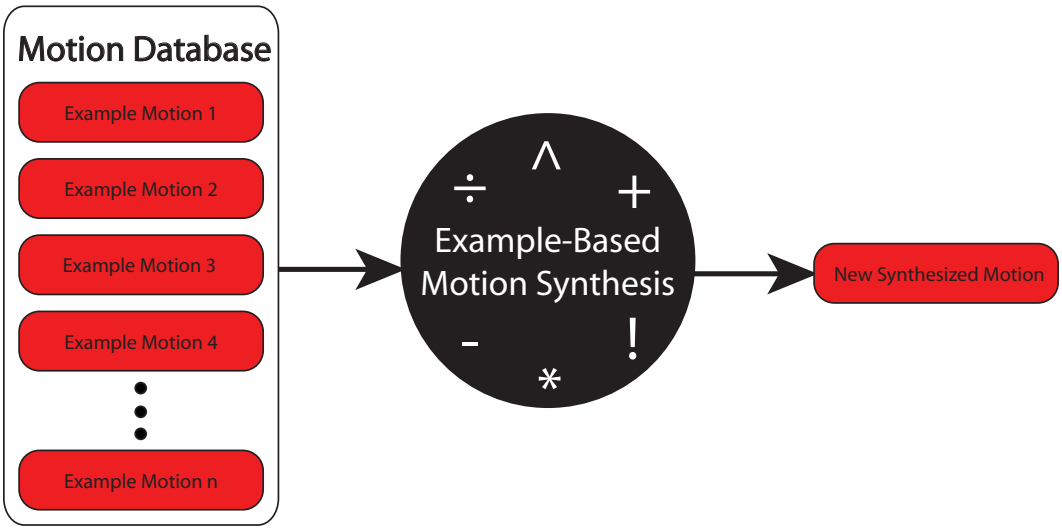


Figure 1.2 Example-based motion synthesis uses a database of example motions to synthesize new ones.

Recent research has focused on ways to improve the utility of motion capture data. The goal of much of this research is to use a database of motion-captured examples to generate new motions that appear as if they were created in the same way as the originals. These techniques are called *example-based motion synthesis methods* (see Figure 1.2). Research in example-based motion synthesis has greatly increased the usefulness of motion capture data by providing new ways to create believable human motion.

While current research has provided new techniques for reusing motion capture data, creating believable human motion is still a problem in many domains. In particular, interactive applications place additional constraints on the way human motion is generated. At runtime, interactive applications must be able to quickly generate motions that not only look realistic but also accurately meet user requests. At the same time, in order to encourage experimentation and allow quality motion generation for interactive applications with a small budget, the process of authoring human character motion should not be overly time consuming.

Unfortunately, current approaches to authoring and synthesizing human motion fall short of meeting all of these constraints at once. Video game companies spend hundreds of man hours carefully hand-tailoring motions that allow efficient motion generation with low latency but sacrifice visual realism. Alternatively, example-based motion generation techniques provide automated authoring of visually realistic motions, but these methods require too much synthesis time and respond too slowly for use in interactive applications. Automated authoring methods that could provide all of the characteristics described above at the same time would not only improve the quality of human motion in high-budget video games and simulations but would also allow low-budget interactive applications to benefit from the ability to generate realistic human motion. *My goal is to increase the utility of example-based motion generation methods for interactive applications by supplying insight into techniques that provide efficiency, low latency, high accuracy, quality, and automated authoring.* For a more detailed explanation of each of these characteristics, see Section 1.1.

There are two distinct but related motion synthesis subproblems. The first is the problem of generating individual motions, or *motion clips*, that meet specified constraints. For instance, an

application might need to synthesize a motion clip where a character walks at a specified curvature and looks in a specified direction. The second problem is how to produce long, continuous *motion streams*. For example, an application might need to synthesize a motion where a character walks at a specified curvature, *then* picks up an item from a shelf, *then* skips away, and so on. Some motion synthesis techniques aim to solve both problems while others focus on only one. This dissertation addresses both. ***It is my thesis that example-based techniques that decouple motion parameters to synthesize motion clips and that use highly-structured control mechanisms for transitioning between these individual clips can provide automated authoring of high-quality, controllable human motion streams for interactive applications.***

This thesis is based on two observations, each of which corresponds to one of the two motion synthesis subproblems. First, any sufficiently complex interactive application needs to generate segments of motion with many different constraints or *parameters*. For example, one application may need to generate motions of a human walking and running at different speeds and curvatures while performing some action with their upper body and gazing at different locations in an environment. It is important that we be able to decouple these parameters during motion synthesis, allowing motions that meet a subset of the parameters to be generated independently and then combined into one composite motion. Unfortunately, most example-based motion synthesis techniques generate an entire frame, or all the degrees of freedom (DOFs), of motion at once. Without decoupling, we face a combinatorial explosion in the number of motions that need to be captured. We would need an example of a character performing every possible combination of controllable parameters. Decoupling allows motions for specific parameters to be captured independently. Similarly, by decoupling, it is possible to produce motions with an unexpected combination of parameters. For example, I may not expect that I will need to be able to control both the location that a character reaches to as well as the direction a character looks simultaneously. Decoupling would allow me to produce motions that control both characteristics without needing to return to a motion capture studio to capture additional data that exhibits variations in both facing direction and reach location at the same time. And, finally, decoupling lends itself well to parallel processing. *Methods that*

allow the decoupling of motion parameters during synthesis can improve the accuracy, synthesis time, and storage requirements of generated motions.

The second observation is that human characters in interactive applications often need to be able to perform many different actions. For example, a character might be able to run, walk, jump, punch, climb ladders, duck, swim, sit down, etc. A number of example motions would be needed for each of these actions in order to synthesize accurate human motion using an example-based approach. It is important that an interactive application be able to find specific motions within this large number of examples in order to accurately synthesize new human motion quickly at runtime and to produce good transitions between the generated motion clips. Structured character control mechanisms facilitate fast decision-making by simplifying the problem through organization. And because structure can be used to organize large numbers of motion clips, it allows the control of many different types of motion (e.g., running, walking, punching, and jumping), as well as a lot of variation within a single motion type (e.g., a character might be able to punch to many more locations in space). *Highly-structured motion representation models allow for fast and accurate motion control without extensively sacrificing quality*.

In order to support my thesis, I present algorithms that decouple some of the most common and important motion parameters and introduce a highly-structured motion transition representation that allows easy interactive control because of its structure. In particular, I make the following specific technical contributions:

1. **Runtime Method for Splicing Upper-Body Actions with Locomotion (Chapter 4).** Interactive applications often divide the problem of character planning into two pieces: what action the character is performing and how/where the character is moving. I provide a method that allows motion synthesis to also *decouple* character action from locomotion, allowing *automated authoring* of *quality* motions that can be *synthesized quickly* and *with a significantly decreased number of required example motions for accurate character control*.

2. **Runtime Method for Adjusting the Gaze Direction of a Character (Chapter 5).** While many have recognized the importance of head facing direction, or *gaze*, for understanding a

person's motivations and for exhibiting realism in virtual characters, few have studied ways to edit gaze in a realistic way. I provide a method for *altering gaze at runtime independent of other motion parameters* by using examples from a motion capture database to build a low-dimensional representation of gaze that is based on scientific findings related to the way humans move. The method allows *quality* motions to be *authored in an automated way without extensive storage requirements*.

3. **Method for Automated Authoring of Parametric Motion Graphs (Chapter 6).** I provide a simple, *automated* method for building parametric motion graphs, or *highly-structured*, graph-based control mechanisms that combine parametric synthesis to gain *accuracy and flexibility* and synthesis by concatenation to produce *quality transitions* between motions.

4. **Runtime Method for Using Parametric Motion Graphs (Chapter 6).** I show how to use parametric motion graphs at runtime to produce *accurate, high-quality, responsive* motion streams at *interactive speeds*. Because parametric motion graphs are *highly-structured, little authoring effort* is needed to use a graph for interactive character control.

With these contributions I show the utility of decoupling parameters during motion clip generation by providing distinct decoupling solutions to two different but important motion generation problems. I also illustrate the usefulness of highly structured control mechanisms for synthesizing controllable motion streams from motion clips by showing how parametric motion graphs provide all of the required properties of interactive applications because of their structure.

The remainder of this chapter provides a more detailed description of the difficulties involved with synthesizing human motion for interactive applications, as well as an overview of the techniques presented in the rest of the document.

1.1 Problem Overview

Because of the complexities of the human body and our sensitivity to human motion quality, synthesizing quality human motion under any set of circumstances is difficult. But the problem

becomes even more difficult when placed within the confines of an interactive application. In particular, interactive applications need methods that have each of the following characteristics:

1. **Efficient Synthesis** Any method for generating human motion at runtime in an interactive application must be able to synthesize each frame of motion at consistent, interactive speeds. A user should never see a dip in frame rate because it took longer than normal to generate a frame of motion.

2. **Efficient Data Storage** Because interactive applications must meet data storage limitations, it is important that representations used for human motion generation use little storage space.

3. **Low Latency or Response Time** At runtime, interactive applications must provide quick feedback to the user; the time between when a user requests an action and when the system actually renders the motion associated with that action should be predictable. A user should always know whether a character has “heard” his or her request.

4. **Accurate Motion Generation** Since a user issues commands in interactive applications, it is important that the motions synthesized at runtime in response to those commands closely meet the requests. For instance, if a user commands a character to punch at a target, the motion displayed by the system should show the character punching at the target as closely as possible.

5. **Visual Quality** Ideally, any motion generated for an interactive application should appear as if it were generated in the same way as the original example motions. While the quality of a motion is partly a subjective characteristic, some aspects of quality can be evaluated. For instance, it is important that generated motions be continuous. Motions cannot look as if they are segmented in time or across body parts. Distracting artifacts, such as knee-popping and jitters, should be avoided to add to the appearance of realism.

6. **Automated Authoring** To help promote easy experimentation and save important man-hours, any method for authoring human motion for an interactive application should not

	Example-Based Research	In Practice
Efficient Synthesis	No	Yes
Efficient Storage	No	Yes
Low Latency	No	Yes
Motion Accuracy	Yes	No
Visual Quality	Yes	No
Automated Authoring	Yes	No

Table 1.1 Table comparing existing motion synthesis methods against the required characteristics of interactive applications.

require extensive setup or hand-tweaking. Ideally, the computer should help alleviate any tedious tasks through automation.

Existing methods for synthesizing human motion fall short of meeting all of the requirements of interactive applications. These existing methods can generally be divided into two categories: methods used in practice in video games or training simulations and example-based motion synthesis methods developed primarily at academic institutions. In general, the methods used in practice are efficient but fail to meet quality requirements, while the existing example-based synthesis techniques produce high-quality results but are often too inefficient to be used at runtime. Table 1.1 summarizes these generalizations. Chapter 2 goes into more detail on how existing example-based methods and methods used in practice fail to meet the needs of interactive applications.

The goal of my work is to provide insight into how to develop example-based synthesis methods that are efficient enough to be used in interactive applications but can still synthesize the high-quality, accurate motions characteristic of these techniques.

1.2 Technical Solutions Overview

This section describes in detail the individual technical problems I have addressed with my work and how I have solved these problems.

Figure 1.3 A motion generated by splicing the upper body of a person carrying a heavy box with the lower body of a person walking in a curve.

1.2.1 Motion Splicing

Often the problem of character control is divided into locomotion and action. For example, a character can carry a box, punch, carry a suitcase, or wave, independent of whether it is walking straight, walking in a curve, walking up some stairs, jogging, or just moving from one foot to the other. Interactive applications that allow many different actions and kinds of locomotion must be able to generate motions for every possible *combination* of locomotion and action. Synthesizing these motions using current example-based methods requires capturing examples of the character performing all possible combinations. If n locomotion examples are needed and there are m actions that the character can perform, nm examples would need to be collected all together. It would be attractive if a character's upper-body action and lower-body locomotion could be captured independently, requiring only $n+m$ examples. However, this independent treatment requires a method for *splicing* the separate pieces together. For example, to create a motion of someone stepping up onto a platform while carrying a cup, an application could splice the lower body of a character stepping up onto a platform with the upper body of someone carrying a cup.

I have developed a simple and efficient technique for synthesizing high-fidelity motions by attaching, or *splicing*, the upper-body action of one motion example to the lower-body locomotion of another. Existing splicing algorithms do little more than copy degrees of freedom (DOFs) from one motion onto another. This naïve DOF replacement can produce unrealistic results because it

ignores both physical and stylistic correlations between various joints in the body. My approach uses spatial and temporal relationships found within the example motions to retain the overall posture of the upper-body action while adding secondary motion details appropriate to the timing and configuration of the lower body. By decoupling upper-body action from lower-body locomotion, my motion synthesis technique allows example motions to be captured independently and later combined to create new, natural-looking motions.

To illustrate my splicing technique, I give a concrete example. Imagine that we have motions of a character walking in a curved path and of a character walking in a straight line while carrying a heavy box. Using motion splicing, one can generate a motion of the character carrying a heavy box down a curved path. This motion will display many essential characteristics of the original box-carrying motion. For instance, the arms will stay the same distance apart throughout the motion, and the upper body will lean back slightly because of the weight of the box. The motion will also exhibit characteristics that are a direct result of the correlations between the upper body and lower body. For example, the shoulders will rotate and bob slightly with the stride of the lower body, and the entire motion will lean in towards the turn. In particular, note that splicing produces motions with correct physical effects without explicitly modeling physics; these physical effects are transferred because the original example motions encode them. In contrast, naïve DOF replacement produces a motion where the upper body leans forward and wobbles unpredictably in relation to the lower body.

1.2.2 Gaze Control

Humans often convey their intentions through subtle cues before performing actions. In particular, the head of a person often moves to fixate on a point before acting. I call these head cues *gaze*, and they are a natural way to convey goals. Unfortunately, virtual human characters often lack gaze cues. Sometimes video game developers do adjust a character's head facing in order to convey characteristics like "aiming" direction, but the motions employed often look unrealistic as they are based on simple inverse kinematics methods, such as just twisting the character's neck.

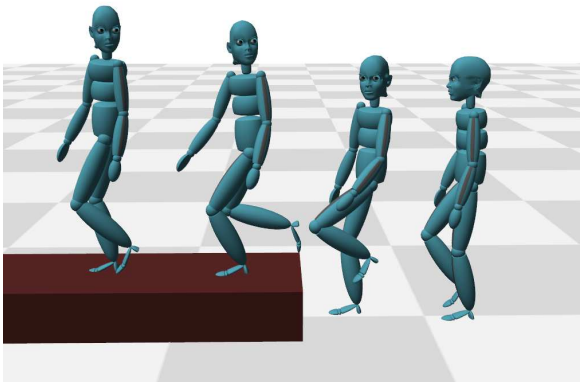


Figure 1.4 A motion generated by applying my gaze control model to adjust this character's gaze towards the left.

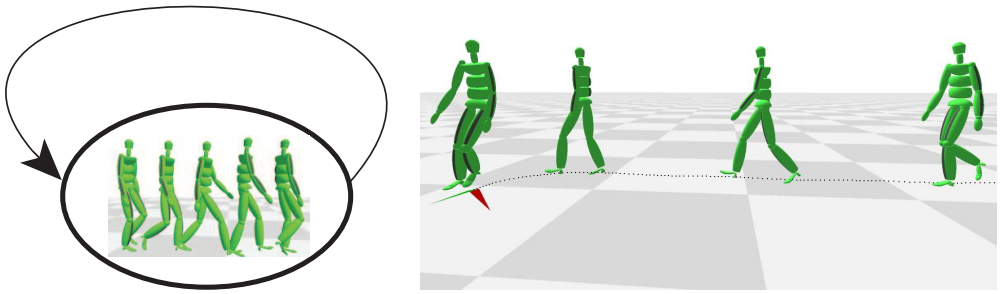
Furthermore, the way an individual adjusts their gaze depends heavily on their personal movement style. It is not sufficient to find a single way to adjust gaze for every character.

I approach this problem from an example-based perspective. If one could determine how to adjust the gaze of a character using motion-captured examples, it would be possible to generate accurate, quality motion that includes gaze cues. I represent the gaze style of a character using a low-dimensional representation of gaze change motions that is informed by existing biological research on how humans adjust their gaze. The model uses raw motion capture data of a person changing their gaze to build a parametric gaze map that can be used at runtime to quickly control the gaze of a character.

Interactive gaze control can be used to create a character that reacts faster to its environment. For example, a character could look at interesting objects as they get near them and look along the direction of travel when moving, all without appearing unnatural. One nice characteristic of interactive characters with controllable gaze cues is that the motion of the head can lead the motion of the character. For instance, when a character is standing still and a user requests that the character turn to the right, a realistic motion takes time to begin moving the character in the requested direction, which can cause a user to be unsure of whether the character has received the request. On the other hand, if the character begins turning his/her head towards the right, the user knows that the character is reacting. The head motion lends a sense of responsiveness to the character because it mimics how humans react in similar situations. See Figure 1.4 for an example of my gaze model being used to make a digital character look towards the left. Unlike the results garnered using a simple head twisting technique, my results exhibit natural motion characteristics such as overshoot and asymmetry.

1.2.3 Parametric Motion Graphs

For interactive applications, it is not only important to quickly generate a motion that meets specified constraints but also to transition between different motions rapidly and smoothly. One downside to using an example-based motion synthesis method is that for any complex application it would require a large number of example motions in order to synthesize all possible user-requested



(a) (b)

Figure 1.5 (a) A parametric motion graph for walking through an environment with smooth turns. (b) An interactively controlled character using the walking graph. The character is in the process of turning around in order to walk in the user requested travel direction depicted by the arrow on the ground.

streams of motion. But an interactive application must be able to find and transition between motion clips quickly in order to minimize latency. So, the problem faced by example-based motion synthesis researchers is one of organization. How can we represent all possible transitions between example motions in a highly-structured way in order to facilitate fast decision-making while not sacrificing the quality, accuracy, or ease of authoring gained by using an example-based synthesis approach?

To tackle this problem, I have developed a highly structured, example-based motion synthesis data structure called a *parametric motion graph*. A parametric motion graph describes possible ways to generate seamless streams of motion by concatenating short motion clips generated through parametric synthesis. Parametric synthesis allows accurate generation of any motion from an entire space of motions, parameterized by a continuously valued parameter. For example, parametric synthesis can generate motions of a person picking up an item from any location on a shelf. While neither seamless motion concatenation nor parametric synthesis is a new idea, by combining both synthesis techniques, parametric motion graphs can provide accurate control through parametric synthesis and can generate long streams of high-fidelity motion without visible seams using linear blend transitions.

To illustrate the utility of parametric motion graphs, consider a concrete example. I can create a character that can be directed through an environment with continuous steering controls. Using an existing blending-based parametric synthesis method, I first build a parametric motion space of a person walking at different curvatures for two steps. Next, I quickly build a parametric motion graph from this motion space using the algorithm presented in this thesis. The resulting graph contains a single node, representing the parametric walking motion space, and a single edge that starts and ends at this node (see Figure 1.5a). This edge describes how to transition from the end of a generated walking clip to any generated walking clip in a subspace of the motion space. This simple structure organizes many motions in a way that allows efficient character control at runtime. By translating a user's desired travel direction into desired curvature requests, the parametric motion graph can be used to synthesize a continuous stream of walking motion that reacts to a user's commands. This stream of motion will be smooth, will run at interactive rates, and will only

contain high-fidelity transitions between clips of motion. Figure 1.5b shows a screen capture of an interactive walking character using parametric motion graphs. Unlike other techniques that can create interactive walking characters, my technique requires little authoring effort, is capable of accurate motion generation, and works with a wide range of different motions. For instance, once I have an interactive walking character, it is easy to create a character that locomotes by running or cartwheeling simply by building a parametric motion space of running or cartwheeling motions, also parameterized by curvature.

Chapter 2

Related Work

This chapter reviews research related to motion synthesis for interactive applications. The chapter is divided into two parts. The first part covers work primarily concerned with synthesizing motion clips, while the second discusses work on generating motion streams that consist of many different kinds of actions. These two problems are closely related, so some related work addresses a bit of both problems.

2.1 Motion Clip Synthesis

Researchers have studied a number of different ways to synthesize individual motion clips. This section reviews the most relevant work in this area.

2.1.1 Synthesis by Manual Keyframing

One of the first methods employed by practitioners for synthesizing digital motion clips was keyframing. Keyframing is the process of manually specifying the pose of the character at key moments in time. These poses, called *key frames*, are then smoothly interpolated by the computer to produce the motion in-between. The idea of keyframing comes directly from the method traditional animators use to produce animation [JT81]; the principle animator in traditional animation draws keyframes of a scene, and then a secondary animator goes through and fills in the in between frames.

Due in part to its grounding in the traditional art world, keyframing is still a useful method for synthesizing motions. It gives an artist detailed control over the animations that are produced.

But keyframing can be a time-consuming process. Because interpolating poses is not likely to produce realistic looking motion, artists must often add additional keyframes when interpolation fails. In the end, a keyframed motion of high quality usually consists of a large number of keyframes. The matter is further complicated by the fact that properly timing keyframed motion can be difficult [TM04].

Despite the difficulties associated with keyframing, keyframing will remain a viable source of quality animation in the future. My work seeks not to replace manual keyframing but to limit the number of motions that would need to be created by an artist, either through keyframing or motion capture. I wish to shed light on how to use the power of the computer to synthesize new high-quality animations from a database of examples.

2.1.2 Parametric Motion Synthesis

One useful approach for generating new motions is parametric synthesis, or the set of techniques that map motion parameters to motion, allowing the generation of an entire space of motions simply by supplying the relevant parameters (see Section 3.5 for a more detailed description of parametric synthesis). I divide existing parametric synthesis methods into two groups: those that use a procedural, algorithmic approach to generate motions with particular parameters and those that blend together motions from a motion database in order to produce new motions with specific parameter values. There has been a considerable amount of work in both areas of parametric synthesis, and some methods combine procedural methods with blending-based methods. My own work draws from existing work in both forms of parametric synthesis, but especially blending-based parametric synthesis, which is used directly for gaze control (see Chapter 5) and in parametric motion graphs (see Chapter 6). I review these methods for parametric synthesis next.

2.1.2.1 Procedural Parametric Synthesis

Some work on parametric synthesis focuses on developing procedural methods for generating specific parametric motion spaces. Early on, Perlin [Per95] showed the utility of using simple, yet efficient, methods combined with noise to generate rhythmic motions. Later, in the *Improv*

system, Perlin and Goldberg [PG96] provided a mechanism for layering procedural methods for motion generation to author whole new controllable actions, such as smiling. Yet it can be difficult to capture the subtle nuances of real human motion using this type of simple motion generation. Thus, simple methods for procedural parametric synthesis are only useful in a small number of constrained cases.

Other methods for procedural parametric synthesis model the motions of the human body using physically-based techniques. Hodgins et al. developed physically-based controllers for generating running, bicycling, and vaulting motions [HWBO95, HW98]. The technique uses proportional derivative servos to control motion during different stages of an action and a hand-tailored state machine to navigate through these stages. Similarly, controllers were introduced by Faloutsos et al. [FvdPT01b] for balance preservation and fall recovery; by Wooten and Hodgins for leaping, tumbling, landing, and balancing motions in [WH00]; by Brunderlin and Calvert for walking [BC89] and running [BC96]; and by Laszlo et al. [LvdPF96] for balancing and walking.

While these physically-based motion synthesis methods work well for motions that are highly governed by physics, such as the ones listed above, controllers for different types of motion are time-consuming to produce and often do not generalize to new types of motion. Some researchers have looked at ways to adapt existing controllers under constrained circumstances [FvdPT01a, HP97, LvdPF96]. Yet, these techniques have been limited thus far in generalizing physically-based controllers beyond specific motion classes, especially if motion quality is a factor. One other drawback to these physical models is that the motions produced often lack the interesting details we see in everyday motion, some of which is stylistic in nature [NF02]. But physical motion synthesis has the advantage that once a model is built that describes how to synthesize a particular motion, the model requires little storage space to generate motions at runtime. Like other example-based techniques, blending-based parametric synthesis, as described in Section 2.1.2.2, suffers from the need to store all of the example motions in memory in order to synthesize new motions.

2.1.2.2 Blending-Based Parametric Synthesis

Motion interpolation methods [BW95], or ways to generate motions that are *in between* other motions, have led to the building of continuous, parameterized spaces of motions using a set of examples from that space. For example, blending-based parametric synthesis can generate motions of a person punching towards any location within an enclosed area by analyzing and blending example motions of the person punching to different locations in the space. Perlin [Per95] examined ways to generate new motions by interpolating and blending between procedurally defined example motions. Wiley and Hahn [WH97] used multilinear interpolation to blend original example motions for parametric synthesis. Similarly, Rose et al. [RCB98] used radial basis functions to do parametric synthesis. Rose et al. extended their initial work in [RSC01] to better meet inverse kinematics requests by densely sampling the parameter space, resulting in synthesized motions that more accurately meet requested parameters; and Kovar and Gleicher [KG04] built upon this idea for more general motion types. In [PZ05], Pollard and Zordan focused on using blending-based parametric synthesis of hand grasping motion capture data annotated with additional physical properties, such as force, in order to tackle the difficult problem of generating hand grasping motions. And most recently, Cooper et al. [CHP07] looked at ways to actively learn a method for blending-based parametric synthesis that allows continuous control of the parameters.

One strength of blending-based parametric synthesis is that it serves as a natural mechanism for organizing many motion clips of the same type of motion into one data structure. I harness this power in the parametric motion graph data structure presented in Chapter 6. I further adapt blending-based parametric synthesis by providing a general way to transition between different parametric spaces of motions, allowing fast synthesis of controllable motion streams in realtime.

While most methods for blending-based parametric synthesis blend motions using a linear model, researchers have looked at methods for using, extending, or replacing linear-blending. Safonova and Hodgins [SH05] studied the physical correctness of linearly interpolated motions. Based on their findings, they suggest a small number of simple modifications to basic linear-blending in order to better retain physical motion properties during blending. And, in [IAF07],

Ikemoto et al. suggest ways to cache multi-way blend information for faster response in interactive applications. As a replacement for linear-blending, Mukai and Kuriyama [MK05] introduced geostatistical interpolation to the field of motion synthesis. Using a technique called universal kriging, Mukai and Kuriyama show how correlations between the dissimilar portions of two motions can be estimated, producing interpolated motions with fewer spatial interpolation artifacts, such as foot-sliding, a common artifact in linearly interpolated motions. These variations on blending have great potential to improve the results produced through linear blends. While I use basic linear blends in all of my work, it should be possible to replace or modify my method of blending to include these improvements without other modifications to my algorithms.

Often, good motion interpolation depends on warping the timing of the motions so that logical actions line up. There are a number of different timewarping methods employed by researchers to do this alignment. Park et al. [PSS02] and Rose et al. [RCB98] both had a user hand-mark key points in the motions to be aligned, which could then be used as a reference to form a time alignment. Ashraf and Wong [AW00] expanded these techniques by locating these key motion points in a semi-automated way.

Other timewarping methods use dynamic timewarping based on frame-to-frame similarity to produce a more optimal time alignment between motions. These dynamic timewarping methods include those of Bruderlin and Williams [BW95], Dontcheva et al. [DYP03], Hsu et al. [HPP05], and Kovar and Gleicher [KG03]. I use the dynamic timewarping method of Kovar and Gleicher [KG03] extensively in my work in order to locate correspondences and align correlations between two motions. See Section 3.4 for more information on how I align motions in time.

2.1.3 Layered Motion Synthesis

While most work on motion synthesis focuses on producing all of the degrees of freedom (DOFs) of a character simultaneously, there has been some work on creating character motions by layering the DOFs of different motions.

To my knowledge, splicing upper-body and lower-body motions together is common practice in the video game industry. For example, to make a character hold a gun while running, the upper

body of a character holding a gun is spliced onto the lower body of a character running. While I am aware of no published descriptions of the methods employed, my experience suggests that splicing is usually accomplished by simply swapping data between two motions, henceforth referred to as naïve DOF replacement. Unfortunately, unless the original motions are carefully tailored, naïve DOF replacement will often produce unnatural results because it does not respect the natural correlations between the upper body and lower body. Even in simple cases, such as transferring the upper body of a walking motion to a different walking motion, naïve DOF replacement can cause disturbing visual artifacts. My method presented in Chapter 4 for splicing upper-body actions onto lower-body locomotions is developed specifically to address these problems.

Naïve DOF replacement was also used by Perlin [Per95] and Perlin and Goldberg [PG96] to layer procedurally-defined motions into composite actions. However, the goal of this work was to produce scriptable characters that could flexibly interact with each other and with human participants, rather than to synthesize high-quality motion clips. More recently, Ikemoto and Forsyth used naïve DOF replacement to transplant pieces of motion [IF04]. Using a classifier to evaluate the results, they added "human" looking motions to a motion database. Ikemoto and Forsyth acknowledge the failures of naïve DOF replacement in their work by using a classifier to guarantee that the results look correct, but their goal of expanding a motion database offline does not depend on having a reliable method for splicing. It is important that any method for splicing motions at runtime be reliable; algorithms for assessing the quality of the produced motions are too slow to be used at runtime, and a strategy of resplicing when a splice fails will not produce spliced motions in a predictable amount of time, which may cause efficiency issues. In contrast, the method I present in Chapter 4 for splicing upper-body actions with lower-body locomotion reliably splices in a predictable amount of time.

To build a representation of how the DOFs of a set of "base" actions (such as walking straight forward, standing, or sitting) are affected by the addition of an auxiliary action (such as throwing), Al-Ghreimil and Hahn [AGH03] captured examples of each base action with and without the auxiliary action and computed the differences. Joint trajectories that varied little in these difference motions were replaced with an average, yielding a more compact encoding. The result was a set

of joint trajectories that could be added on to new instances of a known base motion, e.g., a different actor walking straight forward. It is unclear how well this method generalizes to different base motions, such as turning or walking in a curve. Yet, the Al-Ghreimil and Hahn method does show that decoupling methods can drastically decrease motion data requirements. My methods for decoupling upper-body action from locomotion (Chapter 4) and gaze from full body motion (Chapter 5) take inspiration from this work, while at the same time focuses more on the reliability of the decoupling methods.

Pullen and Bregler [PB00] introduced a method for producing new motions that are statistical variations of a base motion using models that explicitly deal with body correlations. Later, they extended these ideas to add detail to partially keyframed motion. Given keyframed values of a small number of a character's DOFs, Pullen and Bregler [PB02] identified segments of a motion data set where these DOFs were similar at user-specified frequency bands. These segments of motion data were then used to add higher-frequency content to the keyframed DOFs and to fill in the DOFs that were not keyframed. My own work on motion decoupling differs from Pullen and Bregler's work in that I seek to *control parameters independently* during runtime motion synthesis. Their work focuses on synthesizing motion offline based on partial specifications of its DOFs by adding detail to sparse keyframes of a small number of DOFs. Yet, like my work, [PB00] shows the utility of synthesizing motions in layers.

Similar to Pullen and Bregler's method, Lee et al. [LBB02] developed a statistically based method for adding eye motion detail to existing full body motions. Lee et al. found that motions where a character's eyes were animated using their model appeared more lifelike than those without animated eyes or with randomly animated eyes. This finding is important to my own work on gaze control (Chapter 5) as it points out the importance of eyes and gaze in making human motion appear more lifelike.

One algorithm that is similar to my work on motion splicing is the work of Majkowska et al. for splicing hand motion onto full body motion [MZF06]. This hand splicing algorithm uses temporal and spatial relationships to perform a good splice, just as I do in Chapter 4 to produce splices of upper-body action and lower-body locomotion. However, there are many differences between our

two problems other than simply my focus on splicing a different part of the body. Majkowska et al.'s goal was to be able to decouple the capture of hand animation from full-body animation, allowing better accuracy in captured hand motion and the ability to re-use that motion for different actors performing similar full-body motions. Instead, my goal is to allow locomotion and action to be "mixed-and-matched" during motion synthesis. Our differing goals lead to variations in our algorithms that are tailored towards our individual purposes, but the overall approach of temporal and spatial alignment is used for both.

2.1.4 Constraining Motion

One common method for producing motions that meet constraints is to directly force a base motion to meet the constraints. A large body of work exists for locating and enforcing constraints on human motion. Bindiganavale and Badler [BB98] developed a method for identifying contact constraints with objects in an environment by looking at acceleration zero-crossings for end-effectors. Bodik [Bod00] developed a simple and effective method for identifying the important class of constraints known as footplants based on joint velocities. More recently, Ikemoto et al. [IAF06] introduced a special-purpose algorithm for identifying footplant constraints. This work uses a k-nearest neighbor classifier that is trained with labeled motion-captured examples in a semi-supervised way. These methods provide an automated way to identify some classes of constraints in an environment.

Motion displacement maps have also been used by researchers to help enforce kinematic motion constraints. Witkin and Popovic [WP95] first introduced a smooth variant of displacement maps called motion warping for adjusting motion to meet constraints. Gleicher [Gle98] extended this work for retargeting existing motions to new characters . Later, Gleicher [Gle01] adjusted these displacement mapping ideas to allow a user to interactively change the path that a character follows. More recently, Wang et al. [WDAC06] used displacement maps to control how "animated" a character looks. These displacement maps are built using the second derivative of the original motion, resulting in "animated" motions with exaggerated characteristics. Methods for editing motions based on displacement maps allow easy motion modifications by introducing

smooth changes. I extend displacement maps in Chapter 5 by building parametric spaces of spinal displacement maps that are used to decouple gaze control from overall full body motion.

Inverse kinematics has also been a popular method for enforcing motion constraints. Shin et al. [SLGS01] introduced a method for online retargeting of human motion onto computer characters using inverse kinematics methods and the locations of important objects in a scene. Monzani et al. [MBBT00] also used inverse kinematics to make smooth adjustments to motions so that they meet specified constraints. In [SKF07], Shapiro et al. used randomized path planning combined with inverse kinematics in order to adjust motions that are subject to many complex contraints. Lee and Shin [LS99] employed a method based on both hierarchical displacement maps and inverse kinematics for adjusting a motion to specified constraints.

While often fast to compute, inverse kinematics is inherently ill-conditioned, making it challenging to find a single correct answer [Mac90], though the issues involved are well understood making inverse kinematics a common method for practical motion editing. Traditional inverse-kinematics methods also often do not produce high-quality motion if the constraints to be enforced are far from the base motion pose. Again, these quality issues arise from the fact that there are usually many pose configurations and motions that will enforce a constraint but few of them look "human". My work will allow the generation of more accurate motions at runtime, allowing these simple inverse kinematics methods to be applied where they work best - when the base motion is close to the actual requested motion.

Inverse kinematics and dynamics have also been used for special-purpose motion cleanup algorithms. Ko and Badler [KB96] used inverse dynamics to adjust walking motions so a character would stay in balance and have reasonable joint torques. This approach is well-suited for producing effects like leaning to compensate for a heavy weight that is being carried, but it is less appropriate for generating motions that are primarily driven by stylistic preferences (e.g., there are many ways to carry a light tray while staying in balance). Kovar et al. [KSG02] developed a special-purpose inverse kinematics method for adjusting a motion to exactly meet footplant constraints at runtime. Again, these special-purpose inverse kinematics methods work well because they are applied to base motions that are already close to the desired motion.

Recently, there has been interest in example-based inverse kinematics [GMHP04, YKH04]. The idea behind this work is to learn a method for inverse kinematics from a database of example motions by projecting the example motions into a low-dimensional space. This low-dimensional version of the motion database can be used as a lookup table in order to find plausible motions that enforce requested constraints at runtime. In a variation of these example-based inverse kinematics techniques, Hsu et al. [HGP04] controlled coupled motions by building index tables based on a low-dimensional representation of one of the motions; for instance, Hsu et al. generated the motion of a person following a lead dancer using a lookup table based on the motion of the lead dancer. These example-based inverse kinematics methods are motivated by the fact that human motion is complex, making it hard to modify motions in a realistic way without a reliable example. This dependence on examples for realism and focus on motion clip organization for interactive control further illustrates my thesis.

Other researchers have used the principles of physics to enforce physical properties on simulated motion. The work of Shin et al. [SKG03] used the physical laws of ballistic motion and zero moment point constraints to make a motion more physically plausible. In particular, Shin et al.'s method is capable of adjusting the motion of a person walking on a flat plane to the motion of a person walking up a steep hill by adjusting the motion based on it's zero moment point. Tak et al. [TSK02] used Kalman filters to enforce physical constraints on motions and adjust for balance. Liu and Popovic [LP02] also used the laws of physics to correct ballistic motions. This work, however, differed from the work of others by focusing on the cleanup of keyframed ballistic motion (see Section 2.1.1). The idea is to allow a user to keyframe a small number of poses in a jumping motion. Then, using simple motion interpolation combined with the laws of physics, a new physically-plausible ballistic motion can be generated. These physically based cleanup methods work well for making motions look more realistic when there are obvious physical properties of the motion that should hold, but the methods are limited when it comes to producing motions with stylistic characteristics.

The physical and biological sciences have also been used for other motion synthesis tasks. In [KP06], Kry and Pai presented a new motion capture method by which contact forces are

recorded along with the position of points in space. This method of capture allows new motions to be synthesized using the physically plausible model generated from the capture process. Metoyer et al. [MZH+07] used the science of psychology to develop a physically based model of dynamic responses for impacts to the head and upper body. The model allows motions to react quickly and realistically by anticipating the collision and adjusting physically at impact. This approach of building motion models based on observations made within the psychology community is an approach I also take in my work on gaze control, presented in Chapter 5.

2.2 Motion Stream Synthesis

While the previous section reviewed methods primarily concerned with synthesizing individual clips of motion, this section reviews existing research on synthesizing long, continuous streams of motion. The section starts with a discussion on generating motion streams by transitioning between motion clips. This discussion is followed by a review of work that uses an unstructured graph to represent possible transitions within a database of motions. The rest of the section describes techniques for using deliberately structured graphs for better efficiency during motion synthesis, using data structures specifically designed for locomotion synthesis, and representing motion streams using statistical models.

2.2.1 Motion Transitions

One common method for producing a long motion stream is to append motion clips together using a transition. A transition is a segment of motion that seamlessly attaches two motions to form a single, longer motion. Early work in this area focused on ways to transition between two motion clips in a realistic way [RGBC96, LvdP96]. One common method is to linearly blend from one motion to another over a small window of frames. This type of transition is generally referred to as a *linear-blend transition*. In my work, I use linear-blend transitions to piece together motion clips in a continuous way. See Section 3.3 for more details on how I perform transitions.

Because of their common usage in the community, linear-blend transitions have been a topic of study over the past couple of decades. Several researchers have looked at ways to identify

an optimal transition point at which to make a transition between two motions [WB03, LCR+02, KGP02, AF02]. I use the method first presented in [KGP02] to perform comparisons between motions (see Section 3.2).

It might be possible to improve transitions between appended motions by using, extending, or replacing linear-blending using the methods presented in Section 2.1.2.2. Again, while I use basic linear blend transitions for my work, it should be possible to replace or modify my method for appending motions together to include these improvements without any complex modifications to my algorithms.

2.2.2 Unstructured Motion Graphs

Over the last decade, techniques for producing motion transitions were extended to directly represent possible transitions between motions in a motion collection using graph structures [AF02, KGP02, LCR+02, AFO03, KPS03, SNI06]. As with Video Textures [SSSE00], the key to generating long streams of motion is being able to locate frames of motion in a motion database that look similar enough to be used as transition points between different motion clips. These motion graph techniques focus on using automated comparison methods to easily author the transition graphs. But these motion graphs are unwieldy for use in interactive applications because they lack a high-level structure. Without costly global search methods, it is hard to find motions to transition to that meet specified constraints. More recent work in unstructured motion graphs have looked at ways to blend paths within the graph [SH07] in order to produce more accurate motions. But, again, this approach requires a costly search, making the technique infeasible for online applications.

For applications with a limited motion database or unbounded computation time, unstructured motion graphs can be a simple and effective method for organizing motions clips. Because of the quality of the results produced using these unstructured motion graphs and the ease of authoring new graphs for character control, my work builds heavily upon these methods. However, I seek to organize the connections in motion graphs so that global search is not necessary to synthesize new motion streams. This idea is the foundation of my work on parametric motion graphs (Chapter 6).

In general, it is hard to evaluate unstructured graphs or determine the range of possible motions that can be generated using the graph. Reitsma and Pollard [RP04] tackled this problem to study the capabilities of locomotion graphs by embedding these graphs in an environment. The embedding showed the range of possible locations and orientations a character could reach using the specified motion graph. Later Reitsma and Pollard extended their work in [RP07] to be more efficient as well as to evaluate characteristics of other motion types. By organizing motion clips in a structured motion graph, such as in the parametric motion graph I present in Chapter 6, it may be possible to evaluate the capabilities of the graph structure more efficiently. Specifically, the work of Treuille et al. [TLP07] showed how to efficiently determine the capabilities of a parametric motion space during motion capture time. This ability to analyze the usefulness of parametric motion spaces could be useful for practical use of parametric motion graphs.

Researchers have augmented unstructured motion transition graphs by precomputing properties of these graphs to aid in interactive character control. Lee and Lee used reinforcement learning and dynamic programming to determine how a character can best perform a desired action for any given character state [LL04]. This data is stored in a lookup table that can be used at runtime to efficiently generate motions that meet user requests. In a similar way, Srinivasan et al. [SMM05] precomputed *mobility maps* from a motion transition graph to aid in the runtime generation of locomotion. And Lau and Kuffner [LK06] also used precomputed motion graph search trees to aid in the character navigation task.

These augmented motion graphs are still only able to represent transitions between a discrete number of motions and become unwieldy as the number of motions in the graph becomes large. Furthermore, these techniques do not directly address the problem that the underlying motion transition graph is unstructured; they instead provide methods for dealing with the unstructured graph in a more efficient way. I suggest using a method that explicitly organizes motions for the required task, as illustrated in Chapter 6. Other methods that explicitly organize the motions in a motion graph are reviewed next.

2.2.3 Structured Motion Graphs

The gaming industry often uses hand-generated graph structures called move trees to represent ways to transition between clips of motion [MBC01]. Because move trees are constructed for easy interactive character control, they have a deliberate structure that aids in quickly choosing motions based on user requests. This structure facilitates controlling characters in realtime, but this ability comes only after many man hours of work. And because a move tree only represents a discrete number of motion clips, the accuracy of the motions it produces is limited by the granularity of these motions. To increase the accuracy, more and more motions must be added, and each addition to the move tree takes many man hours.

Some researchers have built structured graphs specifically designed for the task at hand. For instance, Lee et al. [LCL06] allowed motion building blocks to be placed in an environment, forming a graph of connectable motions within the environment itself. In [SDO+04], Stone et al. built structured motion graphs for controlling speech using the grammar of the language. This approach of embedding a motion graph in a specific task space is limited in that each method can only be used for the specific task. Furthermore, it might not be possible to build a natural embedding; and even if the embedding is possible, the resulting motions are often limited in quality since a natural embedding does not necessarily correspond to natural transitions between motion clips. Yet, these task-specific, structured motion graphs illustrate the utility of organization for interactive motion control.

McCann and Pollard [MP07] structure their graphs in a tabular way using a statistical model of human behavior to optimize for quick transitions after user requests in interactive applications. They showed how their method allows fast motion stream generation with high responsiveness, but often these quick responses come at the expense of quality. And, similar to motion trees, McCann and Pollard's transition graph can only represent a discrete number of motion clips, resulting in a lack of accuracy.

One automated transition graph technique that deliberately seeks to build highly-structured graphs for general, accurate, quality, interactive motion control is Snap-Together Motion [GSKJ03]. The key of the technique is to identify poses that appear many times in the original motion clips.

These poses become “hub” nodes in a graph, and edges between these nodes correspond to the individual motion clips that can transition between these poses. This technique does a good job of automating the process of building controllable characters, but again the structure can get unwieldy as the number of motion variations grows large. And since Snap-Together Motion can only represent a discrete number of motions, motion synthesis accuracy is dependent on the number of motion examples. Sung et al. [SKG05] targeted this limitation in order to synthesize accurate motion clips for agents in a crowd by using the Snap-Together Motion graph to synthesize a motion that nearly meets requested goals and then using editing operations on these motions to get exact results.

Accuracy is not the only limitation of Snap-Together Motion. The technique also makes no attempt to group logically similar motions, an approach that I use to help add more structure to my parametric motion graph control structures (see Chapter 6). Additionally, snap-together motion graphs are unable to represent continuously changing transition points and ranges within a single type of motion, such as the case where a character can transition to other walking motions with similar curvature to its current one. A snap-together motion graph must use more than one “hub” node in order to capture a portion of the complexity of these shifting transition possibilities.

A recent extension to Snap-Together Motion groups similar clips that connect the same “hub” nodes into parametric edges, forming a new structure called a fat graph [SO06]. Fat graphs and parametric motion graphs are similar in that they combine parametric synthesis and synthesis-by-concatenation to provide interactive control. But parametric motion graphs have a number of other benefits, including the ability to produce more natural looking motion transitions. These benefits are reviewed in more detail in Section 6.2.3.

2.2.4 Controllable Locomotion

Other researchers have studied ways to synthesize controllable locomotion in realtime (see [MFCD99] for a thorough survey of this work through 1999). Sun and Metaxas [SM01] used a procedurally defined parametric walking motion to generate streams of motion that adjust to user-defined curvature requests and uneven terrain. Park et al. [PSS02] introduced a similar technique

for generating locomotion, such as walking and jogging, using a graph built from parametric motion spaces parameterized by curvature. Yin et al. [YLvdP07] used a physics-based controller to produce continuous steams of locomotion that react to many different environmental changes.

Example-based graph locomotion control methods also exist. Kwon et al. [KS05] grouped motion segments based on footstep patterns. Transitions between these groups are encoded in a hierarchical motion graph, where the coarsest level of the graph describes general transition patterns while more detailed levels capture the cyclic nature of locomotion. Choi et al [CLS03] built graph structures called roadmaps based on footstep patterns embedded in the environment. Using a combination of path planning and displacement maps, they used these structures to quickly plan and synthesize locomotion. Pettre and Laumond [PL06] built a blending-based parametric motion space from a set of locomotion example motions to aid in locomotion control. These techniques for generating controllable locomotion illustrate the utility of using structure to produce controllable motion in realtime. The drawback of these methods is that they are designed specifically to work on locomotion only. I will address general motion control for interactive environments with my work on parametric motion graphs (see Chapter 6).

2.2.5 Statistical Motion Graphs

A number of other researchers have developed statistics-based graph structures where nodes represent statistical variations of motion primitives and edges show how to connect these primitives [LWS02, GJH01, Bow00, BH00]. Molina-Tanco and Hilton [TH00] produced graphs that were also based on a statistical model but the final motion clips used in the simulations were only original motion clips or blended versions of the clips.

These statistically-based synthesis methods focus on producing variable motion, not on controlling the motion produced. Because of this, they are difficult to use to control motion. Also, because the statistics of human motion are not necessarily modeled by the statistics of human poses, the motions produced using statistical motion graphs often lack realism.

Chapter 3

Background

The work in this dissertation builds directly on existing work in the area of motion synthesis. This chapter describes in more detail some of the techniques and concepts unique to motion synthesis that are used extensively throughout the rest of the document. In particular, this chapter briefly describes how motions in this dissertation are represented, the metric used to measure the similarity between two pieces of motion, the method used to append two motions together, the process for aligning two motions in time, and the blending-based parametric synthesis method employed. It should be noted that none of the methods presented in this chapter are novel. All of the described techniques are well described elsewhere. They are reviewed here to provide explanation and motivation for the methods I build upon and to introduce important terms and notation that are used throughout the rest of this document. Please refer to the original papers on these techniques for more details.

3.1 Motion Representation

The raw data collected by a standard motion capture system consists of the 3D locations of a set of points that are rigidly attached to the actor's body. Because of the ease of animating a character using a set of hierarchically organized joints, called a *skeleton*, the raw data from a motion capture shoot usually goes through the process of being skeletonized (see [BRRP97, OBBH00, ZH03] for more information on how this process is accomplished).

While the work presented in Chapter 5 uses raw motion capture data directly, I primarily represent the human body as a rigid-body skeleton. Each joint of the skeleton has exactly one parent

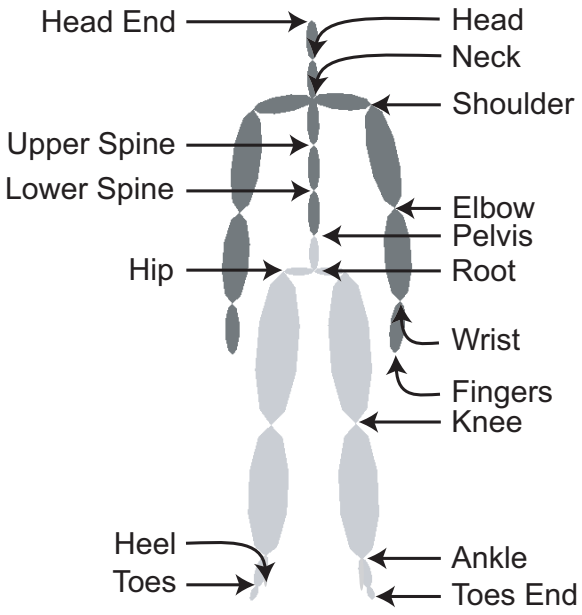


Figure 3.1 Standard Skeleton Hierarchy

joint, with the exception of the *root* joint that has no parent. A skeleton can be defined by its joint hierarchy and initial pose, or the offset of each joint from its parent in local coordinates. Figure 3.1 shows the standard skeleton hierarchy used in all of the experiments presented in this dissertation. Note that the articulated skeleton can only control the body motion of the character; the skeleton does not include joints that represent fingers, toes, or facial features. In general, this dissertation is only concerned with a character's overall body motion, with the notable exception of eye movement as discussed in Chapter 5.

Given a character's skeleton, a motion is defined as a continuous function over time:

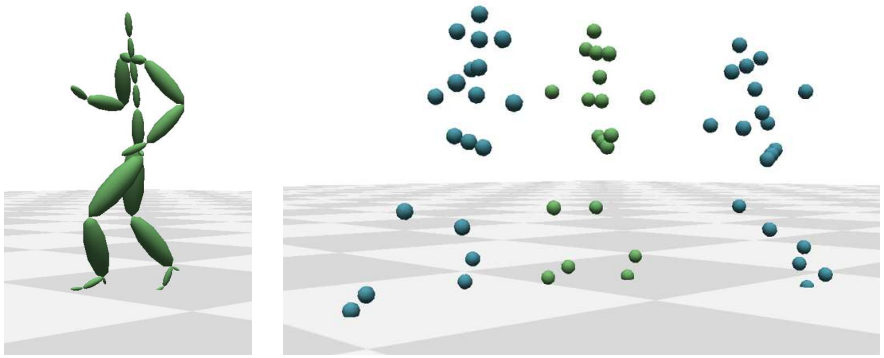
$$\mathbf{M}(t) = \{\mathbf{p}(t), \mathbf{q}_1(t), \ldots, \mathbf{q}_k(t)\} \quad [3.1]$$

where $\mathbf{p}$ is the position of the root with respect to the origin and $\mathbf{q}_j$ is the relative orientation of the j^{th} joint with respect to its parent. In the case of the root, which does not have a parent joint, the orientation is denoted with respect to the global coordinate system at the origin.

In practice, the continuous function $\mathbf{M}(t)$ is represented as a set of samples, or *frames*, taken at regular time increments, $\mathbf{M}(t_1), \ldots, \mathbf{M}(t_n)$. Values of $\mathbf{M}$ in between frames are computed by using linear interpolation of the root position and spherical linear interpolation on the joint orientations represented as unit quaternions [Sho85]. While there are a number of other ways to represent rotations, unit quaternions are particularly well-suited for interpolation (see [Gra98]), making them the rotation representation of choice for many motion researchers.

3.2 Motion Similarity

Much of my work depends on computing the similarity between two frames of motion, or, conversely, to compute the difference between two frames of motion. One could compute the difference between two frames of motion, $\mathbf{M}(t)$ and $\mathbf{M}'(t')$, by directly comparing the sampled motion vectors, $\{\mathbf{p}(t), \mathbf{q}_1(t), \ldots, \mathbf{q}_k(t)\}$ and $\{\mathbf{p}'(t'), \mathbf{q}'_1(t'), \ldots, \mathbf{q}'_k(t')\}$, a method employed by others studying human motion [LZWP03] as well as by researchers in other domains with high-dimensional spaces [BBK01]. There are two problems with this type of direct comparison:



a **b**

Figure 3.2 Point Cloud Representation of a Frame of Motion. (a) shows a frame of running motion in skeletal form. (b) shows the same frame of motion in point cloud form. Notice how the point cloud representation contains information about where each of the joints of the skeleton are located in 3D space on the frame in question as well as the frames surrounding it.

1. It fails to account for differences introduced solely due to a motion's global orientation in the environment.

2. It does not take into account differences between the dynamics of the two motions.

Thus, I compute the difference between two frames of motion, $\mathbf{D}(\mathbf{M}(t), \mathbf{M}'(t'))$, using a metric originally introduced by Kovar et al. [KGP02]. I chose to use this algorithm for two reasons, each of which addresses one of the two problems associated with direct motion vector comparison:

1. The metric is invariant to translations along the ground plane and rotations about the up-axis.

2. The metric can take into account joint velocities and accelerations using finite differences.

The metric works by first representing each frame of motion as a cloud of points, henceforth called a *point cloud*. The points of a point cloud correspond to the locations of relevant skeletal joints over a small window of time surrounding the frame. Figure 3.2 shows an example of a point cloud generated from a frame of running motion. Using the following closed-form solution from [KGP02], I can compute the rotation about the vertical axis, θ, and translation along the floor

plane, (x_0, z_0), that best aligns corresponding points, $\mathbf{p_i}$ and $\mathbf{p'_i}$, in the two point clouds.

$$\theta = \arctan \frac{\sum_i w_i(x_i z'_i - x'_i z_i) - \frac{1}{\sum_i w_i}(\overline{x}\overline{z'} - \overline{x'}\overline{z})}{\sum_i w_i(x_i x'_i + z_i z'_i) - \frac{1}{\sum_i w_i}(\overline{x}\overline{x'} + \overline{z}\overline{z'})} \quad [3.2]$$

$$x_0 = \frac{1}{\sum_i w_i}(\overline{x} - \overline{x'}\cos(\theta) - \overline{z'}\sin\theta) \quad [3.3]$$

$$z_0 = \frac{1}{\sum_i w_i}(\overline{z} + \overline{x'}\sin(\theta) - \overline{z'}\cos\theta) \quad [3.4]$$

where i is an index over the number of points in each cloud, w_i is a weighting term for point $\mathbf{p_i}$ in the point cloud, x_i and z_i are the x and z coordinates of point $\mathbf{p_i}$, and all barred terms correspond to the weighted sum of the barred variables over the index i.

Using this optimal alignment, the distance between the two frames is computed as:

$$\mathbf{D}(\mathbf{M}(t), \mathbf{M}'(t')) = \sum_i (\mathbf{p_i} - \mathbf{\Phi}_{\theta,\mathbf{x_0},\mathbf{z_0}}(\mathbf{p'_i}))^\mathbf{2} \quad [3.5]$$

where the function $\Phi_{\theta,x_0,z_0}(\mathbf{p})$ applies the optimal point cloud alignment transform, $\{\theta, x_0, z_0\}$, to the point $\mathbf{p}$.

Depending on the length of the window over which the point cloud is built, this distance metric can implicitly take into account relative joint positions, joint velocities, and joint accelerations when measuring similarity. Refer to [KGP02] for a more detailed description of this point cloud distance metric.

3.3 Transitioning Between Two Motions

It is often useful to append two motions together with a transition. Unfortunately, appending motions together by simply appending motion frame vectors is likely to cause discontinuities in the motion. Instead, as described in Section 2.2.1, it is common to append two motions together by using a linear blend transition between the two motions over a small window of frames centered at the transition point.

Appending a motion, $\mathbf{M}_2$, to another motion, $\mathbf{M}_1$, using a linear blend consists of three steps:

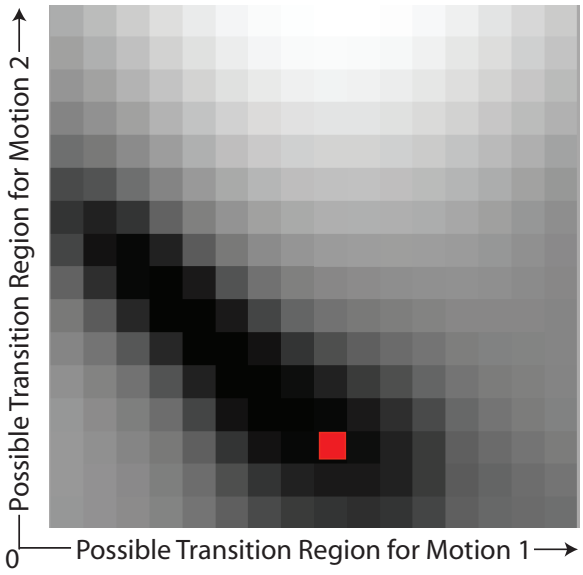


Figure 3.3 A distance grid. Darker regions denote greater similarity between frames. The light red dot marks the optimal transition point.

1. **Choosing a transition point between the two motions.** While a transition point can be chosen arbitrarily between the two motions, linear blend transitioning works best if the motions look as similar as possible. For this reason, my methods choose a transition point between the two motions by locating the point where the two motions look the most similar to each other over the possible transition region. The point cloud distance metric presented in Section 3.2 is used to rate the similarity of different frame pairs to determine the transition point. My method for finding the transition point starts by calculating the distance between every pair of frames in the possible transition regions, forming a grid. The pair of frames corresponding to the grid cell with the minimum distance value, (t_o^1, t_o^2), is called the optimal transition point. Figure 3.3 shows an example of this distance grid computation between two motions.

2. **Aligning the two motions at the transition point.** Because the global position and orientation differs between $\mathbf{M}_1$ and $\mathbf{M}_2$, it is necessary to align $\mathbf{M}_2(t_o^2)$ to $\mathbf{M}_1(t_o^1)$. This can be done by applying the optimal alignment transform associated with $\mathbf{D}(\mathbf{M}_1(t_o^1), \mathbf{M}_2(t_o^2))$, Φ_{θ,x_0,z_0}, to the motion $\mathbf{M}_2$.

3. **Synthesizing the appended motion through blending.** The final step necessary to append motion $\mathbf{M}_2$ to motion $\mathbf{M}_1$ is to blend the motions over time to produce the final appended motion, $\mathbf{M}_{1+2}$:

$$\mathbf{M}_{1+2}(t) = \left\{ \begin{array}{ll} \mathbf{M}_1(t) & \text{if } t < t_o^1 - \frac{w}{2} \\ \mathbf{M}_2(t_o^2 + t - t_o^1) & \text{if } t > t_o^1 + \frac{w}{2} \\ (1 - \alpha(t)) * \mathbf{M}_1(t) + \alpha(t) * \mathbf{M}_2(t_o^2 + t - t_o^1) & \text{otherwise} \end{array} \right\}$$

$$\alpha(t) = \frac{t - t_o^1}{w}$$

   This blending process is illustrated in Figure 3.4.

I chose to use this method for appending two motions together using a linear blend transition because it is a simple method. The limitation is that linear blend transitions only work reliably when the motions that are being appended are already close to one another in terms of the similarity

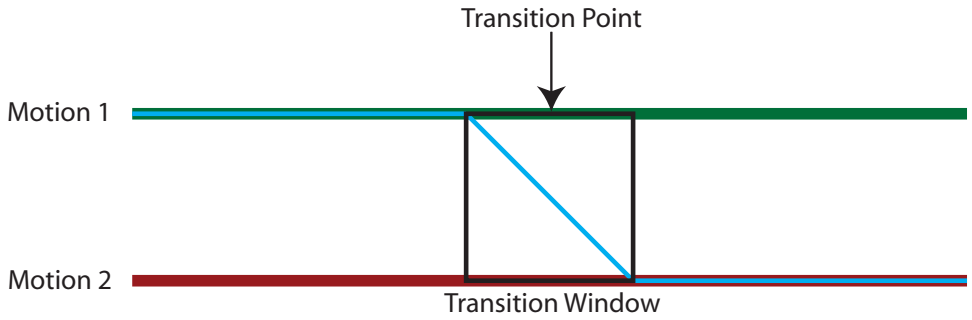


Figure 3.4 Appending two motions through linear blend transitioning. The green bar (top) and red bar (bottom) correspond to the original two motions. The thinner, light blue bar shows the weighting for the blend of the motions over time. Notice how the weights go from being fully on motion 1 to fully on motion 2 over the transition window centered at the transition point.

metric presented in Section 3.2. Rather than developing a more complex method for appending or transitioning between motions, I use this simple method but ensure that I only transition between motions with sufficient similarity.

3.4 Motion Time Alignment

Two motions, $\mathbf{M}_1$ and $\mathbf{M}_2$, may be logically similar in that they represent different motions of the same action. For instance, both motions might be instances of a person walking, following the same footstep pattern. Yet, these logically similar motions might have very different timing details. For instance, each step in $\mathbf{M}_1$ might take twice as long as each step in $\mathbf{M}_2$. For some applications, it is desirable to find a time alignment between logically similar motions such that frames in one motion can be mapped to time-corresponding frames in the other motion. This continuous, strictly increasing mapping is called a *time alignment curve*.

When computing a time alignment curve between two motions, $\mathbf{M}_1$ and $\mathbf{M}_2$, the goal is to find a mapping between frames of $\mathbf{M}_1$ and frames of $\mathbf{M}_2$ that minimizes the average distance between corresponding frames, where distance is computed using the algorithm in Section 3.2. First, the distance between every pair of frames is calculated, forming a grid as in Section 3.3. Then using the dynamic programming method of Kovar and Gleicher [KG03], a continuous, monotonic, and non-degenerate path is computed for every cell in the grid that connects to the lower-left corner, while minimizing the sum of its cells. This optimal path from the lower-left corner to the upper-right corner of the grid provides a discrete, monotonically increasing time alignment, as shown in Figure 3.5. To generate the final time alignment curve, a strictly increasing, endpoint-interpolating B-spline is fit to the optimal path. See [Kov04] for more information on how to increase the speed of this time alignment algorithm.

The dynamic timewarping method that I use in this dissertation to compute time alignment curves is based on the work of Kovar and Gleicher [KG03], but there are several other published methods that would work as well. In particular, the timewarping methods of Bruderlin and Williams [BW95], Dontcheva et al. [DYP03], and Hsu et al. [HPP05] also use dynamic programming to find a temporal alignment between motions. Hsu et al.'s method [DYP03] builds on

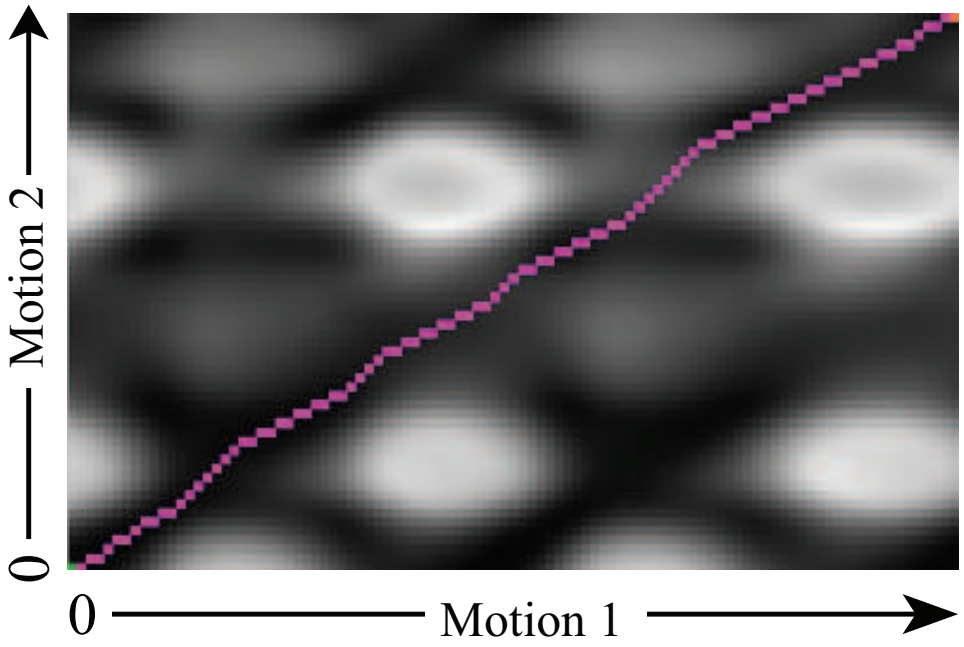


Figure 3.5 A grid depicting the difference between Motion 1 and Motion 2. Dark areas denote similarity between the corresponding frames. An optimal time alignment is shown by the path through the grid.

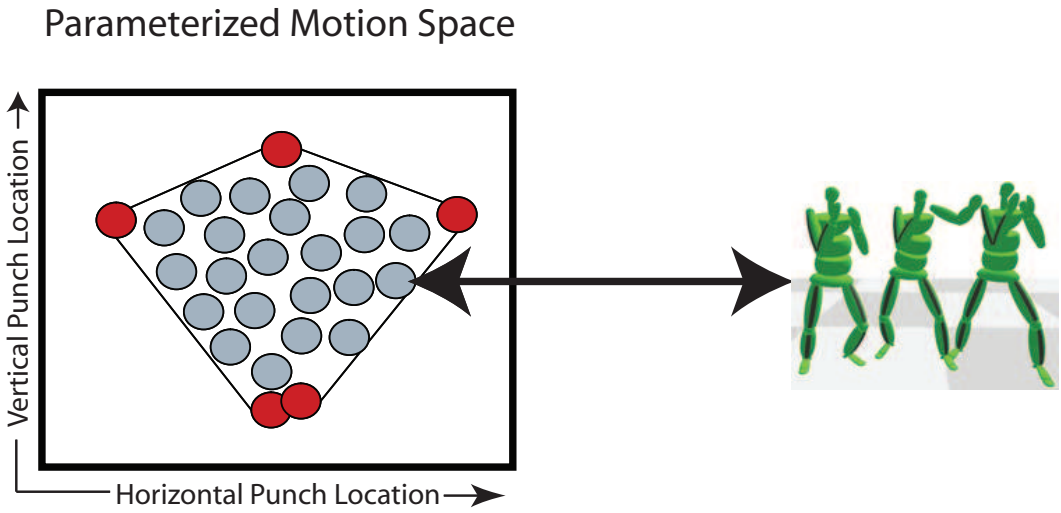


Figure 3.6 A parametric motion space representing punching motions, parameterized on the location of a punch. Each point in this parametric motion space maps to an entire punching motion, not just a single pose.

dynamic timewarping to iteratively converge on a time alignment that explicitly takes into account pose variations. This technique can be used to improve timewarping results but requires a considerable increase in computation time. A simple, greedy timewarping algorithm can also work in cases where the motions are very similar. But I have found that greedy search does not work well for many of the more complex cases of timewarping in this dissertation, and I recommend using a method based on dynamic programming.

3.5 Blending-Based Parametric Synthesis

Many different types of motion are easily described using a small number of *parameters*. For instance, a punching motion might be described by the location of a punch, or a stair climbing motion might be described by the height of the steps. It can be convenient to use these high-level parameters as a way of describing what motions are requested by a user or application. *Parametric synthesis* describes the set of techniques that can generate motions based solely on these high-level parameter vectors. The infinite set of motions that can be generated using a parametric synthesis method can be mapped to the parameter space, forming a *parametric motion space*. Each point

in a parametric motion space maps from a parameter vector to an entire motion that meets those parameters (see Figure 3.6).

One popular method for parametric synthesis is to synthesize new motions that meet requested parameters by blending together example motions from that space (see Section 2.1.2.2 for a review of these techniques). In this dissertation, I perform blending-based parametric synthesis using the method of Kovar and Gleicher [KG04]. Because Kovar and Gleicher's method is so essential to my work on gaze control (Chapter 5) and parametric motion graphs (Chapter 6), I will use this section to provide an in depth overview of their approach.

In [KG04], Kovar and Gleicher described how to:

1. *Automatically find and extract logically similar motions*, or motions where the character is performing the same basic action, from a motion database. Their technique uses an iterative method that locates motions that look similar to a query motion, where similarity is defined by the metric presented in Section 3.2, and then repeats the process on each of the similar looking motions.

2. *Map these logically similar motions to a parametric motion space*. Each of the logically similar motions are registered to each other in both time and space using methods similar to those presented in Sections 3.3 and 3.4. These registered motions become example motions in the motion parameter space by mapping the motion to its relevant parameter vector. For instance, the algorithm might identify the location in space where the character punches. Figure 3.7a shows these example motions mapped in parametric motion space for a space of punching motions.

3. *Sample blends from the space to build a parametric motion*. New logically similar motions can be generated by blending together the base example motions. Yet, because of the non-linearity of human motion, the parameter vectors associated with these new blended motions are unlikely to be a similarly proportioned blend of the base example parameter vectors. So, Kovar and Gleicher sampled the set of motions that can be blended together from these base

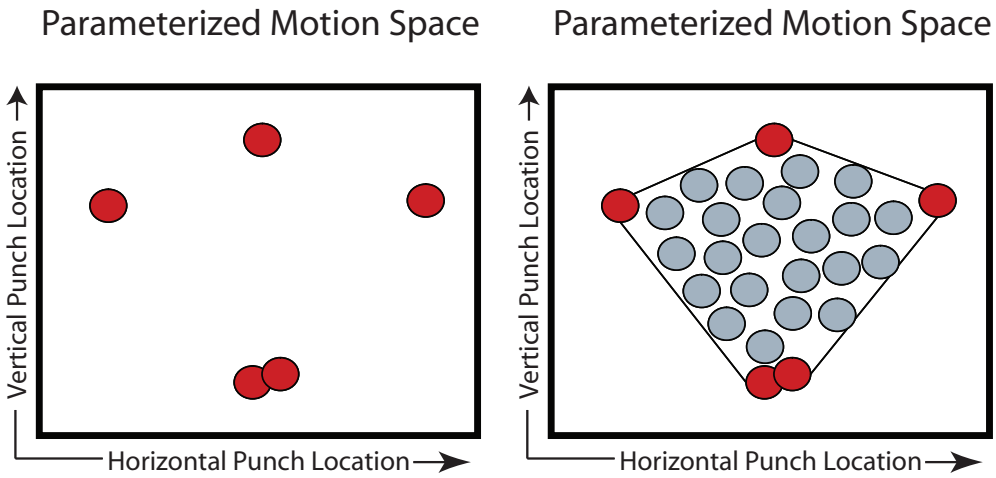


Figure 3.7 A sampled parametric motion space of punching motions, parameterized on the location where a character punches. (a) Logically similar punching motions from a motion database are mapped to the parametric motion space (shown with dark, red circles) (b) Blends between the example punching motions are sampled, producing additional data points (shown with light, gray circles) in parametric motion space.

example motions. Each of these sampled blends is then analyzed, just as the original examples were, to produce an associated parameter vector. The blend information then becomes a new data point in parametric motion space at this computed parameter vector. Figure 3.7b shows these blend samples mapped in the punching parametric motion space.

4. *Use the samples from the parametric motion space to synthesize new motions that accurately meet user-requested parameter vectors*. Once the samples in parametric motion space are sufficiently close together, k-nearest neighbor interpolation is used to synthesize new motions that meet specified parameter vectors. Other methods for blending-based parametric synthesis use a linear fit model to perform this scattered data interpolation instead of k-nearest neighbor interpolation. But k-nearest neighbor interpolation has a number of advantages: it constrains interpolation weights to reasonable, positive values, resulting in more realistic-looking motion; it is computationally efficient for large data sets and does not require a costly optimization to calculate; and it projects all outlier parameter vector requests back into the space enclosed by the original example motions.

I chose to use the method of Kovar and Gleicher to perform parametric synthesis because it produces high-quality results, allows for quick experimentation with many different types of motion, provides a simple and efficient method for producing motion clips at runtime, and results in parameteric motion spaces that are *smooth*. A parameteric motion space is considered smooth if small changes in the input parameters produce small changes in the generated motion. This smoothness property is important to my work on parametric motion graphs in Chapter 6.

Chapter 4

Splicing Upper-body Actions with Locomotion

As described in Section 1.2.1, interactive applications often divide character control into locomotion and action. Yet example-based motion synthesis fails to treat locomotion and action independently. This inability to decouple locomotion and action makes using an example-based approach to motion synthesis in interactive applications infeasible since it results in a combinatorial number of required example motions. This problem is of particular interest to the video game industry as many video games require locomotion and action to be decoupled during motion synthesis.

Existing methods for decoupling locomotion and action during motion synthesis do little to account for the natural correlations within the body. The motion of the upper body and lower body can be be highly correlated – for example when a person walks, the swing phase of the right arm is tightly coupled to that of the left leg – and these correlations are an important part of many motions [PB00, MZF06]. To complicate the matter, these natural correlations not only stem from physical needs, such as balance preservation, but also from harder-to-characterize properties associated with the stylistic form of human motion.

In this chapter, I describe a method for splicing a character's upper body action onto its lower body locomotion in a manner that preserves the fidelity of the original source motions. In light of the complicated and subtle form of the relationships I wish to preserve, I adopt an example-based approach that produces natural motions by identifying and enforcing temporal and spatial relationships between different parts of the body. The simple and efficient method synthesizes spliced motions that appropriately preserve correlations. For instance, Figure 4.1 shows the results of splicing the upper body of a person holding a cup onto the lower body of a person stepping

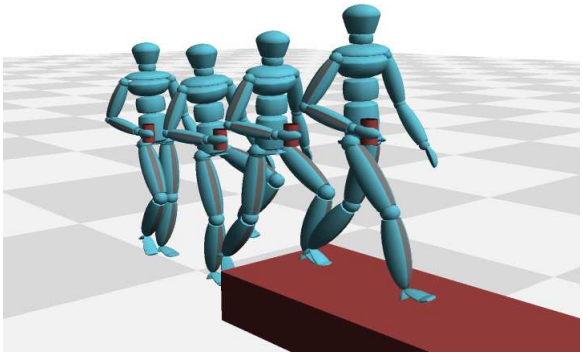


Figure 4.1 A motion generated by splicing the upper body of a person carrying a cup with the lower body of a person stepping up onto a platform.

up onto a platform. My algorithm allows the upper-body action to be decoupled from the method of locomotion yet preserves important details, such as the shifting of the character's weight as he pulls himself up onto the platform, the character's cup-carrying posture, and the forward lean of the upper body prior to the step up.

The rest of this chapter will describe the results from a study that looks at natural correlations in motion (Section 4.1), detail my algorithm for splicing upper-body actions with locomotion (Section 4.2), provide some additional insights into using my algorithm at runtime (Section 4.3), provide the results of some experiments involving motion splicing (Section 4.4), and conclude with a general discussion of the technique, its advantages, and its disadvantages (Section 4.5).

4.1 Correlation Study

As defined by the American Heritage Dictionary [Ame04], a correlation is

> [a] casual, complementary, parallel, or reciprocal relationship ... between two comparable entities

In other words, there is a correlation between two things if one changes in a relatable when the other changes.

The key challenge to splicing one part of a motion onto another is that the body is highly correlated. For example, when a person does jumping jacks, her arms move up at the same time

that her legs spread out. This correlation between the upper-body motion and the lower-body motion is part of what makes the jumping jack motion recognizable.

This chapter is concerned with the splicing of an upper-body action onto a lower-body locomotion. For this specific, yet important, motion splicing problem, parts of the body that seem unrelated are actually closely tied together. While working on my algorithm for splicing upper-body actions with lower-body locomotion, I analyzed these correlations in locomotion examples.

My method for analyzing the upper body and lower body correlations in a human locomotion example consists of five steps:

1. Perform forward kinematics on the motion under analysis.

2. Plot the global orientations of each upper-body joint and the global x-rotation of the hip joints (i.e., the part of the hip's rotation which controls the forward and backward motion of the leg) with respect to the root.

3. Fit a polynomial to each data line, producing a trend curve.

4. Identify correlations by finding trend curves that are in phase with the hip trend curves or 180 degrees out of phase with the hip trend curves.

5. Repeat steps 3, 4, and 5 on the local orientations of the upper body joints and the x-rotation of the hip joints.

For the results presented here, a computer was used to perform forward kinematics, plot data lines, and fit trend curves. The final correlation identification was done manually.

I analyzed the correlations for a number of locomotion examples; I will discuss here some of the correlations that appeared consistently in all of my examples. The global analysis of locomotion shows a consistent bobbing of the entire upper body (e.g., the upper body leans forward and backward in time with the motion of the legs). This correlation is a lot stronger in motions where the upper body is highly constrained, for example, when a walking character carries a heavy box (see Figure 4.2).

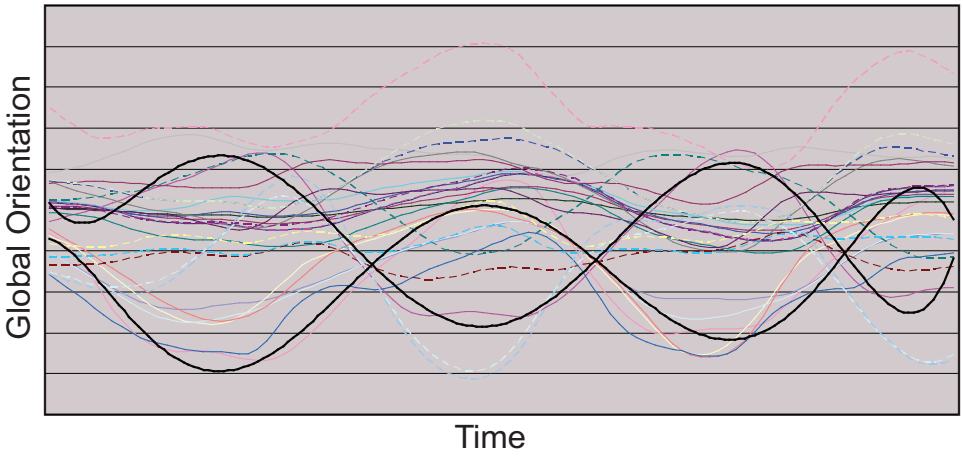


Figure 4.2 Plots of the global orientations of correlated joints in a walking motion as determined by my method. This graph shows the large number of joints that are globally correlated in a simple walking motion.

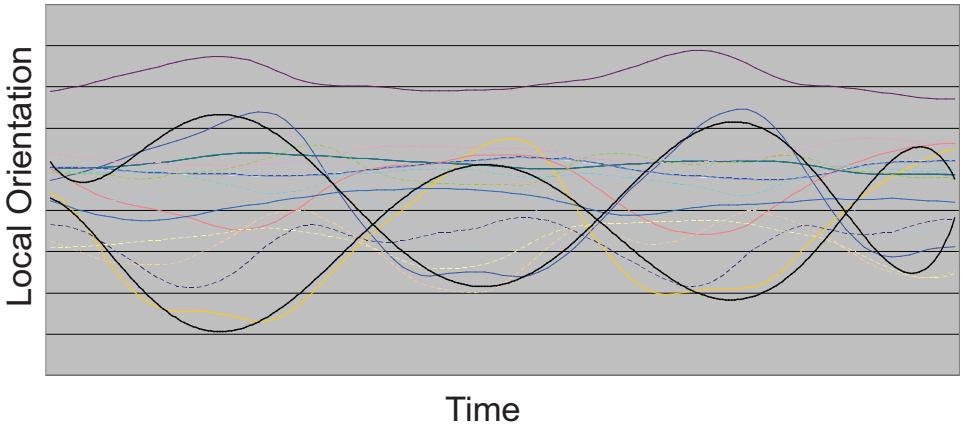


Figure 4.3 Plots of the local orientations of correlated joints in a walking motion as determined by my method. This graph shows the large number of joints that are locally correlated in a simple walking motion.

The local orientation analysis consistently showed that as the character moves, the upper body twists through the spine causing the relationship between the shoulders and the hips to change. And, in general, the head also needs to twist in the opposite direction to allow the character to continue looking in the original direction of travel (see Figure 4.3).

Many other correlations show themselves using this type of analysis. The number and degree of the correlations varies greatly depending on the exact motion, but the importance of correlations to natural human locomotion becomes obvious when looking at this data. The importance of these correlations is the driving motivation behind my algorithm for quality splicing of upper-body action and locomotion presented in the next section.

4.2 Splicing Algorithm

My technique for splicing the upper-body action of one motion onto the lower-body locomotion of another is motivated by two key observations. First, a captured locomotion example encodes temporal and spatial relationships between the upper body and lower body that can be used for splicing; in other words, if I have one example of a character performing an action while moving around, I can transfer that action onto a different example of a character moving around. For this reason, I require that all captured examples of upper-body actions be performed while locomoting. In particular, this requirement provides a basis for temporal correlations, which are extraordinarily important during locomotion (see Section 4.1).

The second key observation is that changes made within the upper body or the lower body of a motion can affect a viewer's perception of the action or locomotion. For example, when carrying a heavy box, a person's arms move little relative to the torso, and any editing operation that causes additional arm movement will incorrectly make the box appear lighter. Since I do not know what the upper body or lower body is doing, only that the upper body is performing an action and the lower body is locomoting, I cannot safely make any changes within the upper body or lower body. This leads me to restrict the kinds of changes that can be made during splicing. Specifically, I only allow a temporal alignment and a per-frame rigid transformation at the attachment point.

This helps retain the meaning of the upper-body and lower-body motions while allowing better correlation between the two pieces.

4.2.1 A Technical Overview

Motion splicing is concerned with attaching the upper body of one locomotion example, $\mathbf{M}_U$, onto the lower body of another locomotion example, $\mathbf{M}_L$, yielding a spliced motion, $\mathbf{M}_S$. My goal is to preserve the relative locations of the joints within the upper body of $\mathbf{M}_U$ and within the lower body of $\mathbf{M}_L$ while still exhibiting details related to the correlations between the two halves. To do this, I construct $\mathbf{M}_S$ by identifying and enforcing temporal and spatial relationships within the motions. The process consists of three stages:

1. **Time Alignment (Section 4.2.2).** Using the configuration of the lower bodies of the two example motions, find a *time alignment curve*, $\lambda(t)$, that relates corresponding frames of $\mathbf{M}_L$ and $\mathbf{M}_U$. In particular, $\mathbf{M}_L(t_i)$ is at the same phase of the locomotion cycle as $\mathbf{M}_U(\lambda(t_i))$. The time alignment curve is used to identify temporal correlations between the example motions.

2. **Spatial Alignment (Section 4.2.3).** For each frame $\mathbf{M}_L(t_i)$, find a rotation about the pelvis that best aligns the upper body of $\mathbf{M}_U(\lambda(t_i))$ with the upper body of $\mathbf{M}_L(t_i)$. Intuitively, this alignment correlates the upper-body motion of $\mathbf{M}_U$ with the lower-body motion of $\mathbf{M}_L$ by using the upper-body motion of $\mathbf{M}_L$ as a reference.

3. **Posture Transfer (Section 4.2.4).** The *posture* of a motion is the global relationship between the shoulders and hips of the character. A potentially undesirable side effect of spatial alignment is that it will alter the posture of $\mathbf{M}_U$ so that it roughly matches the posture of $\mathbf{M}_L$. Thus, the final step is to apply a posture transfer technique to retain the posture of $\mathbf{M}_U$ while still preserving the high-frequency details necessary to correlate it with $\mathbf{M}_L$'s lower body.

The result is a spliced motion, $\mathbf{M}_S$, of the form

$$\mathbf{M}_S = \{\mathbf{p}_L(t), \mathbf{Q}_L^l(t), \mathbf{q}_P^S(t), \mathbf{Q}_U^u(\lambda(t))\} \qquad [4.1]$$

where the only quantities that need to be computed are the time alignment curve, $\lambda(t)$, and the pelvis orientations, $\mathbf{q}_P^S(t)$. Note in particular that, up to timewarping, the parameters of every joint except for the pelvis come directly from the captured motions $\mathbf{M}_L$ and $\mathbf{M}_U$.

Each step of the motion splicing algorithm uses a different part of the original motion examples as a reference, thereby transferring and preserving important correlations: time alignment uses the lower body of $\mathbf{M}_L$; spatial alignment uses the upper body of $\mathbf{M}_L$; and posture transfer uses the posture of $\mathbf{M}_U$.

The remainder of this section expands on each step of this algorithm.

4.2.2 Time Alignment

Before the upper body of $\mathbf{M}_U$ can be attached to the lower body of $\mathbf{M}_L$, it must be warped in time in order to retain important temporal correlations. For example, without timewarping the arms may appear to swing out of time with the legs. Because naïve DOF replacement (see Section 2.1.3) does not consider the temporal alignment of natural locomotion correlations, it can easily cause motions to be spliced out of sync. In practice, example motions are carefully hand-tailored to avoid timing issues, but this process is extraordinarily time-consuming and greatly limits the motions that can be spliced together without temporal correlation problems. Figure 4.4 shows a graph of a motion that has been spliced together without any timing adjustments.

Timewarping is accomplished by constructing a time alignment curve, $\lambda(t)$, that maps frames of $\mathbf{M}_L$ to corresponding frames of $\mathbf{M}_U$. Since the characters in both example motions are locomoting, λ can be built by using the similarity of the lower-body motions of $\mathbf{M}_L$ and $\mathbf{M}_U$ to build the alignment curve.

Specifically, in this step of the motion splicing algorithm, the goal is to find a mapping between frames of $\mathbf{M}_L$ and frames of $\mathbf{M}_U$ that minimizes the average distance between corresponding frames. The distance between frames is calculated as described in Section 3.2, using the positions of the root and both knees to form the point clouds. To compute the optimal time alignment, first assume that the initial frames $\mathbf{M}_L(t_0)$ and $\mathbf{M}_U(t_0)$ are in phase, i.e., both motions begin at the same point in the locomotion cycle. Then, use the algorithm presented in Section 3.4 to compute

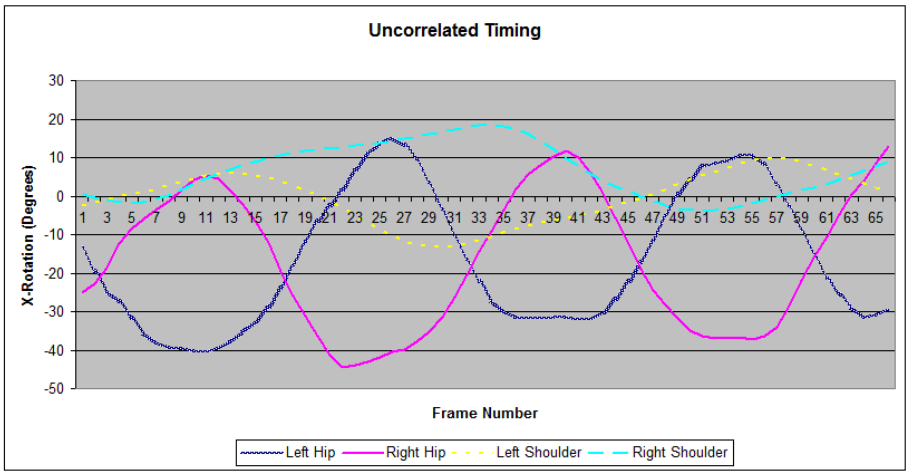


Figure 4.4 Timing difference caused by splicing two motions that have not been time aligned. Notice how the rotations of the arms are in sync and the rotations of the legs are in sync, yet the arms and legs are not correlated with each other. These correlation issues make the motion appear odd, even to someone unfamiliar with the reason why.

the optimal time alignment curve grid. Next scan the cells in the top and right edges of the grid for the path whose cells have the smallest average value. As described in Section 3.4, this optimal path defines a continuous time alignment curve for the two motions.

If the initial frames of $\mathbf{M}_L$ and $\mathbf{M}_U$ are not in phase, then crop an appropriate number of frames from the beginning of $\mathbf{M}_U$ before computing the time alignment curve. This is done by computing the distance between $\mathbf{M}_L(t_0)$ and every frame of $\mathbf{M}_U$ and cropping $\mathbf{M}_U$ at the first local minimum.

4.2.3 Spatial Alignment

Once a time alignment curve is available, the system can attach corresponding frames together. The question at this stage is what local rotation should be used for the pelvis of the new motion. One possible choice is the local orientation of the pelvis in $\mathbf{M}_U$. However, the local orientation of the pelvis is tightly coupled to that of the root. The local orientation not only controls how the upper body should turn, bend, and twist in relation to the lower body, but it also compensates for movement within the root, effectively stabilizing the upper body. Because the root orientations of $\mathbf{M}_U$ and $\mathbf{M}_L$ are not identical, simply copying the local pelvis orientation will destroy the coordination of the movement. This coupling is one of the main reasons why naïve DOF replacement often produces wobbly looking upper body motions (see Section 2.1.3). Figure 4.5 illustrates the wobbling issues associated with naïve DOF replacement.

Another possibility for the local rotation of the pelvis is to preserve the joint's original global orientation, but this can also be problematic. For instance, if a motion of a character walking east is spliced with one of a character walking west, the upper body will face backwards in the result (see Figure 4.6).

I instead approach this problem as one of spatial alignment. Just as the lower bodies of the two example motions were used for time alignment, the upper bodies can be used for spatial alignment. Specifically, for each pair of corresponding frames, the goal is to find a local pelvis orientation that best aligns the shoulders and spine of $\mathbf{M}_U$ with those of $\mathbf{M}_L$. This method ensures that the upper body faces the correct general direction while also adding details to the orientation that are appropriately correlated with the motion of the lower body. For example, when a person

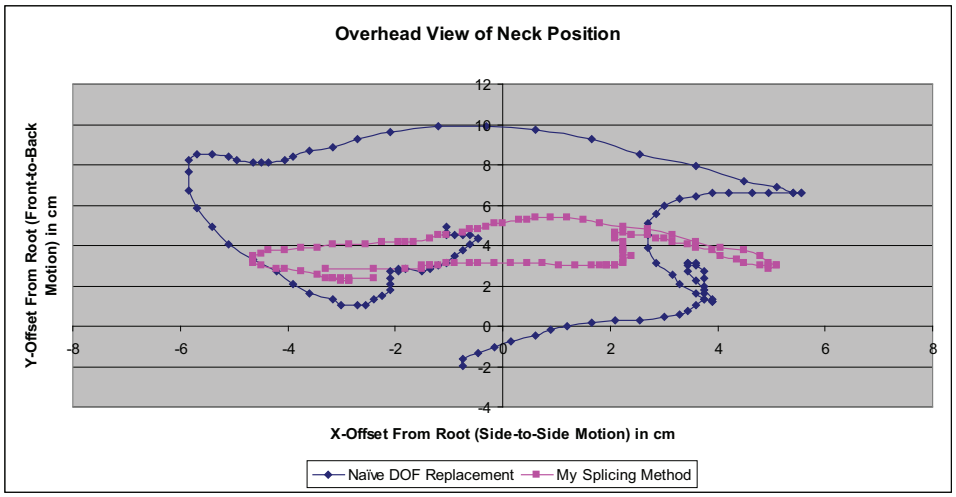


Figure 4.5 Stability comparison of naïve DOF replacement and my motion splicing algorithm. This graph shows the position of the neck in relation to the root for a motion spliced together using naïve DOF replacement and a motion spliced together using my method. Both motions were spliced from the same source data - the upper body of a person carrying a cup and the lower body of a person stepping up onto a platform. Notice how the naïve DOF replacement result moves erratically and wildly when compared with the more stable result produced using my technique.

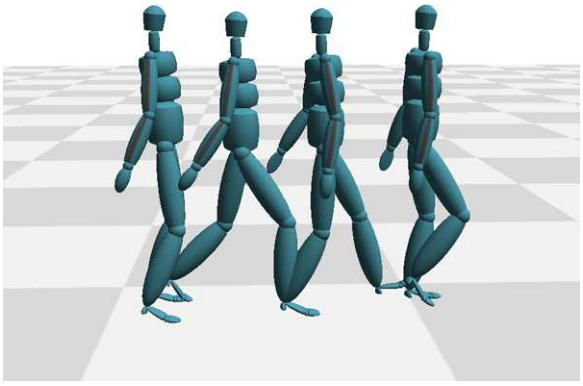


Figure 4.6 The result of splicing the upper body of a person walking west with the lower body of a person walking east using the global orientation of the characters for alignment. The result does not look natural because the upper body faces backwards on the lower body.

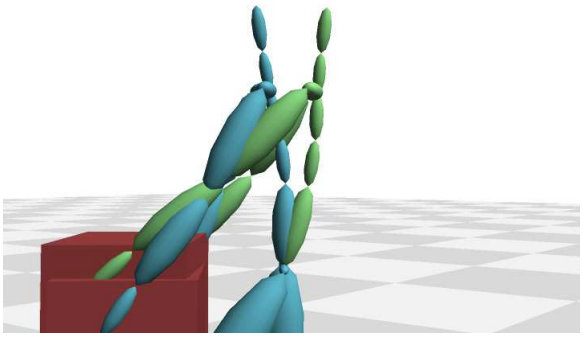


Figure 4.7 A comparison between two motions spliced from a person carrying a heavy box and a person walking in a curve. The green motion was generated using my entire technique, while the blue motion lacks the adjustments made by posture transfer. Note that the green motion leans back, balancing the weight of the heavy box. The character is rendered as a stick figure to better illustrate this difference.

moves, their upper body rotates with each step. Step size, speed, and path curvature are just a few characteristics that can affect the exact details of these rotations. By aligning the upper body of $\mathbf{M}_U$ with the upper body of $\mathbf{M}_L$, these orientation details are transferred to the spliced motion.

As with the temporal alignment step, point clouds are used to perform spatial alignment (see Section 3.2 for more information on point clouds). To spatially align the upper bodies of $\mathbf{M}_L(t_i)$ and $\mathbf{M}_U(\lambda(t_i))$, first form a point cloud for each of the two motions based on the locations of the pelvis, spinal joints, and both shoulders. Then translate and rotate the point clouds so that the coordinate systems of the pelvis coincide with the origin. Finally, using the method of Horn [Hor87], find the 3D rotation at the origin, $\mathbf{q}_A$, that minimizes the sum of squared distances between corresponding points. Then the local pelvis orientation in relation to the root of $\mathbf{M}_L$ that best aligns the upper body of $\mathbf{M}_U$ with the upper body of $\mathbf{M}_L$ is simply $\mathbf{q}_P^L * \mathbf{q}_A$, where $\mathbf{q}_P^L$ is the local pelvis orientation of $\mathbf{M}_L$.

4.2.4 Posture Transfer

For many upper-body actions, an important and necessary aspect of the motion is the overall orientation of the shoulders in relation to the hips over the duration of the motion. For example, a person carrying a heavy box leans back so as to counter the weight of the box. I call this overall

relationship between the shoulders and the hips the *posture* of the motion. The spatial alignment procedure described in Section 4.2.3 preserves subtle correlations between the upper-body and lower-body motions by tracking the movements of the torso and shoulders, but it has the potentially undesirable side-effect that it changes the motion's posture. For instance, if the upper body of the person carrying the box were to be attached to the lower body of a person walking normally, the per-frame spatial alignment would rotate the upper body forward so that it would line up better with the straight-backed upper body of the normal walking motion (see Figure 4.7).

Let $\mathbf{M}_{S'}$ be the result of splicing the upper body of $\mathbf{M}_U$ with the lower body of $\mathbf{M}_L$ using only time alignment (Section 4.2.2) and spatial alignment (Section 4.2.3). The goal of posture transfer is to replace the posture of $\mathbf{M}_{S'}$ with the posture of $\mathbf{M}_U$ while still preserving the high frequency details dictated by the lower-body motion of $\mathbf{M}_L$. The general strategy is to compute a point cloud representation (again, see Section 3.2 for more details on point clouds) for the overall posture of each motion, and then to find the 3D rotation, $\mathbf{q}_G$, that best aligns these temporally global point clouds, as was done in Section 4.2.3. This rotation specifies how $\mathbf{M}_{S'}$ should be globally adjusted in order to transfer the posture of $\mathbf{M}_U$.

To compute the point cloud representation for a single motion, form a point cloud for each frame of the motion based on the locations of both shoulders. Then translate and rotate each point cloud so that the pelvis is at the origin and the vector defined by the hips is aligned with the x-axis. The point cloud posture representation is simply the average of these aligned per-frame point clouds. Then find the 3D rotation, $\mathbf{q}_G$, that minimizes the sum of squared distances between corresponding points, again using the method of Horn [Hor87]. So, for each frame, the final local orientation of the pelvis, $\mathbf{q}_P^S$, is $\mathbf{q}_P^{S'} * \mathbf{q}_G$.

4.3 Runtime Modifications

To effectively use the methods presented in this chapter to synthesize a motion clip at runtime in an interactive application, it is necessary to apply the algorithm in a slightly different manner from that presented. In particular, the time alignment (Section 4.2.2) and posture transfer (Section 4.2.4) steps require that the motions be processed globally. Even more problematic is that the

time alignment algorithm scales poorly with the length of the motions being compared; if there are u frames in motion $\mathbf{M}_U$ and there are l frames in motion $\mathbf{M}_L$, the algorithm scales with $\mathbf{O}(ul)$. Thus, I make the following two suggestions for performing motion splicing at runtime:

1. The time alignment curve, $\lambda(t)$, should be precalculated and stored in a lookup table for easy access. This can either be done directly between every pair of possible spliced motions, or done indirectly by time aligning each motion to a baseline reference locomotion example. The latter method is overall a better approach as it is considerably faster to compute and requires only $\mathbf{O}(n)$ storage space, where n is the total number of example motions. The former method does have a slight advantage over using a baseline motion - determining the time alignment between two motions at runtime only requires a single time alignment curve lookup while the baseline motion method requires the composition of two time alignment curve lookups. Yet the storage savings garnered by using the baseline motion method far outweigh this constant time increase in computation time. For either case, these stored time alignments can be used to temporally align motions for quick splicing at runtime.

2. For each of the upper-body example motions, I also suggest computing and storing the posture of the motion prior to runtime. Again, these precomputed postures can be accessed quickly for posture transfer. The posture computations of $\mathbf{M}_{S'}$ can be calculated in one of two ways: as a running average, a method that should work well given that posture could change slightly over the course of a motion; or, since the spatial alignment step (Section 4.2.3) causes the posture of $\mathbf{M}_{S'}$ to roughly match the posture of $\mathbf{M}_L$, a precomputed posture for $\mathbf{M}_L$ can be used as an approximation of the posture of $\mathbf{M}_{S'}$.

These simple modifications to the application of the algorithm make it applicable as a direct replacement for runtime naïve DOF replacement in existing interactive applications.

4.4 Results

I tested a number of different motion combinations while developing this motion splicing technique. Here I describe in detail some of these splicing results:

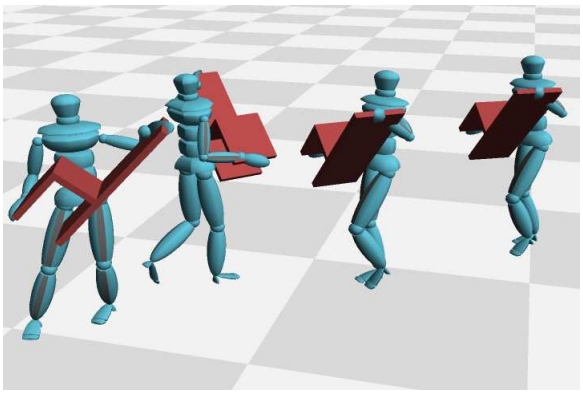
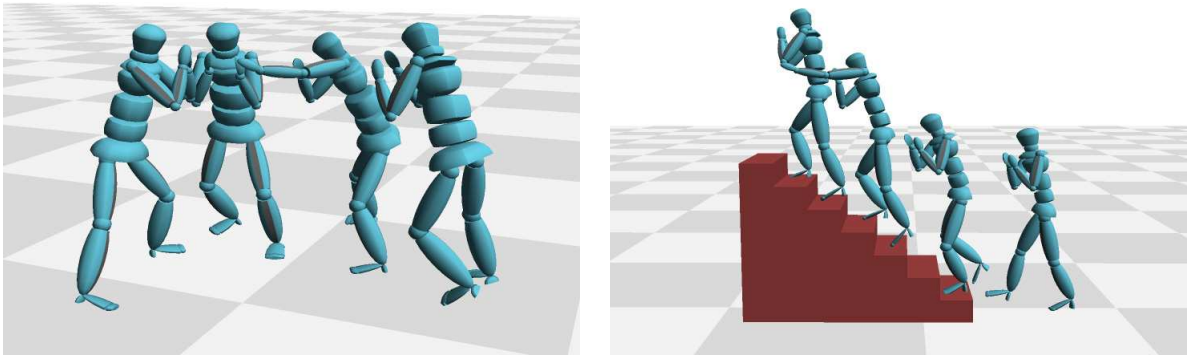


Figure 4.8 A motion generated by splicing the upper body of a person carrying a chair with the lower body of a person making a sharp 180 degree turn.



a **b**

Figure 4.9 An unusual motion splice. (a) is an original motion-captured example of a person boxing. (b) was generated by splicing the upper body of the person boxing with the lower body of a person walking up a set of stairs.

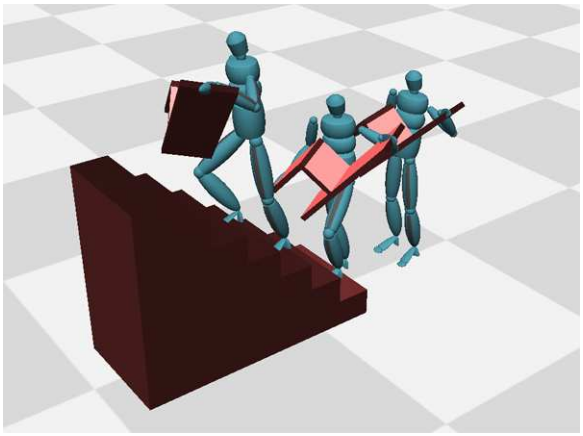


Figure 4.10 A motion generated by splicing the upper body of a person carrying a chair with the lower body of a person walking up a set of stairs.

Carrying a Box in a Curve. Figure 1.3 shows the result of splicing the upper body of a person carrying a heavy box with the lower body of a person walking along a curved path. As described in Section 1.2.1, the spliced character leans into the turn, leans back to counter the weight of the box he is carrying, and twists his upper body slightly with each step. When this result is compared with ground truth, motion-captured data of an actual person walking along a curved path while carrying a heavy box, it is difficult to distinguish one from the other; both characters twist, turn, and bend in similar ways.

Stepping-Up While Carrying Cup. Figure 4.1 shows the results of splicing the upper body of a person holding a cup onto the lower body of a person stepping up onto a platform. Important correlating details are clearly seen in the motion; the character shifts his weight as he pulls himself up onto the platform, the character retains a rigid cup-carrying posture that would keep the contents of the cup inside the cup, and the character leans forward with his upper body prior to stepping up onto the platform.

Sharp 180 with a Chair. I have found that my method works well even in cases where the two lower bodies look very different. For example, a motion of a person making a sharp 180-degree turn looks considerably different from a typical forward walking motion. Yet the principles presented in this chapter remain valid, and I have successfully spliced motions where a person walks forward while performing a variety of actions, such as carrying a chair, onto a sharp turn (see Figure 4.8).

Punching while Climbing Stairs. Figure 4.9 shows another example of a challenging splice: an upper body of a boxer is spliced onto the lower body of a person climbing a set of stairs. Notice how the lower body of the boxing motion is not a traditional locomotion example, but since the character generally moves from one foot to the other throughout the motion, the algorithm is able to correlate the upper-body punching motion with the lower-body stair climbing motion. The punch is timed correctly with the motion of the legs as the character climbs the steps.

Climbing Stair with a Chair. Figure 4.10 shows the result of splicing the upper body of a person carrying a chair with the lower body of someone climbing a set of stairs. With each step up the stairs, the character's upper body twists slightly more than it does when walking straight forward, a characteristic of step climbing motions. The character also leans forward with his upper body before taking the first step.

Descending Stairs with a Heavy Box. I can also splice the upper body of a person carrying a heavy box with the lower body of a person descending a set of stairs. The character correctly settles his weight backwards when he reaches the bottom of the steps in the spliced motion (see Figure 4.11).

Running While Carrying a Cup. Motions of people walking can also be spliced onto motions where the character is moving much faster. For instance, Figure 4.12 shows the result of splicing the upper body of the cup-carrying character onto the lower body of a person running in a curve. Again, the cup stays upright throughout the motion, the left arm swings in time with the motion of the legs, and the entire motion leans into the curve.

Walk-to-Run with a Heavy Box. The motion splicing algorithm also works when the timing of the locomotion changes. Figure 4.13 shows the result of splicing the upper body of the character carrying a heavy box onto the lower body of a person who is walking and then begins to run. The character leans back throughout the motion to counter the weight of the heavy box, and the upper body twists from side-to-side with varying intensities as he changes from walking to running.

Side-stepping While Pushing. The last example shows another unusual splice. The upper body comes from a character who is stepping randomly while pushing something, and the lower body is from a character who steps around an obstacle. The spliced motion is of the character pushing something as he steps around it (see Figure 4.14). Notice how the character appears to look to the left as he steps around the object, a characteristic that is transferred from the original lower body motion.

Figure 4.11 A motion generated by splicing the upper body of a person carrying a heavy box with the lower body of a person walking down a set of stairs.

A few other locomotion examples that I have tested my algorithm on include motions of a person jogging, walking with a jaunt, and tip-toeing. I have also run tests using a number of different upper-body actions including acting like a zombie and carrying a ceramic pot with both hands.

4.4.1 Limitations

While the method presented in this chapter works well on a wide variety of motions, there are limitations. Most importantly, since the method adjusts the way in which the shoulders twist, turn, and bend in relation to the lower body, an upper-body action whose shoulder *motion* is important to the meaning of the action cannot be spliced safely. For instance, when a person throws a football, not only does the arm rotate backward, so does the shoulder. My splicing method cannot guarantee preservation of this important shoulder twist.

One potential drawback of the motion splicing method is that it does not explicitly take into account the physics of the original motions. In some cases, not explicitly dealing with dynamics could affect the perceived physical characteristics of the objects in the scene. For example, a very heavy object may look less heavy because the character is able to accelerate with relative ease. Some of these potential problems could be fixed by running a physics cleanup algorithm, such as those in [TSK02] and [SKG03]. However, my results show that many effects due to physics are transferred from one motion to another using my example-based method. Even more importantly,

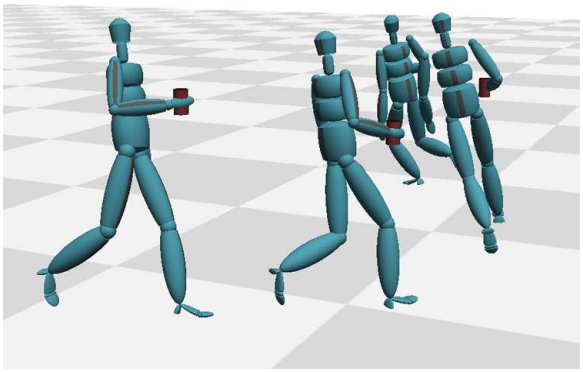


Figure 4.12 A motion generated by splicing the upper body of a person carrying a cup with the lower body of a person running.

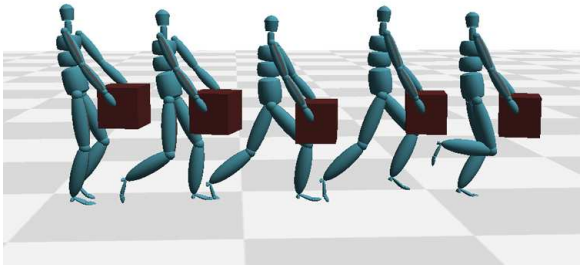


Figure 4.13 A motion generated by splicing the upper body of a person carrying a heavy box with the lower body of a person walking and then running.

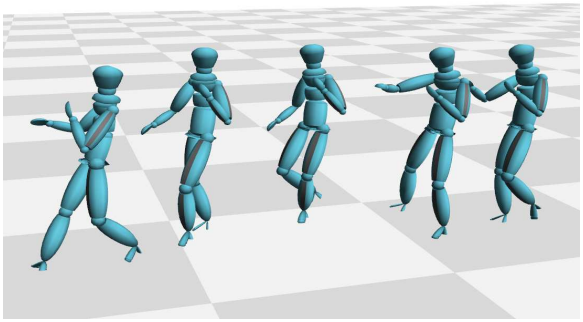


Figure 4.14 A motion generated by splicing the upper body of a person pushing something with the lower body of a person side stepping around an obstacle.

my method has the added advantage of preserving stylistic properties that are hard to capture with purely physically-based methods.

Additionally, because it depends on the locomotion cycle for time alignment, my motion splicing algorithm will not work on motions where the character is not locomoting. But because of the speed of my algorithm, it is feasible to try it on non-obvious motions, such as the boxing motion presented earlier. Unlike the results produced using the naïve DOF replacement algorithm, I have found that my motion splicing algorithm performs predictably, identifying and enforcing the desired temporal and spatial relationships.

4.4.2 Algorithm Performance

In this section, I describe how my motion splicing algorithm performs in each of the six categories described in Section 1.1.

Efficient Synthesis All of the examples in this chapter were computed on a 2.0GHz Intel Pentium 4. Each example took less computation time to produce than the duration of the final clip. For example, the 110-frame spliced motion (1.83 seconds) depicted in Figure 4.1 took $.67$ seconds to compute. This timing data includes the time it takes to build a time alignment curve and calculate the global posture information. If temporal and postural information is precomputed, then each frame of motion can be computed in $\mathbf{O}(1)$ time. On average, this constant amount of time is only $.0008$ seconds. This data shows that the splicing algorithm works quickly and in a predictable amount of time per frame.

Efficient Data Storage Because the algorithm effectively decouples locomotion from action, it reduces the amount of required storage space for example motions and associated data from $\mathbf{O}(nm)$ to $\mathbf{O}(n+m)$, where n is the number of locomotion examples needed for flexible navigation and m is the number of actions that the character can perform while locomoting.

Low Latency or Response Time Because the motion splicing algorithm is designed for motion clip generation and not motion stream generation, this characteristic is not applicable.

Accurate Motion Generation For splicing motions, requests take the form of which two motions should be spliced together. Since the algorithm always does the splice between the two requested motions, it accurately meets this request.

Visual Quality Since motion quality is partially subjective, it can be difficult to measure the quality of the motions produced. However, by limiting the changes that are made to the original motions (see Section 4.2.1), there is little deviation from the original example motions, which are assumed to be of high-quality. In addition, all of the changes made to temporally and spatially align the spliced motions are smooth and continuous.

Automated Authoring The algorithm for splicing the upper body of one motion with the lower body of another is completely automated and uses no user-defined parameters.

4.5 Discussion

This chapter presented a simple and efficient method for splicing the upper-body action of one motion sequence onto the lower-body locomotion of another, thereby decoupling a character's action from the method and path of locomotion. To preserve realism, the spliced upper body is timewarped and rotated about its attachment point on a per-frame basis in order to achieve better correlation with the lower body. Given enough example locomotion data to allow flexible control, adding different upper-body actions requires capturing just one additional example motion. In contrast, with previous algorithms for data-driven motion synthesis, creating a character that can perform multiple motions simultaneously requires a combinatorial explosion in the number of example motions that must be captured. Motion splicing has the potential of reducing the data requirements to more manageable levels by allowing independent examples to be layered atop one another.

For the examples in this chapter, I primarily used motion-captured example motions in the experiments, but procedural, keyframed, or edited motions would serve just as well as example motions. For instance, standard techniques can be used to automatically cyclify a motion in order

to extend it in length [LCR+02, AF02, KGP02]. This is a technique that was used to create some of the examples presented in this chapter.

My focus on the practical and important problem of splicing upper-body actions with lower-body locomotion rather than the more general problem of transferring any subset of the body from one motion to another has allowed me to develop a fast, simple, and reliable algorithm. But while splicing characters at the waist in order to decouple upper-body action from locomotion is useful for a large number of applications, there may be applications where splicing the body in a different location is desirable. For example, one could imagine splicing only the arm of a waving motion in order to make a character wave. While the details of my algorithm do not apply in this single limb case, the general technique of using example motions as references in order to identify and enforce spatial and temporal relationships is still relevant. Furthermore, it is likely that the three steps of the algorithm - temporal alignment, spatial alignment, and overall spatial relationship transfer - can be generalized to deal with these other splicing problems. This belief is supported by the similarities between my algorithm and the algorithm of Majkowska et al. for splicing hands onto a full body motion [MZF06]. By showing the utility of decoupling motion parameters through splicing, I hope that my algorithm with inspire other related work.

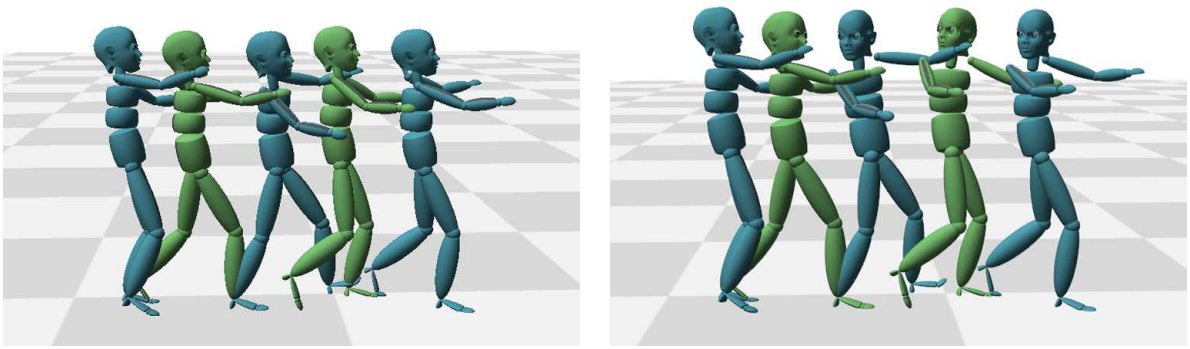
Chapter 5

Gaze Control

Because humans depend heavily on their sense of sight, it is natural for a person to look at interesting objects in the environment or down the direction of travel. To an observer, shifts in the location or direction where a character is looking might not only indicate a shift in attention but can also convey a person's goal before they act on it. For instance when faced with a choice between five different colored cups lined up on a table, a person will fixate on the cup they choose before reaching their hand out to grasp the cup. I call these natural cues the *gaze* of a human.

Currently, methods for motion clip synthesis fail to provide a way to synthesize high-quality motion of a character adjusting their gaze independently of overall body motion. Because of this, interactive characters in video games and training simulations most commonly lack gaze cues altogether. This lack of gaze cues leads to characters who look as if they are disconnected from the environment they are placed in. A human in a real environment needs to adjust their gaze many times in order to effectively take in their surroundings.

In this chapter, I present a new method for decoupling the gaze of a character from his full body motion in a way that allows high-quality, controllable motion clips to be synthesized at runtime. This model takes the form of a *parametric gaze map* that maps a requested gaze change to a low-dimensional representation of the way a human adjusts their gaze. This parametric gaze map can be applied to a base full body motion in order to adjust the gaze direction of that motion realistically and by an accurate amount. This model combines knowledge from the biological and psychological sciences with examples captured in a motion capture studio in order to achieve these realistic-looking results using little storage space. The method produces results that retain the correlations originally exhibited in each base motion while effectively adjusting the motion's gaze



a **b**

Figure 5.1 Gaze adjustment for a zombie motion. (a) A motion captured example motion of a person walking like a zombie. (b) A motion generated by adjusting the gaze of the zombie walking motion so that the gaze is directed over the right shoulder.

direction. Figure 5.1 shows the result of the technique applied to a motion of a person acting like a zombie. The gaze adjusted motion still clearly shows the character acting like a zombie but the overall configuration of the upper body has been adjusted in a natural way in order to achieve the desired gaze direction.

The rest of this chapter is organized as follows. Section 5.1 describes in detail the example-based, biologically and psychologically inspired model used to control gaze. Next, Section 5.2 describes how to construct and use a parametric gaze map to adjust character gaze. This section includes a detailed explanation of the capture procedure for example motions of a person adjusting their gaze in a motion capture studio. Section 5.3 then presents some results of the technique. Finally, Section 5.4 concludes with a general discussion of the method's uses.

5.1 Gaze Motion Model

In an interactive application, there are many instances when character gaze could be important. A character might look at objects that they are interacting with or gaze at key people or scenery as they walk by. These types of environmentally driven gaze cues can cause the character to appear more connected to the environment. Gaze cues might also be used more directly by a user. In particular, for interactive environments where many different virtual characters are being controlled by different people, such as in a massively-multiplayer online video game, a user could directly adjust the gaze of a character in order to point out interesting landmarks. As it is already commonplace in video games to control the global orientation of a character independently of the viewing direction, it would be possible to use these same controls to adjust the gaze direction of a character in order to provide better cues to other players.

A small number of recent video games have exhibited gaze cues, indicating an interest in gaze, but typically the methods employed consist of little more than twisting the head in order to point the eyes in the gaze direction[1]. But studies originating in the biological and psychological sciences indicate that gaze control is not a simple function but one that involves coordination between the

[1] One notable exception is the game Half-Life 2 where many man hours of work produced an intricate facial muscle model and eye focus controller, creating NPCs with realistic eye motion. However, the focus in Half-Life 2 is on the face and eyes, not necessarily on the motion of the body that achieves an overall gaze pose.

eyes, head, and spinal joints. In fact, these studies have found that head rotation actually factors little into overall gaze change [GV87, FS00].

Interactive applications are in need of a realistic method for adjusting the gaze of a character. While a character is performing some other action or actions with their full body, such as walking, running, shooting a gun, or holding a suitcase, it would be desirable to be able to redirect the character's gaze in response to environmental or user-directed gaze goals without unduly affecting the overall body motion.

The goal of gaze control in this chapter is to be able to synthesize a new *clip* of motion where a character performing some independent full body motion, called the *base motion*, adjusts his pose in a natural way such that his eyes point in a specified direction in relation to where they would point without the adjustment. Gaze control allows gaze to be directed independently of full body motion.

Because of the importance of both efficiency and realism, my approach to gaze control is to use motion-captured examples of a person adjusting their gaze to build a low-dimensional model for motion adjustment that effectively captures important aspects of gaze. In the rest of this section, I review work in the biological and psychological sciences that study how real humans adjust their gaze. I then describe in detail my example-driven model for gaze control, called a *parametric gaze map*, that is informed by these observations.

5.1.1 Biological and Psychological Observations

For many decades, the biological and psychological communities have been interested in understanding how humans (and other primates) unconsciously adjust their gaze. These studies have observed a number of interesting facts about human gaze that can be used to support and inform a model for virtual gaze control.

Many of the studies on how humans adjust their gaze focus only on how the head and eyes contribute to overall gaze by restricting the motion of the shoulders [KGM07, GV87, FS00], but more recently researchers have begun to focus on the importance of the spine for adjusting gaze as well. In particular, McCluskeyl and Cullen's [MC07] recent study of the way unrestrained rhesus

monkeys adjust gaze shows that the spinal joints can have a significant effect on gaze direction, especially for gaze shifts of more than $40°$.

Psychological literature observes that the proportional motion of the spinal joints, head, and eyes used to adjust gaze to a desired direction is primarily guided by complex psychological processes in the brain rather than by the mechanics of the body [GV87, FS00]. And this work further shows that these proportions change depending on the subject as well as the size of the gaze adjustment. These observations argue against models of human gaze that attempt to achieve a desired gaze goal by minimizing the amount of energy expended.

But the biological and psychological literature also observes that gaze adjustments follow a definable pattern. One pattern observed in gaze adjustment motions is that the onset of movement used to adjust gaze cascades from the eyes down the spine [KGM07, GV87, FS00, MC07]; the eyes begin to move before the head, and the head begins to move before the upper back, and so on. The study performed by Kim et al. [KGM07] drew attention to another pattern of gaze adjustment motions. In this study, when presented with a target to look at, a subject would quickly adjust their gaze direction in a gross way, usually overshooting their target, and then slowly readjusting in order to correct for this gross motion. However, Kim et al. also observed that the details of how a person adjusts their gaze differs from one subject to another. The speed of the motions, the point at which each joint begins to move, and the amount by which subjects overshoot their target differ between subjects.

These findings in the biological and psychological communities support my approach of using an example-based, low-dimensional gaze model to adjust character gaze in two ways. First, the findings show that individuals adjust gaze differently depending on many factors, including personal characteristics, such as flexibility. This individuality motivates an example-oriented approach that would allow the gaze of a virtual character to be adjusted in accordance with an individualized model. Second, biologists and psychologists observe that even though different people adjust their gaze differently, each of these individuals follows patterns that are consistent across all gaze adjustment motions. These similarities can be used to design a low-dimensional model for the way an individual gaze adjustment is made.

5.1.2 A Biologically and Psychologically Inspired Model for Gaze

The biological and psychological literature reviewed in Section 5.1.1 provides a pattern for gaze control. In particular, it observes that:

1. the spinal joints, head, and eyes contribute to the adjustment of gaze
2. the initiation of the movement of these different joints follows a cascading effect, and
3. during the gaze adjustment, gaze often overshoots its goal orientation, resulting in a slow readjustment period.

But this general pattern does not provide the quantitative details necessary to adjust gaze. These missing details include how much the gaze overshoots, the time at which this *overshoot peak* is reached, the time delay between the movement initiation of each of the joints, and the proportional amount that each joint contributes to the overall gaze adjustment. In fact, the literature suggests that there is not a single correct value for these terms; instead, details differ depending on the size of the gaze adjustment as well as the individual who performs the adjustment.

My method for decoupling character gaze from overall body motion is strongly guided by these observations. To represent the adjustments made to a base motion in order to adjust gaze, I have developed a low-dimensional model that explicitly accounts for the patterns observed in real human motion. Quantitative values associated with the model are filled in using example motions of a human subject adjusting their gaze.

My model is called a *parametric gaze map*. A parametric gaze map maps the amount by which a character's gaze needs to be adjusted to a motion of the spinal joints, head, and eyes that in a short amount of time will achieve the desired gaze adjustment when added to the base motion[2]. I can parameterize the amount of a gaze adjustment as the change in yaw, r_{yaw}, and pitch, r_{pitch}, of the eyes in relation to the root. This parameterization is equivalent to a latitude/longitude parameterization of the unit sphere. Biologically, it is possible to adjust the gaze of a person looking straight forward by an amount that corresponds to any point on the unit sphere, (r_{yaw}, r_{pitch}) where

[2]This mapping assumes that the way that a person adjusts their gaze does not depend on their starting configuration, but within the psychological community there is a debate as to whether this is true or not [KGM07].

$r_{yaw} \rightarrow \{-180°, 180°\}$ and $r_{pitch} \rightarrow \{-90°, 90°\}$. In fact, [KGM07] observes that horizontal gaze in the range $\{-180°, 180°\}$ is possible *even* when the shoulders are rigidly held.

For each (r_{yaw}, r_{pitch}) pair, the parametric gaze map supplies the spine, head, and eye adjustments needed to achieve the desired gaze change. I call these additive spinal motions *gaze displacement motions*. A gaze displacement motion consists of two parts that explicitly encode the patterns observed in real human motion. A gaze displacement motion consists of two parts. The first explicitly represents the change in a character's skeletal *pose* that is used to achieve a desired gaze orientation. The second focuses on the pattern of the *motion* used to achieve this final pose.

I represent the final pose adjustment for a gaze displacement motion, $\mathbf{P}^+$, as a displacement of the spinal joints, head, and eyes from the base pose[3]. For my skeletal model, this can be stored as a 5-tuple of quaternion rotational offsets:

$$\mathbf{P}^+ = \{\mathbf{q}^+_{Pelvis}, \mathbf{q}^+_{LowerSpine}, \mathbf{q}^+_{UpperSpine}, \mathbf{q}^+_{Neck}, \mathbf{q}^+_{Eyes}\} \quad [5.1]$$

The motion portion of a gaze displacement motion uses a 7-tuple of floating point numbers. This includes:

- l: The length of time in seconds that it takes to adjust gaze, starting from the point when the adjustment begins and ending when the full gaze adjustment, $\mathbf{P}^+$, has been reached.
- $\mathbf{T} = \{t_{Pelvis}, t_{LowerSpine}, t_{UpperSpine}, t_{Neck}\}$: The time at which each joint starts moving, other than the eyes, which are assumed to start moving immediately. For convenience, this time is normalized to the range 0 to 1 with respect to the length of the motion.
- t_β: The time at which the motion reaches its overshoot peak, also normalized to the range 0 to 1 with respect to the length of the motion.
- β: And, finally, the amount by which the motion overshoots, stored as a multiplier associated with the final pose displacement.

[3]Note that because eyes are not a part of the skeletal hierarchy presented in Section 3.1 but are an important part of gaze, I augment my skeleton hierarchy with the addition of eyes. For data without eye orientation information, the eyes are assumed to be looking straight forward from the head.

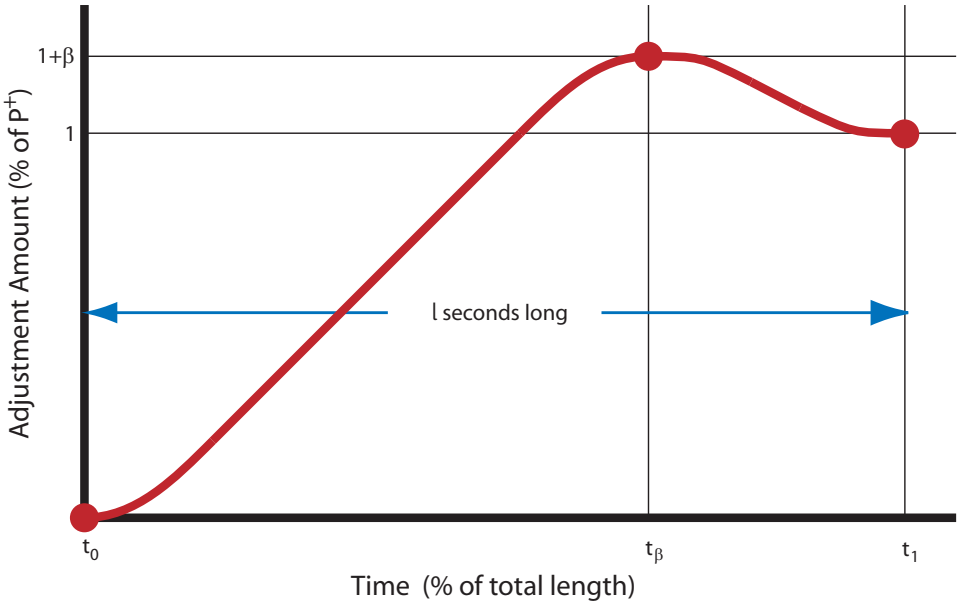


Figure 5.2 A pictorial depiction of a gaze displacement motion.

Putting all of these numbers together, a gaze displacement motion, $\mathbf{G}$, can be defined as:

$$\mathbf{G} = \{\mathbf{P}^{+}, l, \mathbf{T}, t_{+}, \beta\} \quad [5.2]$$

This representation of the motion of a person adjusting their gaze is a direct mapping of the three pattern observations from the biological and psychological sciences described at the beginning of this section. The values for any particular gaze displacement motion can be filled in using an example gaze adjustment motion (see Section 5.2.2.2). See Figure 5.2 for a pictorial representation of a gaze displacement motion.

I construct a parametric gaze map from a set of example gaze displacement motions using a form of blending-based parametric synthesis. The gaze displacement motion, G, returned for any (r_{yaw}, r_{pitch}) can be layered onto a base motion to adjust the character's gaze. By representing an entire example motion with just a few parameters, my method not only reduces the amount of storage needed for a gaze displacement motion but also allows gaze to be applied efficiently at runtime.

5.2 Parametric Gaze Maps

This section describes my method for capturing example motions of a person adjusting their gaze in a motion capture studio, converting the raw motion capture data collected at the studio into a gaze displacement motion, building a parametric gaze map from these gaze displacement motions, and applying the parametric gaze map to a base motion to adjust gaze.

5.2.1 Capturing Gaze

For this dissertation, I sought to capture example motions for a parametric gaze map that show how a human adjusts their gaze when asked to look at different locations in 3D space. Unfortunately, because of the confines of the capturing environment, it was necessary to limit vertical gaze to a much smaller range - $r_{pitch} \rightarrow \{-12^{\circ}, 12^{\circ}\}$ for all of the experiments presented in this dissertation; the horizontal gaze range of the subject was captured in full. Because of my model's

dependence on a specific type of gaze motion, this section describes in detail how these example motions were captured for this dissertation.

5.2.1.1 Motion Capture Preparation

To capture example motions of a subject adjusting their gaze, it is useful to have objects in the motion capture environment that can be used as gaze targets. Acting as these points of interest, 33 signs labeled with letters were placed around the motion capture environment. With a human subject at the center, signs were placed at 36° increments along the horizontal axis and at 12° increments along the vertical axis, starting from the point $(0, 0)$, or the point where the subject's gaze is directed when at rest. Figure 5.3 diagrams the setup of the signs along these axes.

Markers were then attached to the subject's upper body along the spine, on the head, and on the arms. For the subject's spine, I marked three landmarks with an equilateral triangle of markers: the lower back, the upper back, and the base of the neck, between the shoulder blades. By using a triangle of markers, I was able to guarantee that I could construct a coordinate system at each of these points along the subject's spine during processing (see Section 5.2.2.2). I also placed a marker at the base of the spine as a reference point for the motion. The subject's head was outfitted with a hat with four markers attached to the sides in a square and one marker attached to the top, in the middle. A small additional marker was placed between the subject's eyes. I used several extra markers for visualization purposes only: one on each of the subject's shoulders, elbows, and wrists. These markers were observed and recorded by a standard optical motion capture system. Figure 5.4 shows the arrangement of these markers in 3D space.

My marker setup does not support capturing the motion of the eyes, but as stated previously, the eyes play an important role in gaze. As described later in Section 5.2.2.1, the orientation of the eyes in relation to the head is inferred from the overall orientation of the upper body in conjunction with the known target gaze direction.

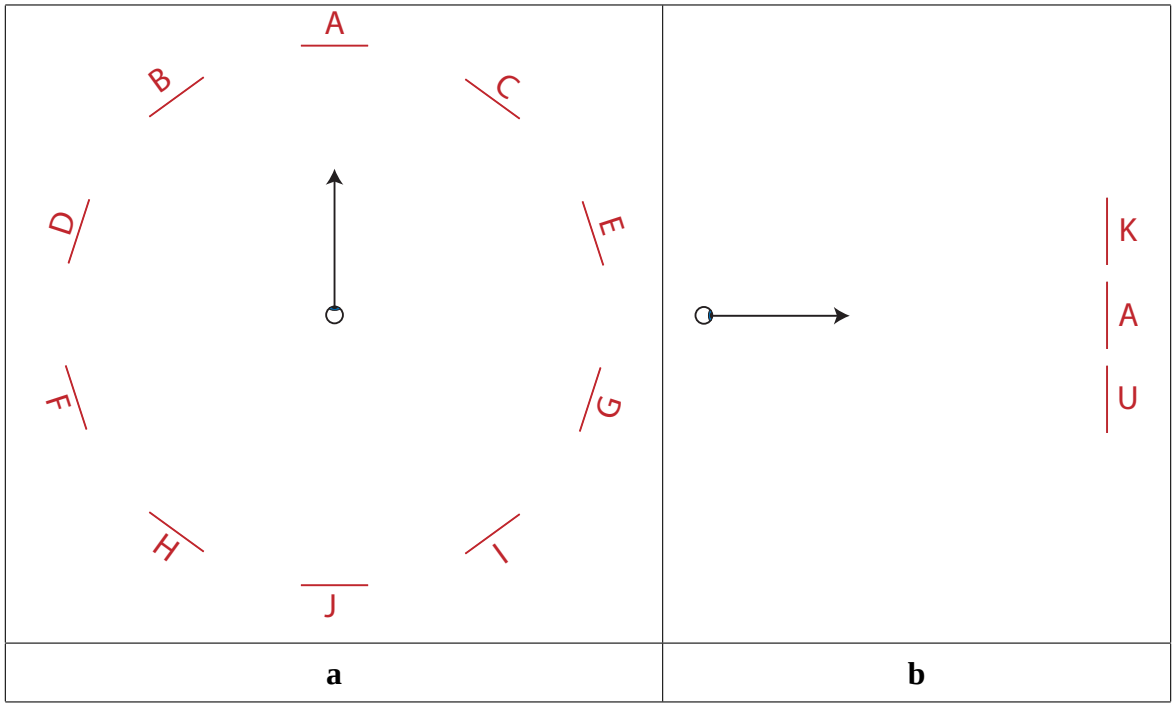


Figure 5.3 Sign placement. Signs with letters on them were arranged within the motion capture environment to act as targets for gaze capture. In all, there were 33 signs arranged in 3 latitudinal lines and 11 longitudinal lines. (a) shows an overhead view of sign placement along the middle latitudinal line. The eye and arrow in the center depict the location and orientation of the subject at rest. (b) shows a side view of sign placement along the front longitudinal line. Again, the eye and arrow depict the location and orientation of the subject at rest.

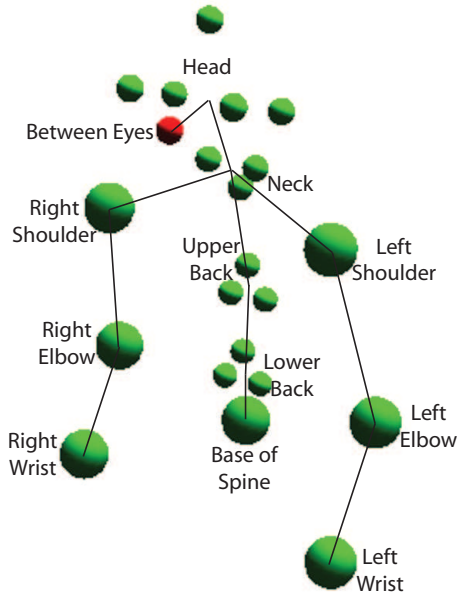


Figure 5.4 Marker placement for gaze capture.

5.2.1.2 Capturing Each Example Motion

The goal of the motion capture session was to capture the motion of the subject adjusting his gaze from a starting position to each of the target locations indicated by the lettered signs. For each captured motion, an assistant and I would:

1. point a laser pointer at the target sign, and
2. play a tone twice, in a short, long pattern, using a tone making device positioned behind the target sign. The tone making device was capable of playing 3 tones. The highest tone was always played for targets located along the highest latitude line, the middle tone was always played for targets located along the middle latitude line, and the lowest tone was always played for targets located along the lowest latitude line.

Because I wanted to capture the natural way that the subject adjusts his gaze, it was important that the subject not consciously plan his motions. Thus, the 33 motions were captured in a random order. The subject was told to do the following for each of these trials:

1. Look straight ahead at the letter **A**.
2. Close your eyes.
3. When you hear the first, shorter tone, open your eyes.
4. When you hear the second, longer tone, look at the target sign, as indicated by the tone and laser pointer.
5. Read what is written on the target sign.

Each of the motions were captured at $60hz$ and stored as raw motion capture data (i.e., the 3D locations of each of the markers on each frame). The next section describes how I use this raw data to create a parametric gaze map.

5.2.2 From Raw Motion Capture Data to a Parametric Gaze Map

The motions captured using the method described in Section 5.2.1 provide examples of the way the subject adjusts his gaze over a large range of possible gaze changes. In this section, I describe how to transform these motion-captured observations from moving 3D positions in space to a parametric gaze map that can be effectively applied at runtime to adjust a character's gaze.

5.2.2.1 Cleaning the Data

When motion is captured in a motion capture studio, the raw data that is produced often needs to be cleaned to conform to the standards needed for the example motions. For the case of the gaze example motions captured for gaze control, it is important that the 3D locations of each of the markers appear in every frame of motion, that the upper body motion is stabilized in relation to any extraneous lower body motion, and that the motion is trimmed to only include the region of time that is interesting for gaze control.

When a motion is captured using the sensors at a motion capture studio, it is possible that one marker might become "invisible" to the sensors for a brief period in time. For instance, for the optical motion capture system used to capture all of my example motions, it is possible for markers to become blocked from the cameras by objects in the environment or self-occlusion of the subject. If they are well-placed, it is unlikely for the markers to disappear for long. Thus the first step for cleaning the captured gaze examples is to fill in any missing marker data points.

For each motion-captured gaze example, missing marker locations are inferred using linear interpolation of existing data in temporally nearby times. In particular, a missing marker data point at time t is calculated as the weighted average of the marker's last known location, prior to time t, and next known location, after time t, weighted according to the amount of time between the observations.

After filling in missing marker data, the motion examples can be stabilized. My gaze model presented in Section 5.1 considers only the motion of the spine, head, and eyes. Yet, slight changes in the global position of the base of the subject's spine affects the velocities of the markers on the rest of the upper body. Thus, the second step needed to clean the gaze example data captured at

the motion capture studio is to stabilize the root such that it does not move, while maintaining the relative distance between each of the markers in 3D space. This can be done efficiently by shifting the entire captured motion on a frame-by-frame basis so that the marker at the base of the spine is always at the origin.

Finally, because the motion capture equipment begins recording each motion before the subject starts to move and continues to record after the subject has completed the task of changing his gaze, the raw data for each motion-captured example contains additional frames of motion before and after the needed example motion. Ideally, each motion would begin at the moment when the subject begins to move and end after the subject has finished reading the letter printed on the target sign (see Section 5.2.1 to review the sequence of steps used to capture the gaze example motions in a motion capture studio). This region of motion can be extracted from the adjusted motion by locating periods of time when the subject is barely moving, henceforth referred to as *periods of rest*. To do this, my system uses the following simple, automated technique:

1. For each frame of motion, t_i, it calculates the average velocity magnitude of the markers, v_i, using finite differences.

2. Next, the system labels a frame i as "still" if v_i is below a tunable threshold, V. For processing all of the data in this dissertation, $V = 6mm/s$. A sequence of "still" frames are then grouped together to form a period of rest.

3. To ward against the effects of noise, the system combines periods of rest that are separated by a short amount of time ($1/6$ of a second for my data) to form a single larger period of rest.

4. I can assume that the first period of rest temporally corresponds to the period of time when the subject is waiting for the cue to change his gaze. Similarly, the second period of rest most likely corresponds to the time when the subject has settled his gaze on the target sign and is reading the writing. Thus, the motion data is automatically cropped by the system such that it only contains the frames of motion starting from the end of the first period of rest to the end of the second period of rest.

In most cases, this simple technique effectively crops the example motions. The notable special case is the gaze motion whose target is $(0, 0)$. Since a change of gaze from $(0, 0)$ to $(0, 0)$ does not actually include a period of motion, the example must be cropped manually. All of the other example motions for this chapter where cropped automatically.

5.2.2.2 Building a Parametric Gaze Space

After the captured example motions have been processed, they can be quickly and automatically constructed into a parametric gaze map that allows fast and accurate synthesis of motions with gaze adjustments. First, each of these motions are translated into a gaze displacement motion (see Section 5.1). These gaze displacement motions are then mapped into a parametric motion space of gaze motion, forming the parametric gaze map. This parametric gaze map can be applied at runtime to adjust gaze as described later in Section 5.2.3.

To calculate gaze displacement motions for the processed motion data, the system analyzes the orientations of the joints throughout the example motion. To do this, it is necessary to be able to calculate the orientations of each joint in the captured motion. For the joints along the spine that were marked with a triangle of points, the orientation can be defined by constructing the local coordinate system defined by the triangle (or the square, for the head). When placed in matrix form, this coordinate system represents the orientation of the landmark in relation to the origin. This matrix can be converted to a quaternion representation using the standard conversion equation [Sho85].

Given this method for computing the global orientation of a joint from the raw motion capture data, my system calculates the pose portion of the gaze displacement motion, $\mathbf{P}^{+}$, that coorresponds to a processed, example motion using the following method:

1. The final orientations of the joints are translated into a displacement from the initial pose by subtracting the orientation of each joint in the first frame from the corresponding joint in the last frame.

2. These global displacement orientations are then converted into hierarchical displacement orientations, or into a local orientation displacement from its parent joint.

3. Finally, because each gaze displacement motion was captured with a known, calibrated gaze goal, (r_{yaw}, r_{pitch}), the local orientation of the eyes in relation to the head can be inferred as the orientation needed in order to achieve a global displacement of the eyes equal to (r_{yaw}, r_{pitch}).

Next, to fill in the motion portion of the gaze motion, it is necessary to locate the point in time when the motion reaches its overshoot peak, as well as the points in the example motion where each joint begins to move.

Locating the Overshoot Point:

1. The method starts by calculating the velocity of the global orientation of the head in each frame, i. This velocity can be calculated using finite differences as the angle between the orientation of the head in frame $i - k$ and and the orientation of the head in frame $i + k$, where k is a tunable parameter used for finite differencing.
2. A frame j is identified as a possible overshoot peak if the velocity at j has a different sign from the velocity of $j - 1$.
3. The overshoot point is identified as the first flagged frame where the average velocity of the frames $\{j - m, j - 1\}$ and the average velocity of the frames $\{j + 1, j + m\}$ have different signs. Again, m is a tunable parameter used to define a local window size around the frame in question.

Identifying the Start Times for Each Joint: To identify the start time of each joint, the algorithm calculates the velocity of the local orientation of the joint in question using the same method used to locate the overshoot point. The first frame at which the magnitude of this velocity surpasses a tunable threshold is defined as the start point for that joint. If no start point is found, the algorithm assumes that the joint begins moving slowly at the start of the gaze change. For all of the examples in this chapter, I set the tunable velocity threshold to $18^{\circ}/s$.

As described in Section 5.1, the motion of a person changing their gaze is strongly dependent on the gaze parameters (r_{yaw}, r_{pitch}). The gaze example motions act as a sampling of the space of all possible gaze changes. Following the blending-based parametric synthesis work presented in Section 3.5, I can represent the way a person adjusts their gaze as a parametric gaze motion space, parameterized on the orientation of the gaze, (r_{yaw}, r_{pitch}). Each motion in this space is defined as a blend of the example gaze motions. My system directly uses the parametric synthesis method of Kovar and Gleicher [KG04] presented in Section 3.5 to construct this space of gaze motions. However, since the motions in this motion space are represented using the gaze motion model presented in Section 5.1, it is necessary to be able to blend motions in this representation.

The form of the representation of a gaze motion makes blending two gaze motions together straightforward. Because the pose of a gaze motion is represented as a displacement from the base pose, blending the poses of two gaze motions does not require the motions to be spatially aligned. Thus, to blend the pose portions of two gaze motions, the 5-tuples of quaternion offsets can be linearly interpolated without modification. Similarly, the motion portion of a gaze motion does not require temporal alignment since the key events are already explicitly marked in time (e.g., the gaze motion representation explicitly stores the time at which the head begins to move). So blending the motion portion of two gaze motions is as easy as linearly interpolating each of the 7 floating point numbers used to describe the shape of the motion (see Section 5.1).

5.2.3 Applying the Parametric Gaze Map

A parametric gaze map can be used to efficiently adjust the gaze of an existing base motion clip. A user or application simply supplies the desired change in gaze, (r_{yaw}, r_{pitch}). These parameters are used by the parametric motion space of gaze motions represented by the parametric gaze map. Using the blending-based synthesis method discussed in Section 3.5 combined with the gaze motion blending algorithm presented in Section 5.2.2.2, the parametric gaze map can supply a gaze displacement motion that accurately describes how to adjust the character's gaze by the desired amount.

This new target gaze motion can then be applied to a base skeletal motion by adding in the appropriate spinal, head, and eye orientation displacements on each frame. Using the definition of a gaze motion from Section 5.1, the orientation displacement of a joint, j, at time t, is:

$$\mathbf{q}_j^+(t) = \left\{ \begin{array}{ll} I & \text{if } t < t_j \\ (\frac{t-t_j}{t_\beta - t_j} * (1+\beta)) * \mathbf{q}_j^+ & \text{if } t_j \leq t \leq t_\beta \\ (1 + (1 - \frac{t-t_\beta}{1-t_\beta}) * \beta) * \mathbf{q}_j^+ & \text{if } t_\beta < t \end{array} \right\}$$

where I represents the identity rotation. In other words, there is no local orientation displacement of a joint, j, from its parent until time t_j. At time t_j, the joint's displacement begins changing linearly until it reaches its overshoot orientation, $(1+\beta) * \mathbf{q}_j^+$, at time t_β. Finally, the joint's displacement changes linearly from this overshoot point until it reaches its goal orientation, $\mathbf{q}_j^+$, when time equals 1.

In practice, the system produces motions with $c(1)$-continuity by replacing the linear interpolation terms $\frac{t-t_j}{t_\beta - t_j}$ and $\frac{t-t_\beta}{1-t_\beta}$ with an ease-in/ease-out function that was originally presented in [HWG07]. This function introduces acceleration and deceleration periods to the interpolation, providing better continuity.

By only applying this smooth displacement map to the base skeletal motion, the system guarantees that it will not introduce discontinuities that might be striking to the eye. Additionally, the method avoids producing motions that appear too smooth by layering these low-frequency gaze changes on top of the motion that already exists. The high-frequency details that exist in the base motion often contain correlating submotions, such as the slight bobbing of the head in time with footsteps, that are necessary to make a motion appear natural. All of these correlating details are retained when a parametric gaze map is used to adjust a character's gaze.

5.3 Results

To test the methods presented in this chapter, I applied the parametric gaze map I developed to a number of different base motions.

Figure 1.4 shows the result of adjusting the gaze of a character stepping up onto a platform so that he looks over his left shoulder. The motion retains the correlating details of the original

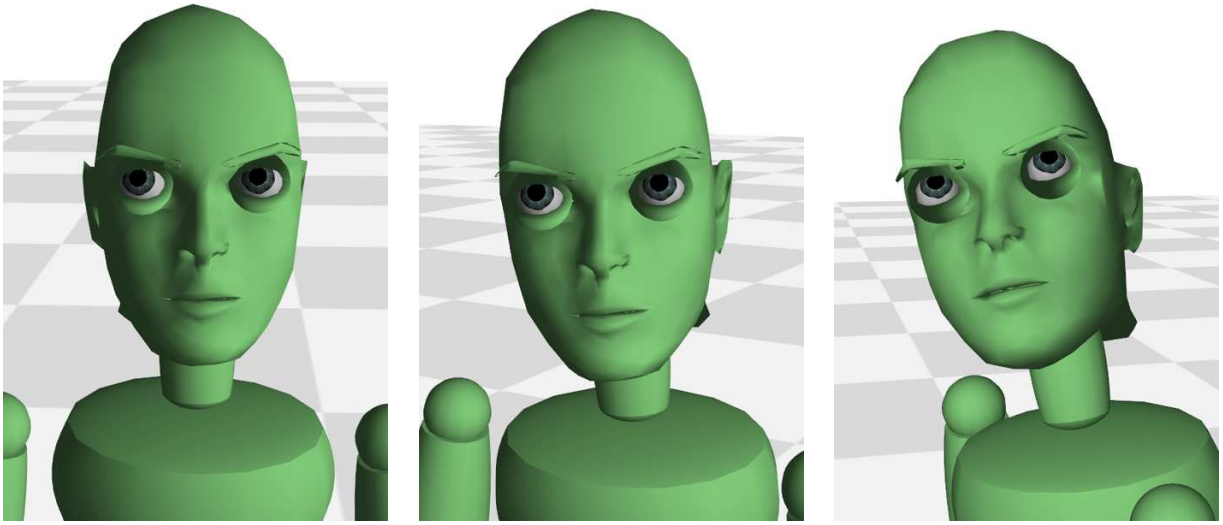
base motion, such as a dramatic nod of the head as the character steps up onto the platform, while the overall gaze of the character is adjusted realistically to meet the requested gaze change. The motion of the character also exhibits overshoot of the gaze goal, a characteristic of real human motion [KGM07]. Figure 5.1 shows another example of a gaze change applied to the motion of a person walking like a zombie. Note that the resulting adjusted motion is still recognizable as a zombie motion, but the pose of the character's upper body has been adjusted to change gaze.

Figure 5.5 more effectively illustrates the overshoot phenomenon. This figure shows the results of adjusting a walking character's gaze such that it is pointed towards the ceiling over the left shoulder. Before the gaze adjustment is made, the eyes peer straight forward in relation to the root (Figure 5.5a). As the character adjusts his upper body orientation in order to meet the requested gaze, the eyes pass their intended goal (Figure 5.5d). The character then follows up by slowly correcting for this overshoot until he achieves the desired gaze adjustment (Figure 5.5f).

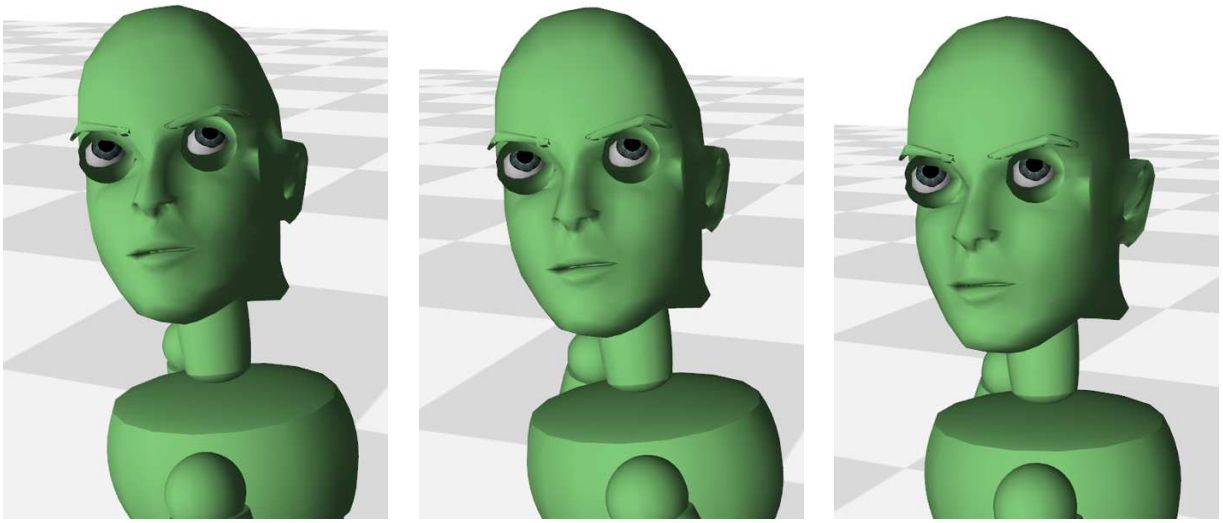
One of the strengths of using an example-based method for controlling gaze is that it captures nuances of gaze adjustments that are particular to specific subjects. For instance, the subject whose gaze was captured for the experiments in this chapter does not adjust his gaze in a symmetrical way; through evaluation of the captured motion data, it is clear that the subject's spine is more easily twisted towards the right. For all of the captured example motions where the subject turned towards the left, the subject's eyes and head contributed in a much greater proportion to the overall gaze change than when the subject turned towards the right. This asymmetrical gaze characteristic is transferred using the parametric gaze map. Figure 5.6 shows the final skeletal configuration after applying the parametric gaze map to a motion consisting only of the dress pose[4]. This figure illustrates how motions that represent a gaze change to the left result in a pose where the spinal joints contribute relatively little to the overall orientation when compared with the same gaze change to the right.

Figures 5.7 and 5.8 show additional examples of a character's gaze being adjusted using a parametric gaze map. In the first is a detailed look at the motion synthesized by adjusting the gaze of a person carrying a heavy box along a curved path so that the character looks more intently at

[4]The dress pose of a skeleton is the pose of the skeleton where all of the local joint orientations are set to 0

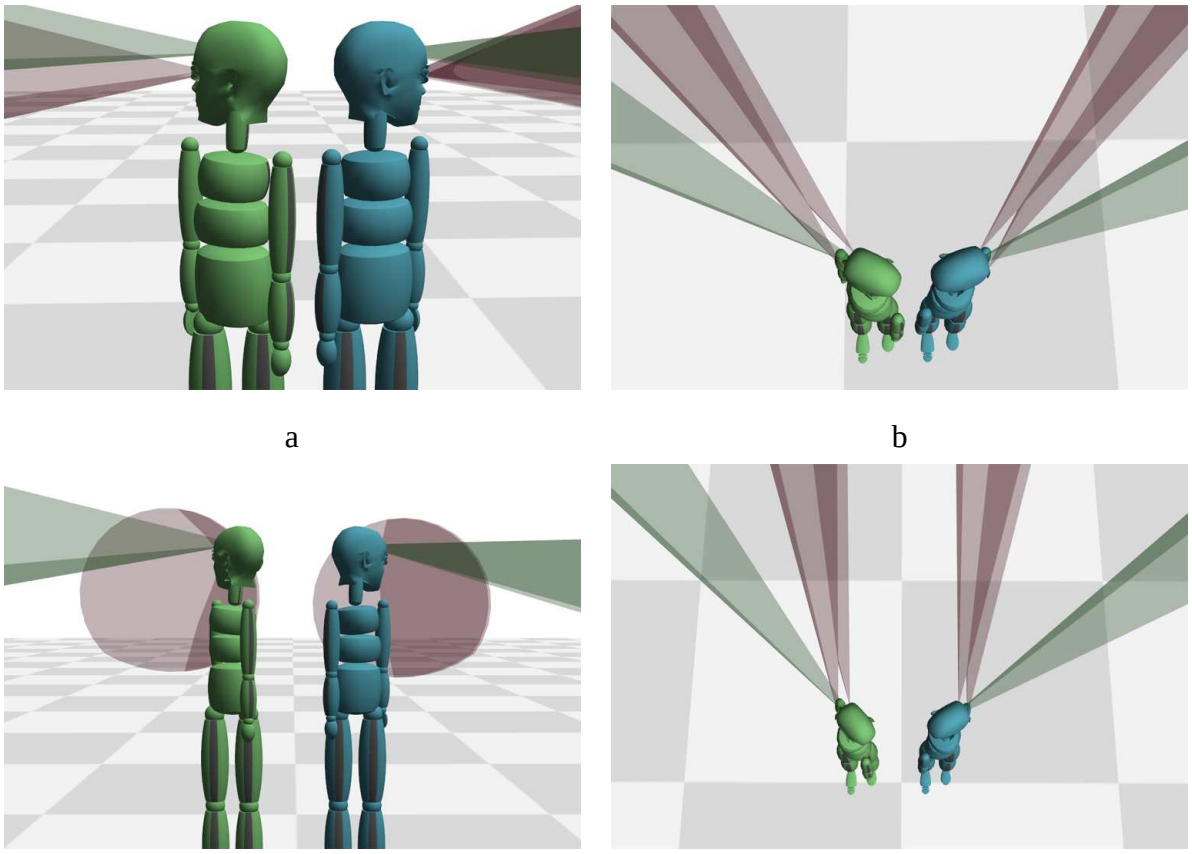


a b c



d e f

Figure 5.5 A motion of a person adjusting their gaze toward the ceiling over their left shoulder. This motion was synthesized by applying a parametric gaze map. The character starts by looking straight forward in (a). Notice how in (d) the character has overshot his gaze goal (focus on the rotation of the eyes, as this is the easiest way to notice the overshoot in a series of still images) before coming to rest at his goal adjustment in (f).



a b

c d

Figure 5.6 Asymmetric gaze changes captured by a parametric gaze map. (a) and (b) show front and top views, respectively, of the final pose configuration of a skeleton after applying a gaze change of approximately 140° to the right and left. The skeleton was initially configured in a dress pose, where each of its local joint orientations were set to 0. (c) and (d) show a similar application of the parametric gaze map for rotations of approximately 175° to the right and left. Notice how in both instances, the righthand gaze change is not symmetrical with the lefthand gaze change. Instead the eyes, whose orientation is clearly indicated by the red cones protruding from the head, contribute more to the overall orientation when the character turns towards the left. Compare the angles between the red eye cones and the green head orientation cones.

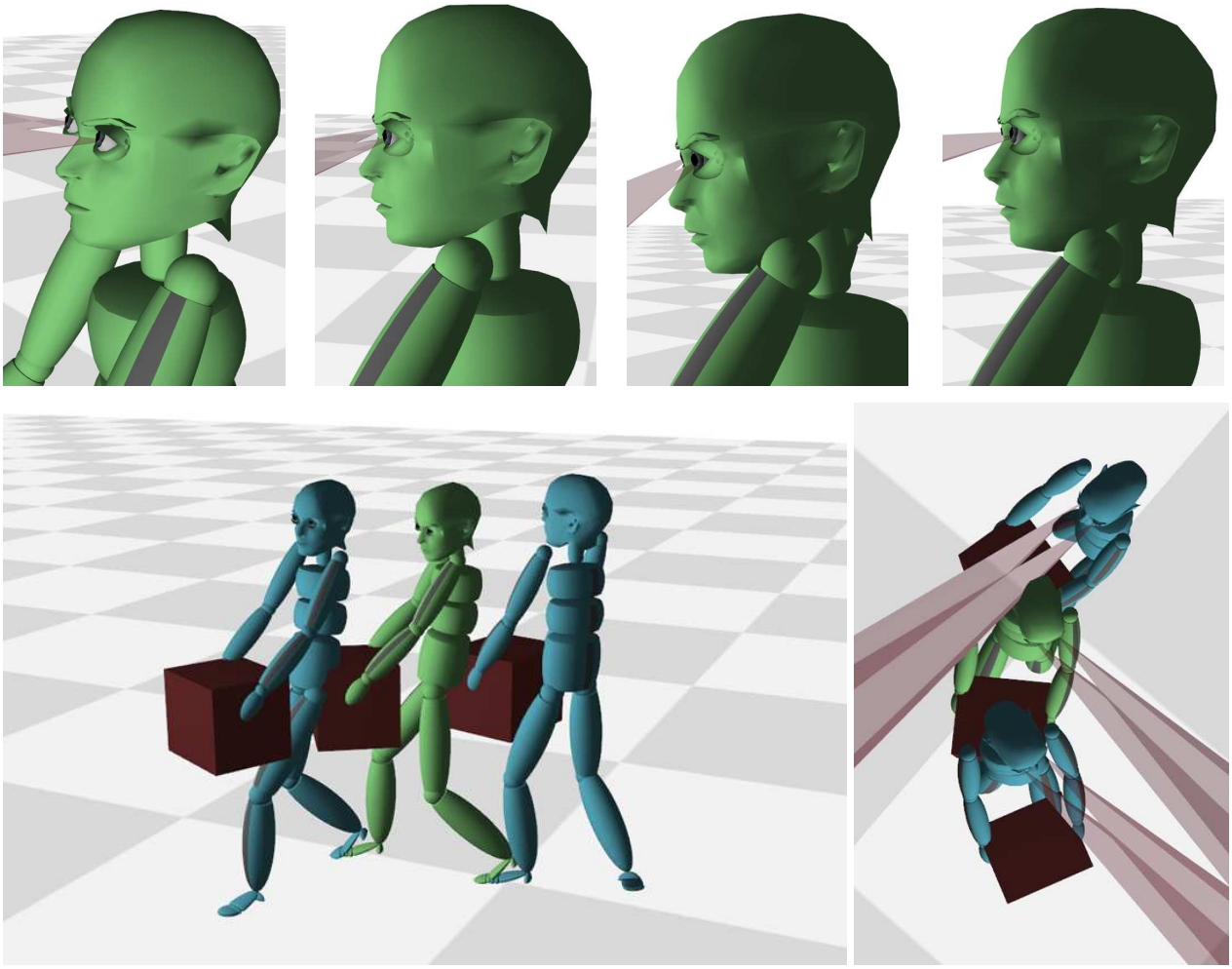


Figure 5.7 A motion synthesized by adjusting the gaze of a character walking along a curved path such that the character looks at the ground along his intended direction of travel. The images in the top row focus on the motion of the head and eyes over time, while the images in the bottom row provide an overview of the entire motion from the side and the top.

the ground along his direction of travel. Figure 5.8 compares the results of different gaze changes applied to the same base motion.

5.3.1 Algorithm Performance

In this section, I describe how my biologically and psychologically inspired method for adjusting the gaze of a character at runtime performs in each of the six categories described in Section 1.1.

Efficient Synthesis The examples in this paper were computed on a laptop computer with a $1.75GHz$ Pentium M Processor, $1GB$ of RAM, and an ATI Mobility Radeon X300 graphics card. All of the generated motions were sampled at $30Hz$. Because applying a gaze change to an existing motion clip is a fast, constant time operation, it is possible to apply gaze changes quickly at runtime. The only part of applying a parametric gaze map that is not necessarily constant is constructing the blended gaze motion in the parametric motion space of gaze motions. The time it takes to construct a blended gaze motion is dependent on the number of example motions being blended together. But because Kovar and Gleicher's method for blending-based parametric synthesis [KG04] limits the number of motions that can be blended together at each of the sample points in a parametric motion space, this time is effectively bounded.

Efficient Data Storage The storage requirements for a parametric gaze map are low. Since the example motions are stored using the compact, biologically and psychologically inspired gaze motion model, it is possible to store many example motions in a small amount of space. Additionally, since my algorithm effectively decouples gaze from full body motion, it reduces the number of example motions that need to be stored for quality control of more than one motion parameter simultaneously.

Low Latency or Response Time Since the method for adjusting the gaze of a character is designed for motion clip generation and not motion stream generation, this characteristic is not applicable.

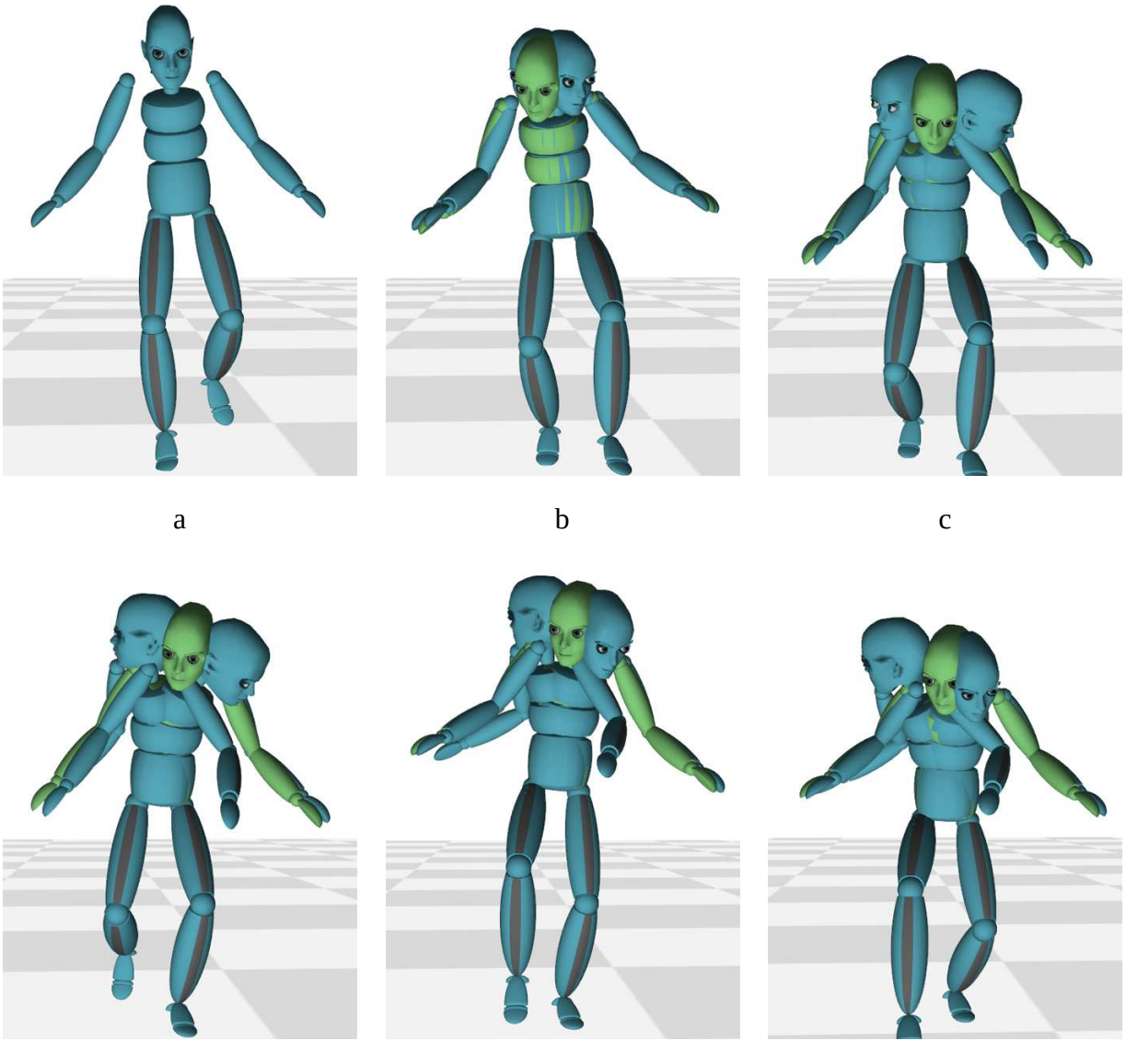


Figure 5.8 Variations on a motion: Three motions synthesized by applying a parametric gaze map to a tip-toeing motion using different gaze goals. One character is turning to look high over his right shoulder, another is looking down towards his feet, and the last is turning towards the left by a small amount.

Accurate Motion Generation Because each gaze displacement motion is computed from the example motions using blending-based parametric synthesis methods, it is possible to identify a motion that should accurately adjust a character's gaze. But there are limitations to the model that could compromise the accuracy of the method. It is necessary that the base motion that a parametric gaze map is applied to not already contain any gaze changes. A low-frequency gaze direction change in the base motion would require a new gaze change goal. Furthermore, the small details in the base motion that are retained by only applying a smooth displacement map during gaze changes might cause the gaze direction of the character to "bob." This bobbing can be naturally corrected by adjusting the eye and/or head orientation slightly in order to compensate for these small submotions.

Visual Quality Again, the quality of a virtual character motion is a partially subjective characteristic. By limiting the changes made to the base motion (see Section 5.1), deviations are small relative to the original example motion, which is assumed to be of high-quality. In addition, all of the changes made to adjust the spine, head, and eyes are smooth and $c2$-continuous. As the gaze model is informed by studies in the biological and psychological communities, it explicitly captures characteristic features of gaze change motions. However, the gaze motion model is lossy. In the gaze motion representation, the motion of each joint from starting pose to overshoot pose and from overshoot pose to final pose are assumed to be simple functions. Any deviation of the actual motion from this simple representation will be lost by the model.

Automated Authoring The algorithm for building the parametric gaze map for gaze control is highly automated, using only a small number of user-defined parameters, with the notable exception that a human is needed to capture the original example motions. It is also necessary for a user to supervise the automatic processing of the raw motion capture data in order to intervene when automated cleanup fails.

5.4 Discussion

This chapter presented a new method for decoupling gaze from overall body motion, greatly reducing the number of example motions needed to control gaze on top of other parameters simultaneously. Because the method is inspired by biological and psychological research, it treats motions of a person changing their gaze in a natural way, explicitly capturing many characteristics of these motions, such as overshoot and cascading joint motion. And, because the model uses captured example motions of a person adjusting their gaze, it is capable of synthesizing motions with stylistic properties associated with the way an individual adjusts their gaze, such as asymmetry.

For the examples in this chapter, only motion-captured example motions and base motions were used in my experiments, but procedural, keyframed, or edited motions would serve just as well. For instance, it should be possible to synthesize a stream of walking motion using one of the techniques discussed in Section 2.2.4 and then to apply the gaze model to these synthesized motions in order to have the character look at interesting objects in the environment. An artist could also keyframe (see Section 2.1.1) a set of example motions of a character changing their gaze. For instance, an artist could produce a small number of example motions where a character whose torso has been injured changes his gaze. These example motions could then be used to build a parametric gaze map for the way the character adjusts their gaze when injured.

While the work presented in this chapter provides a reliable method for adjusting the gaze of a character when the target is known, it does not tackle the equally important question of where a character's gaze should be directed at any point in time. Yet my reliance on psychological literature to develop a model for gaze could be extended to look at models for realistically determining when humans direct their gaze towards specific types of targets. By providing a method for decoupling gaze from full body motion, I hope that my work will inspire others to tackle the problems associated with gaze control.

Chapter 6

Parametric Motion Graphs

This chapter presents a new approach for controlling interactive human character using a novel example-based motion synthesis data structure called a *parametric motion graph*. Like other example-based data structures, parametric motion graphs provide easy authoring of high-quality motions, but they also supply the responsiveness, precise control, and flexibility demanded by interactive applications. A parametric motion graph describes possible ways to generate seamless streams of motion by concatenating short motion clips generated through blending-based parametric synthesis. As described in Section 3.5, blending-based parametric synthesis allows accurate generation of any motion from an entire space of motions, by blending together examples from that space. For instance, parametric synthesis can generate motions of a person picking up an item from any location on a shelf by blending together a small set of example motions. While neither seamless motion concatenation nor parametric synthesis is a new idea, by combining both techniques, parametric motion graphs provide accurate control through parametric synthesis and can generate long streams of high-fidelity motion without visible seams using linear-blend transitions.

In contrast to many other automated methods for representing transitions between motions (see Section 2.2), parametric motion graphs are highly structured, facilitating efficient interactive character control. The nodes of a parametric motion graph represent entire parametric motion spaces that produce short motions for given values of their continuously valued parameters. The directed edges of the graph encode valid transitions between source and destination parametric motion spaces. This structure efficiently organizes the large number of example motions that can be blended together to produce the final motion streams. Because of this structure, I have been able

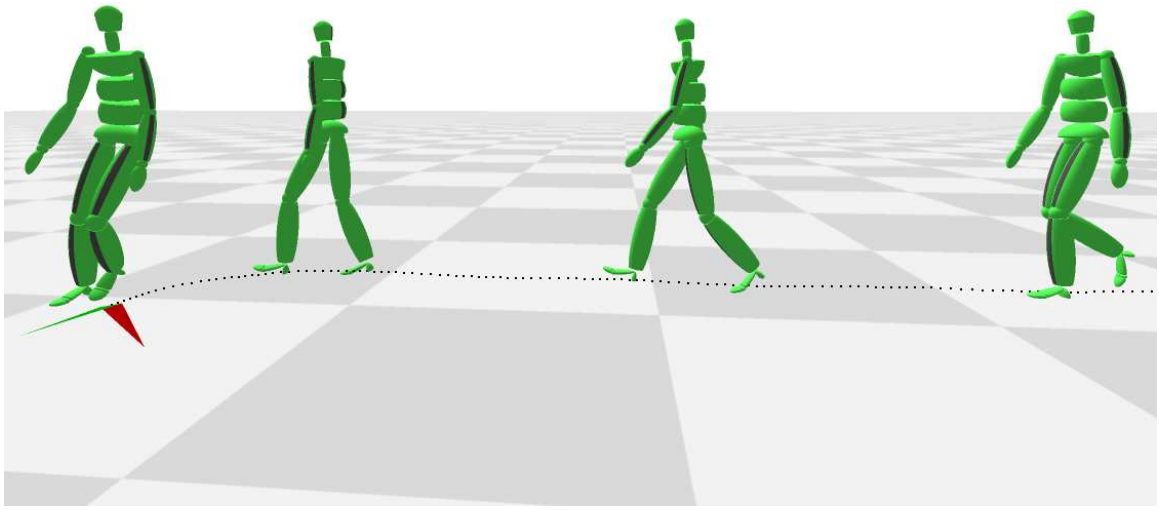


Figure 6.1 An interactively controllable walking character using parametric motion graphs to smoothly move through an environment. The character is turning around to walk in the user-requested travel direction, depicted by the red arrow on the ground.

to easily author interactively controllable characters that can walk, run, cartwheel, punch, change facing direction, and/or duck in response to user-issued requests.

While prior work on synthesis by concatenation has focused on representing seamless transitions between individual clips of motion (see Section 2.2.1), I face the problem of defining valid transitions between parametric *spaces* of motions, where it is not often possible to transition from any motion in one parametric motion space to any motion in another. For example, consider a parametric motion space representing a person taking two steps, parameterized on curvature. One can imagine that this parametric motion space can follow itself; a person can take two steps, and then take two more, and so on. However, a transition should not be generated between a motion where the character curves sharply to the right and another where the character curves sharply to the left; the resulting transition would not look realistic. Thus, the edges in a parametric motion graph must encode the *range* of parameters of the target space that a motion from the source space can transition to, as well as the correct way to make the transition between valid pairs of source and destination motions. The key challenge to parametric motion graphs is finding a good way to compute and represent these transitions. By approaching the problem from a sampling perspective, I provide an efficient way to compute and encode the edges of a parametric motion graph, allowing automated authoring and fast transition generation at runtime.

To provide parametric motion graphs as a method for interactive character control, this chapter describes how to:

Build Parametric Motion Graphs: Using a method based on sampling, I can efficiently locate and represent transitions between parametric motion spaces.

Extract Data from Parametric Motion Graphs: My representation of transitions allows fast lookup of possible transitions at runtime using interpolation.

Use Parametric Motion Graphs for Interactive Control: Because parametric motion graphs are highly structured, they facilitate the fast decision-making necessary for interactive character control. Furthermore, because all motion clips in the graph are generated using parametric synthesis, motions accurately meet relevant constraints.

The rest of this chapter is organized as follows. Section 6.1 details my methods for building and extracting information from a parametric motion graph. Then, Section 6.2 presents results from some experiments using parametric motion graphs, including controlling interactive characters in realtime. Finally, Section 6.3 concludes with a general discussion of the presented technique, including a number of the technique's limitations.

6.1 Parametric Motion Graphs

This section describes in detail the methods developed for building parametric motion graphs and extracting data from them. My methods for controlling a character using a parametric motion graph are presented later in Section 6.2.

6.1.1 Building a Parametric Motion Graph

To facilitate efficient motion synthesis at runtime, much of the needed computation for controlling interactive characters is done while building a parametric motion graph offline. A parametric motion graph only needs to be built once, resulting in a small text file representation of the graph that can be loaded at runtime.

As described at the beginning of this chapter, each node of a parametric motion graph represents a parametric motion space implemented using blending-based parametric synthesis. For all of the examples in this dissertation, blending-based parametric synthesis is performed using the method presented in Section 3.5. While the nodes of a parametric motion graph can be built using this existing technique, the key challenge is finding a way to identify and represent possible transitions between these parameterized nodes. Because the parametric motion spaces represented by the graph nodes are smooth, as described in Section 3.5, I can tackle this challenge using sampling and interpolation. The rest of this subsection describes in detail how to identify and represent edges between source and target graph nodes, $\mathbf{N}_s$ and $\mathbf{N}_t$ respectively. Throughout this description, the parametric motion space represented by node $\mathbf{N}_i$ is denoted by $\mathcal{P}^i(l)$, where l is a vector of relevant motion parameters, such as the target of a punch; a parametric motion space produces a short motion, $\mathbf{M}_i$, for any given value, l_i, of its continuously valued parameters.

6.1.1.1 Identifying Transitions Between Motion Spaces

To start, consider the case where the nodes $\mathbf{N}_s$ and $\mathbf{N}_t$ represent small motion spaces whose valid parameter ranges only include a single point. This case reduces to the traditional synthesis-by-concatenation problem; is there a frame of motion near the end of the motion generated by $\mathbf{N}_s$, $\mathbf{M}_1$, and a frame of motion near the beginning of the motion generated by $\mathbf{N}_t$, $\mathbf{M}_2$, that are similar enough to allow a linear-blend transition from one to the other over a short window centered at these frames? A good transition exists from $\mathbf{M}_1$ to $\mathbf{M}_2$ if and only if there exists a frame, t_1, near the end of $\mathbf{M}_1$ and a frame, t_2, near the beginning of $\mathbf{M}_2$ such that $\mathbf{D}(\mathbf{M}_1(t_1), \mathbf{M}_2(t_2)) \leq T_{GOOD}$, where T_{GOOD} is a tunable threshold. If the distance value of the optimal transition point found using the method presented in Section 3.3 is below T_{GOOD}, then it is possible to transition between $\mathbf{M}_1$ and $\mathbf{M}_2$ at that point, (t_o^1, t_o^2).

Now consider the general case where $\mathbf{N}_s$ and $\mathbf{N}_t$ represent larger spaces. For any sufficiently large space, it is unlikely that the motions represented by the space look similar enough to be treated like a single motion. For instance, in the walking example discussed at the beginning of this chapter, the walking character can only transition to other walking motions where the character walks at a similar curvature to its current one. However, since each parametric motion space represents an infinite number of motions, it is infeasible to compare all possible pairs of motions represented by each of the parameterized nodes. One possible approach is to reduce each parametric motion space to a discrete number of motions chosen from the full space. To find and represent good transitions between all pairs of motions from a source set of size m and a target set of size n, I would need to repeat the technique described above mn times. Unfortunately, by transforming a continuous motion space into a discrete set of motions, I lose much of the accuracy that parametric synthesis provides; accuracy can be increased by adding more motions to these sets but this results in a combinatorial explosion in the number of required comparisons and the amount of space needed to store the possible transitions.

Yet, in a smooth parametric motion space, motions generated for any local neighborhood of parameter space look similar. For example, consider a parametric motion space representing motions of a person punching, parameterized on the location of the punch. Two motions in this space

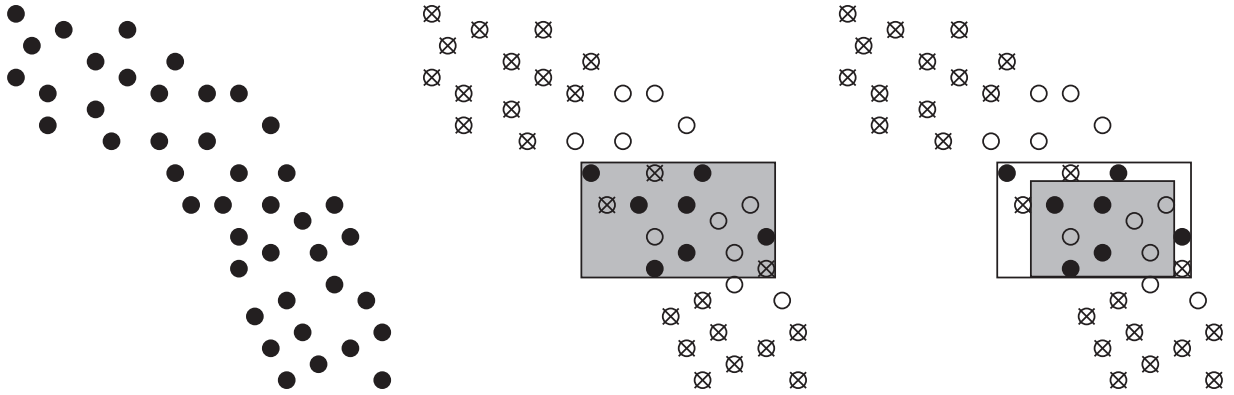


Figure 6.2 Process of determining the valid transition region in target parameter space. (a) A set of randomly chosen samples from the target space. (b) Darkened circles produce good transitions, crossed out circles produce bad transitions, and empty circles produce neutral transitions. The shaded box encloses all good samples but also includes some bad samples. (c) The adjusted, shaded box excludes all bad samples. In practice, little to no adjustment is usually made to the bounding box.

where the punches land $1mm$ apart look similar. In this case, I can compute the possible transitions from one of these motions and use the result for both. This observation leads me to approach the problem of identifying and representing transitions between parametric motion spaces using sampling, extending the method presented in Section 3.3 for locating possible linear blend transitions between individual motions.

6.1.1.2 Building a Parametric Motion Graph Edge

An edge between source and target nodes, $\mathbf{N}_s$ and $\mathbf{N}_t$ respectively, maps any point, l_i^s, in $\mathcal{P}^s$ to the subspace of $\mathcal{P}^t$ that can be transitioned to from $\mathbf{M}_i^s = \mathcal{P}^s(l_i^s)$. It also supplies the time at which that transition should occur. Assuming it is possible to transition from every point in $\mathbf{N}_s$ to some subspace in $\mathbf{N}_t$, we can build an edge between these nodes using sampling. I start by generating two lists of random parameter samples, $\mathbf{L}^s = \{l_1^s, \ldots, l_{n_s}^s\}$ and $\mathbf{L}^t = \{l_1^t, \ldots, l_{n_t}^t\}$ (see Figure 6.2a). In order to accurately capture the variations in the target space, n_t should be large. The exact number depends on the size of the parameter space, but I have found 1000 samples to be more than enough for all of the cases I have tried, even for parametric motion spaces that have three

parameters. In contrast, n_s should be small, while still covering the extremes of the source space, as this number affects the amount of storage needed for an edge as well as performance efficiency of the graph when used to produce motion at runtime (see Section 6.2.4). For the examples in this dissertation, n_s ranged from 4 to 200.

Now consider a sample from $\mathbf{L}^s$, l_1^s. This sample corresponds to the motion $\mathbf{M}_1^s = \mathcal{P}^s(l_1^s)$. I can determine if $\mathbf{M}_1^s$ can transition to each motion represented by the parameter samples in $\mathbf{L}^t$ by computing the optimal transition point with each motion $\{\mathbf{M}_1^t, \ldots, \mathbf{M}_{n_t}^t\}$ using the method presented in Section 3.3. Samples from $\mathbf{L}^t$ that produce good transitions are added to the list of parameter samples $\mathbf{L}_{GOOD}^t$.

Using the observation that motions close in parameter space look similar, I can assume that any parameter vector for $\mathcal{P}^t$ whose nearest parameter samples from $\mathbf{L}^t$ appear in $\mathbf{L}_{GOOD}^t$ can also be transitioned to from $\mathbf{M}_1^s$. Thus, the list $\mathbf{L}_{GOOD}^t$ defines the subspace of $\mathcal{P}^t$ to which $\mathbf{M}_1^s$ can transition.

Unfortunately, I cannot represent the subspace of $\mathbf{N}^t$ that can be transitioned to from $\mathbf{M}_1^s$ by listing the points in $\mathbf{L}_{GOOD}^t$ because, as described at the beginning of Section 6.1, I plan to determine what transitions are possible at runtime using a simple and efficient interpolation scheme (as shown in Figure 6.3); interpolating between potentially different numbers of uncorrelated points in a meaningful way is difficult, if not impossible. So, instead, I represent each subspace as a simple shape that can always be interpolated (i.e., bounding boxes, spheres, triangles). I have found axis-aligned bounding boxes work well for my data; I use axis-aligned bounding boxes to represent all of the transition parameter subspaces.

Using simple, easily interpolated shapes to represent transition regions introduces a considerable problem. Any simple shape that contains all points in $\mathbf{L}_{GOOD}^t$ could also contain other points from $\mathbf{L}^t$ that were not deemed good transition candidates (see Figure 6.2b). To guarantee that bad transitions are not included in the transition subspace of $\mathbf{N}^t$, I take a conservative, double threshold approach. First, while constructing the list $\mathbf{L}_{GOOD}^t$, I also form a list, $\mathbf{L}_{BAD}^t$, containing all samples from $\mathbf{L}^t$ that generate motions whose optimal transition point distance is greater than T_{BAD}, where $T_{BAD} \geq T_{GOOD}$. Next, I compute the bounding box of all parameter samples in $\mathbf{L}_{GOOD}^t$. Finally,

I consider each sample in $\mathbf{L}^t_{BAD}$; if the sample falls within the subspace defined by the bounding box, I make the minimal adjustment to the dimensions of the bounding box so that the sample falls at least ϵ away, where $\epsilon > 0$. In this way, I construct a bounding box that contains many, if not all, of the samples from $\mathbf{L}^t_{GOOD}$ without including any of the samples from L^t_{BAD}. Neutral samples from $\mathbf{L}^t$ whose optimal transition point distance falls between T_{GOOD} and T_{BAD} are considered good enough if they fall within the transition subspace of $\mathbf{N}^t$ but will not be explicitly included in the space (see Figure 6.2c). In practice, the system makes few bounding box adjustments to remove bad samples and in most cases makes none at all.

I also compute a single transition point from $\mathbf{M}^s_1$ to any of the motions located in the subspace of $\mathbf{N}^t$ defined by the computed bounding box. In Section 3.3, I described the optimal transition point of two motions as the pair of frames where the two motions are most similar. For computing a generic transition point for the entire subspace, it is useful to normalize these frame numbers to the range 0 to 1. Again, because nearby motions in a motion space look similar, the optimal transition points are likely to be at similar normalized times. So, I average the normalized optimal transition points for each sample of $\mathbf{L}^t_{GOOD}$ that falls inside the adjusted bounding box to calculate the normalized transition point for the subspace.

Putting all the pieces together, an edge can be defined between $\mathbf{N}^s$ and $\mathbf{N}^t$ as a list of transition samples, one for each parameter vector in $\mathbf{L}^s$. Each sample includes:

- The value of the parameter vector l^s_i
- The computed transition bounding box for l^s_i
- The average, normalized transition point for l^s_i

I could also store the average alignment transform between the motion $\mathbf{M}^s_i$ and each of the motion samples in $\mathbf{L}^t_{GOOD}$ but recomputing this alignment using the method presented in Section 3.3 is fast; instead I save storage space by computing the alignment transform for each transition at runtime.

Up until this point, I have assumed that I can transition from every point in $\mathbf{N}_s$ to some subspace of $\mathbf{N}_t$. I define that a transition exists between nodes $\mathbf{N}_s$ and $\mathbf{N}_t$ if and only if for any motion

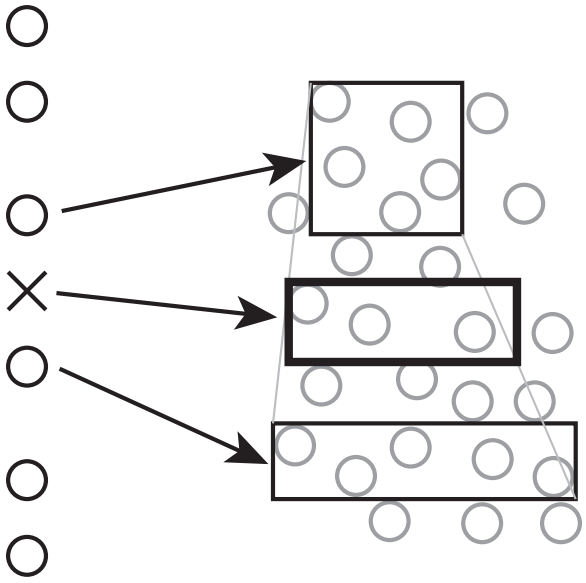


Figure 6.3 Mapping a parameter vector, depicted by the X, from the 1-D parameter space on the left, to a valid transition region in the 2-D parameter space on the right. X's bounding box is the weighted average of the bounding boxes for its 2-nearest neighbors.

contained in $\mathbf{N}_s$ there exists *some* subspace in $\mathbf{N}_t$ that it can transition to. Thus, if I find any sample in $\mathbf{L}^s$ whose adjusted bounding box is empty, I cannot create an edge between $\mathbf{N}_s$ and $\mathbf{N}_t$

6.1.2 Extracting Data from a Parametric Motion Graph

Synthesizing motion using a parametric motion graph is quick and efficient. The data that is stored in each node of the graph allows fast lookup for possible transitions. In particular, given the node, $\mathbf{N}_s$, and relevant parameter vector, $\tilde{l}^s$, for a motion clip, I can determine what subspaces of other parametric motion spaces can be transitioned to as well as when that transition should occur.

For each outgoing edge of $\mathbf{N}_s$, begin by finding the k-nearest neighbors to $\tilde{l}^s$ from the transition sample list, in terms of Euclidean distance, where k is normally one more than the number of dimensions of $\mathcal{P}^s$. Call these neighbors $l^s_1, \ldots, l^s_k$, ordered from closest to farthest from $\tilde{1}^s$. Following the work of Allen et al. [ACP02] on skinning human characters using k-nearest neighbor interpolation and on Buehler et al.'s work on rendering lumigraphs using k-nearest neighbor

interpolation [BBM+01], each l_i^s is associated with a weight, w_i:

$$w_i = \frac{w_i'}{\sum_{j=1}^{k} w_j'} \quad [6.1]$$

$$w_i' = \frac{1}{\varepsilon(\tilde{l}^s, l_i^s)} - \frac{1}{\varepsilon(\tilde{l}^s, l_k^s)} \quad [6.2]$$

where ε gives the Euclidean distance between parameter samples. This method of determining weights has two relevant advantages over using a linear map constructed as a best fit optimization over weights and motion parameters. First, this method does not introduce large negative weights. Using a linear fit method, these large negative weights often appear in order to artificially produce a better fit, even though the quality of the results suffer. Second, computing weights using this algorithm is fast as it does not require a costly global optimization and scales well with the number of example motions in the database.

For any outgoing edge of $\mathbf{N}_s$, calculate the subspace of the target node, $\mathbf{N}_t$, that can be transitioned to, $\mathbf{B}(\mathbf{N}_s, \mathbf{N}_t)$, as follows:

$$\mathbf{B}(\mathbf{N}_s, \mathbf{N}_t) = \sum_{i=1}^{k} w_i * \beta(l_i^s) \quad [6.3]$$

where $\beta(l_i^s)$ gives the value of the bounding box for the sample l_i^s, represented by the location of the box's center and its width in each dimension, as stored in the edge (see Figure 6.3). Similarly, compute the normalized transition point as a weighted sum of the average, normalized transition points for each l_i^s stored in the edge.

6.2 Results

This section provides details for some of the example parametric motion graphs I designed for interactive character control. Following the description of these graphs, I present the results of a number of experiments for testing the usefulness of these graph structures in interactive applications.

Graph Name	# of Nodes	# of Edges	# of Example Motions
Walking	1	1	44
Running	1	1	198
Cartwheeling	1	1	10
Walking and Running	2	4	242
Many Everyday Actions	7	14	256
Boxing	3	9	275

Table 6.1 Size and make-up of the parametric motion graphs in this dissertation. Each line provides the name of the parametric motion graph, the number of nodes in that graph, the number of edges connecting those nodes in the graph, and the total number of example motions organized by the graph.

6.2.1 Graphs

I have constructed six different parametric motion graphs in order to show the utility of the technique. These graphs are described throughout this section. Refer to Table 6.1 for a summary of the size of these graphs in terms of number of nodes, edges, and example motions.

The process of building parametric motion graphs is highly automated. An author starts by choosing the parametric motion spaces needed for the graph from an available motion space database built using the blending-based parametric synthesis technique described in Section 3.5. These parametric motion spaces then appear as disconnected nodes in the graph.

Next, the author chooses two nodes to generate an edge between and specifies values for $\mathbf{T}_{GOOD}$, $\mathbf{T}_{BAD}$, n_s, and n_t. While it is possible to set the values of $\mathbf{T}_{GOOD}$ and $\mathbf{T}_{BAD}$ without user input, the ability to adjust these values allows an author to determine where to set the tradeoff between motion quality and flexibility discussed later in this section. In practice, it took two or three iterations in order to tune the parameters $\mathbf{T}_{GOOD}$ and $\mathbf{T}_{BAD}$ for each edge. Empirically, setting $\mathbf{T}_{GOOD}$ to $.5$ and $\mathbf{T}_{BAD}$ to $.7$ served as a good starting point. For my example graphs, the amount of time it took to generate a single edge varied from $2 - 147$ seconds, depending on the complexity of the source and target parametric motion spaces.

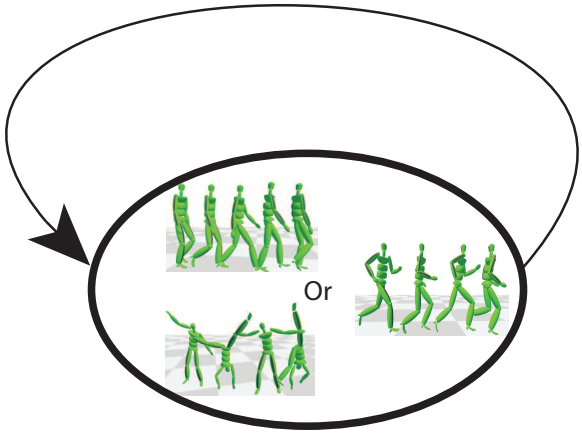


Figure 6.4 Graph for walking, running, or cartwheeling.

6.2.1.1 Single Node Locomotion Graphs

While other researchers have dealt specifically with generating controllable streams of locomotion in realtime (see Section 2.2.4 for a review of these methods), I chose to create several single-node locomotion graphs because it is easy to see artifacts in this commonly performed activity. In my first graph, I encoded streams of walking motion that only contain smooth turns. This graph consists of a single node representing a parametric motion space of a character walking for two steps at different curvatures. The parametric motion space maps the angular change in the character's travel direction from the beginning to the end of the motion (between -131 degrees and 138 degrees) to synthesized motions. Similarly, I built a running graph as a single node representing a parametric motion space with a valid angular travel direction change between -120 degrees and 99 degrees.

Since my technique requires little authoring effort, it is possible to experiment with non-obvious motions. I also built a parametric motion graph that encodes locomotion control through cartwheeling. Like the graphs for walking and running, my cartwheel locomotion graph contains only a single node. This node represents a parametric motion space of a character doing a cartwheel, rotating towards the right by varying amounts on one foot, and then doing a cartwheel

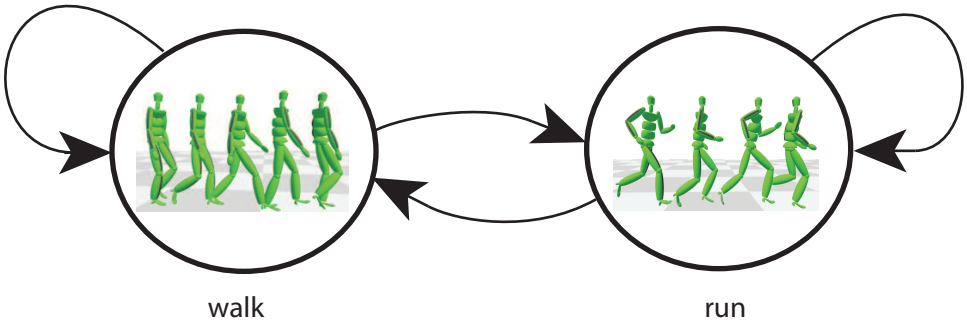


Figure 6.5 A locomotion graph for walking and running.

in another direction. Again, the parametric motion space maps the angular change in travel direction of the character from the beginning of the motion to the end (between -13 degrees and 157 degrees) to synthesized motions.

Each of these single node locomotion graphs take less than 5 minutes to build from beginning to end using my unoptimized system.

6.2.1.2 General Graphs

In addition to single-node locomotion graphs, I have also built several larger graphs. The simplest is a two-node graph that combines the walking and running nodes described earlier (see Figure 6.5). This graph can control the travel direction of a character that can both run and walk.

I have also built a seven-node, fourteen-edge graph containing motions for a number of different everyday actions: walking and running at different curvatures, sitting down and standing up from chairs of heights between 1ft and 1.9ft tall, stepping up onto and stepping off of platforms of heights between $.8$ft and 1.8ft tall, and leaping over distances between 2 and 3ft (see Figure 6.6). It takes about 11 minutes to build this graph. The final graph organizes a total of 256 example motions so that they can be blended to produce continuous streams of controllable animation.

In order to show that my technique works when controlling a number of different non-locomotion actions, I built a parametric motion graph that encodes the motions of a boxer punching, ducking, and "dancing" from one foot to the other. The boxing graph consists of three nodes. The first node represents all motions of a boxing character punching to some location in a 6ft wide, 2ft tall, and 5ft

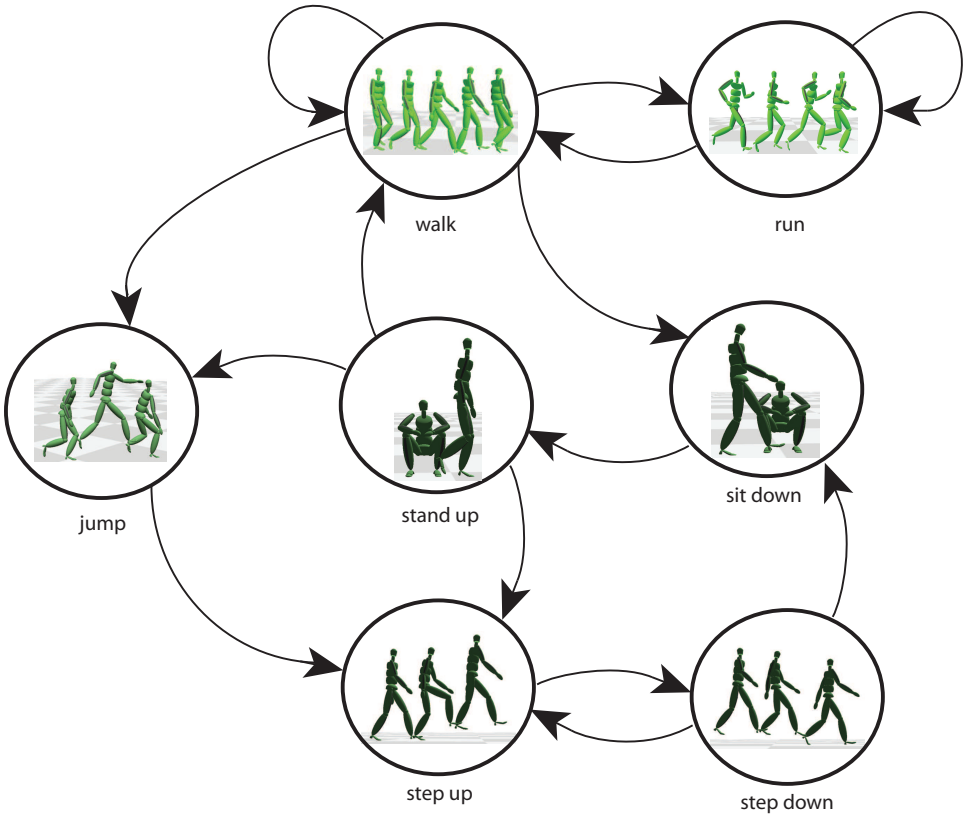


Figure 6.6 A graph for controlling a number of different everyday actions.

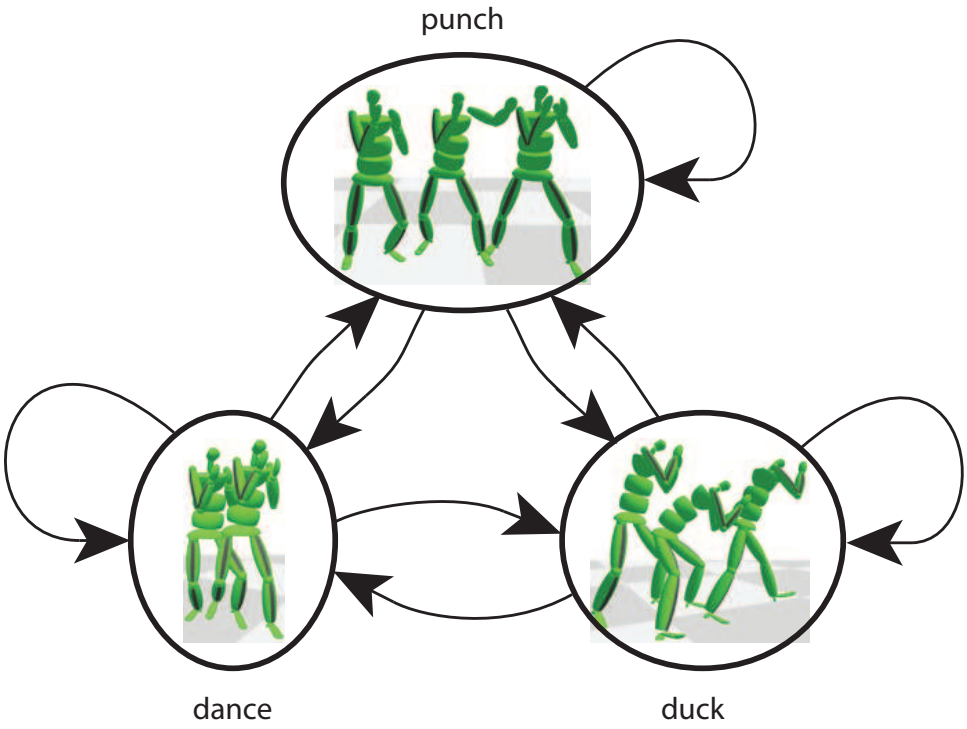


Figure 6.7 A boxing graph.

deep space. The parametric motion space maps desired punch locations in relation to the starting configuration of the root to synthesized punching motions. The second node of the boxing graph represents motions of a boxing character ducking below different heights (between 3.4ft and 5.6ft from the ground) and is parameterized on how low the character ducks. The third and final node encodes motions of a character "dancing" from one foot to another while maintaining a boxing ready stance. When "dancing", the character rotates by different amounts (between -27 and 46 degrees). Thus, the "dancing" motion space maps the change in facing direction from the beginning of the motion to the end of the motion to synthesized "dancing" motions. In total, the parametric motion spaces used for these graph nodes blend between 275 different motion-captured examples. A discrete motion transition graph, like those described in Section 2.2.2 and Section 2.2.3, that represents transitions between this number of motions would be large and unwieldy. In contrast, the final parametric motion graph (Figure 6.7) contains only nine edges, one connecting every pair of nodes. It takes approximately 7 minutes and 40 seconds to build the graph.

6.2.2 Applications

I implemented a number of different applications to test the usefulness of my technique. In this section, I describe these applications in detail and provide an overview of my results.

6.2.2.1 Random Graph Walks

My first application shows that parametric motion graphs can generate *seamless, high-fidelity motion streams in realtime*. For each of the graphs described in Section 6.2.1, I can produce a random stream of motion by taking random walks on the graph.

I start by choosing a random node and parameter vector from the graph. When the parametric motion space associated with the node is supplied with the chosen parameter vector, I can render a motion that matches this parameter request in realtime using the method presented in Section 3.5. While playing the motion, when I reach the possible transition region, I randomly choose an edge from those leaving the current node. The node that this edge points to is the new target node. Using the method described in Section 6.1.2, I compute the optimal transition point and the parameter

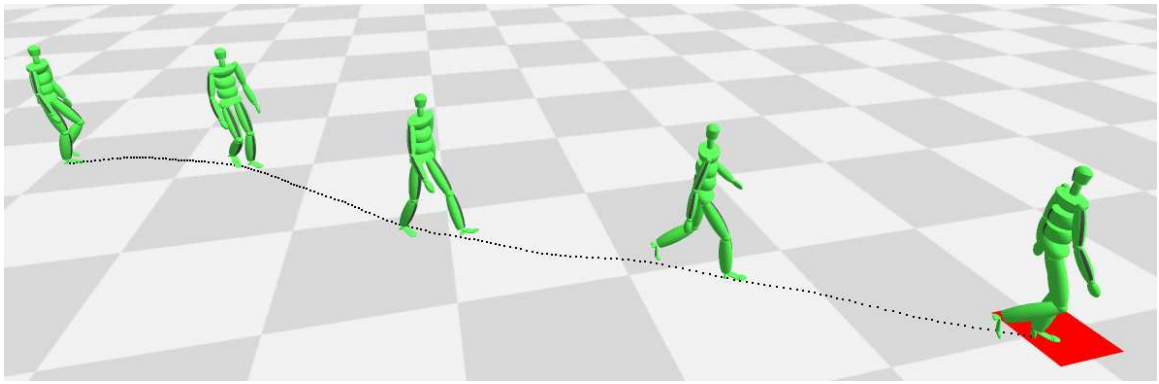


Figure 6.8 Using parametric motion graphs, this character walks to a specified location, depicted by the red square on the ground. The path on the ground plane is not prespecified. It is shown only to illustrate the path the character takes to the target.

subspace of the target node that I can transition to from my current parameter vector. I then randomly choose a new target parameter vector enclosed in this subspace. Finally, when I reach the blending window centered at the optimal transition point, I can append my current motion to my newly chosen motion using a linear blend transition, as described in Section 3.3. This process is then repeated indefinitely to produce an infinitely long stream of motion.

By randomly generating long streams of motion, I can confirm that my technique produces continuous motions and avoids poor transitions. I can also show that the algorithm for synthesizing new motion with a parametric motion graph is efficient enough to be used in an interactive application.

6.2.2.2 Target Directed Control

My second application tests whether my walking character can *accurately* reach a target location using a greedy graph search similar to ones used for locomotion control [SMM05] and crowd control [SKG05]. For this application, I generate a motion stream in the same way as for random graph walks (see Section 6.2.2.1), except that when it is time to choose a new parameter vector from the target bounding box, I choose the parameter vector that best adjusts the character's travel direction towards a target. Figure 6.8 shows that the walking character is able to accurately reach a target location without wandering by using this simple control algorithm.

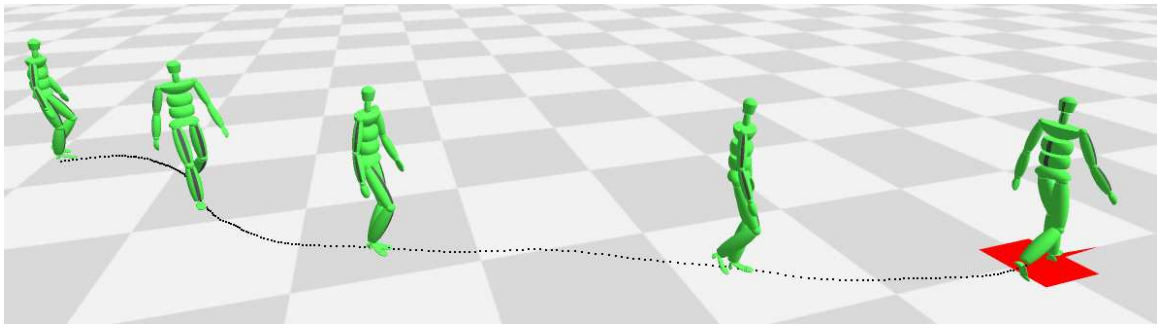


Figure 6.9 Using parametric motion graphs, this character walks to a specified location, and arrives while oriented in the requested direction. The red box and arrow on the ground depict the desired location and orientation respectively. The path on the ground plane is not prespecified. It is shown only to illustrate the path the character takes to the target.

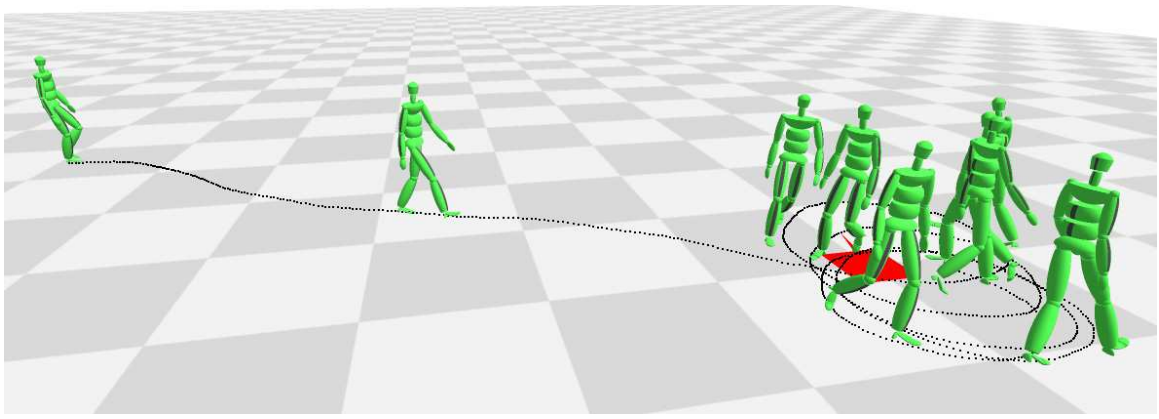


Figure 6.10 Using parametric motion graphs, this walking character cannot arrive at the specified location while oriented in a particular direction. The turning radius of the character is too small.

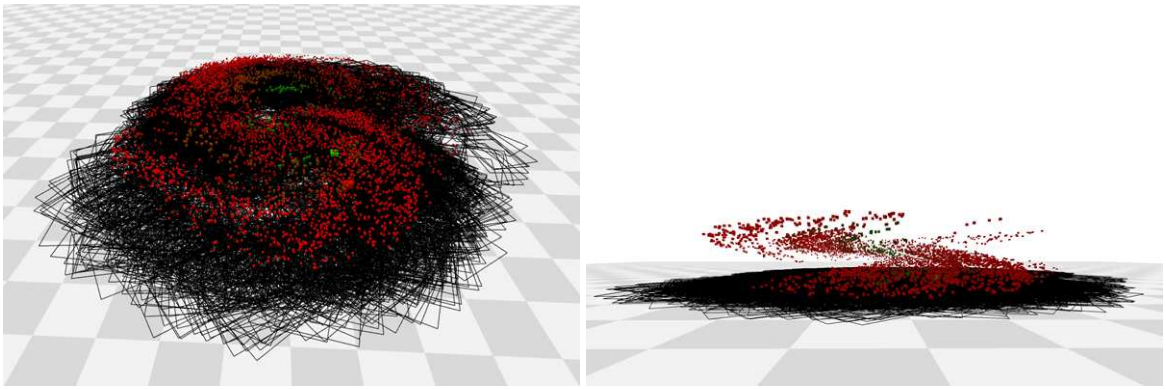


Figure 6.11 Locations that the walking character can reach in a short period of time. Height corresponds to possible orientations that the character can be in when it reaches the location. This experiment shows that most locations on the ground plane can be reached by the walking character but that there are only a small number of orientations in which the character can be in when they arrive at each of these locations.

I also allow a user to request that the character reach the target location oriented in a particular direction. For this case, I choose the parameter vector that both adjusts the character's travel direction towards the target and orients the character towards the desired facing direction. I place more weight on the orientation component of this optimization function as the character gets closer to the target. In several cases, the walking character can perform the requested action well (see Figure 6.9). But I find that in others, the character approaches the target and then turns in circles trying to orient itself (see Figure 6.10). This result is anticipated as I know that the character's minimum turning radius is quite large.

Inspired by the work of Reitsma and Pollard [RP04, RP07], I used a discrete, brute force method to embed the walking parametric motion graph in the environment in hopes of better understanding this problem. The embedding made it clear that the walking character could meet location constraints within a reasonable radius but that for most locations, there were only a few orientations that the character could be in when they arrived. Figure 6.11 shows the results of this embedding.

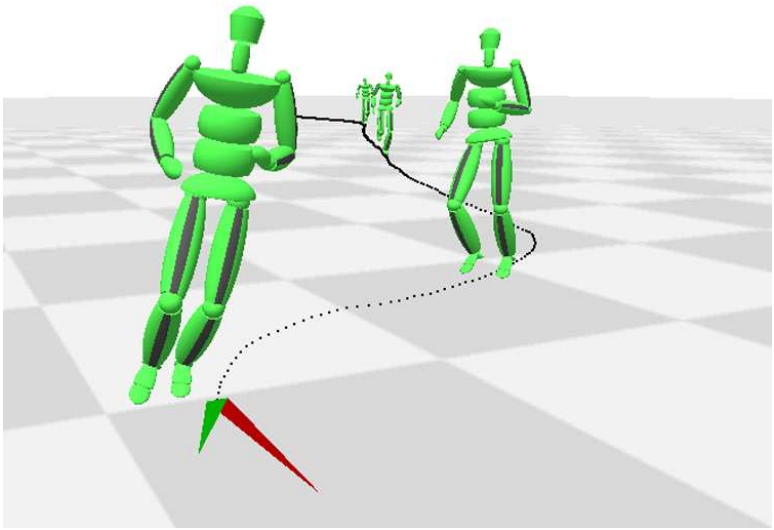


Figure 6.12 An interactively controllable running character using parametric motion graphs to smoothly move through an environment. The character has changed running direction in order to travel in the user-requested direction depicted by the red arrow.

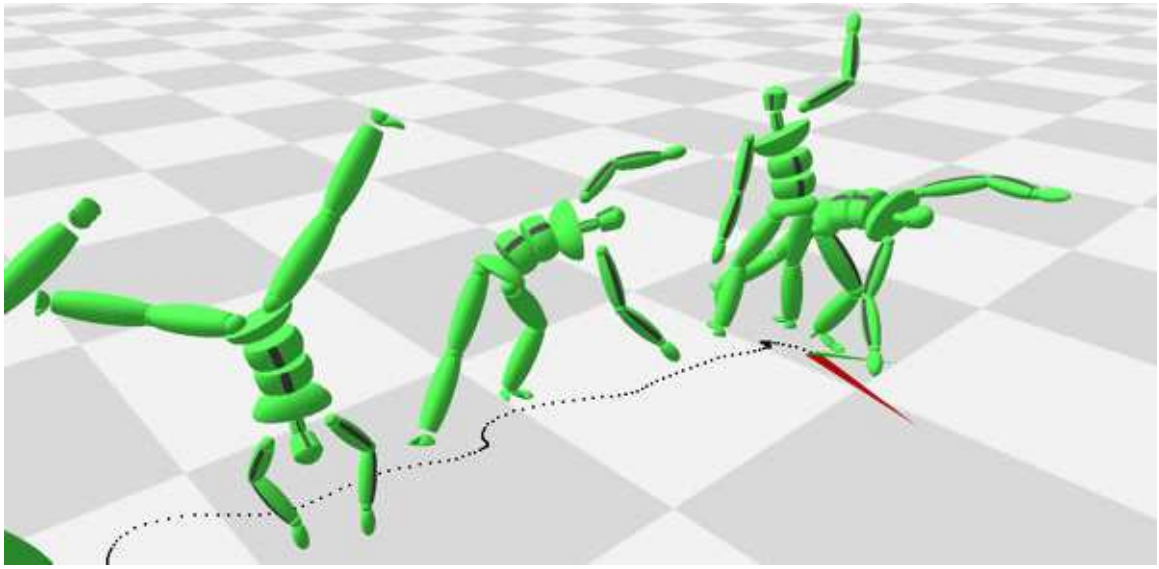


Figure 6.13 An interactively controllable cartwheeling character using parametric motion graphs to smoothly move through an environment. The character has changed cartwheeling direction in order to travel in the user-requested direction depicted by the red arrow.

6.2.2.3 Interactive Character Control

My last and most important application allows users to interactively control a character, testing all of the necessary characteristics of interactive applications (see Section 1.1). To do this, I attach a function to each node that translates user requests to parameters. For example, for walking and cartwheeling, I wanted a user to control the travel direction of the character by specifying the desired travel direction using a joystick. So, I attached a function to each of these nodes that could compute the angular change between the character's current direction of travel and desired direction of travel.

With these translation functions in place, I can again generate motion streams as I did when generating random graph walks (see Section 6.2.2.1) except that when it is time to choose a parameter vector from the target bounding box, I query the user's current request. Then I use the translation function for the requested node to compute a parameter vector. These parameter values are adjusted so that they fall within the target bounding box if they were not within bounds already.

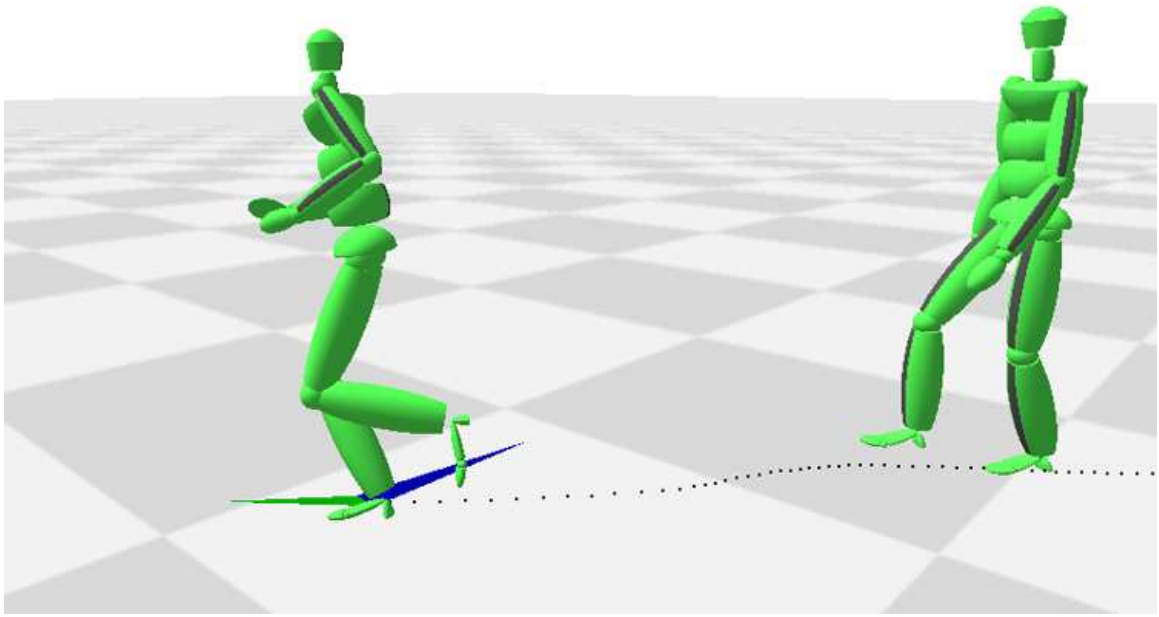


Figure 6.14 An interactively controllable character using parametric motion graphs to smoothly move through an environment by walking or running. The character has just transitioned from walking to running, and is now changing his travel direction in order to meet the user-requested direction depicted by the blue arrow.

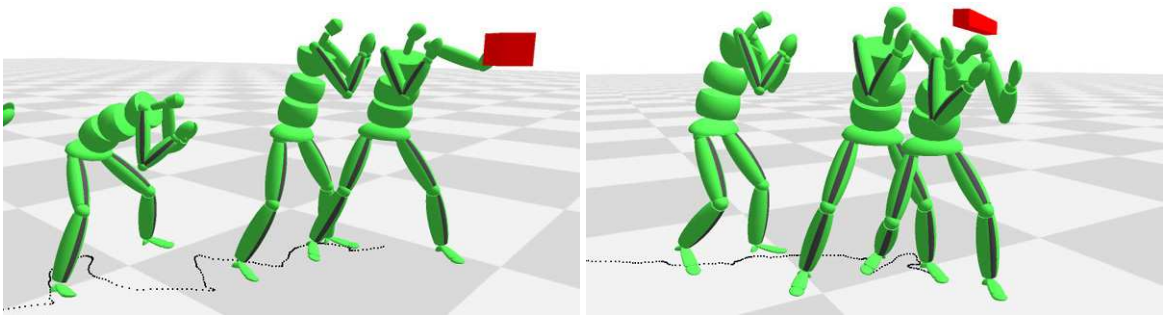


Figure 6.15 An interactively controllable boxing character that uses parametric motion graphs. The character is punching towards a user-requested target in the top image. In the bottom image, the character is ducking below a user specified height.

This process has the effect of creating interactive characters that perform requested actions as accurately as possible without introducing poor transitions between motion clips. By limiting the transitions to good ones, our characters occasionally miss targets; in these cases, the character still "reacts" to the target by choosing a good transition that gets closest to meeting the request. For instance, the boxing character shown in Figure 6.15 occasionally misses its punching target when the target appears on the periphery of the region in front of his body. This is because it is not possible to hit the target without producing a bad transition. So, instead, the character will rotate his body as much as possible and punch near to the target, thus "reacting" to the request but not quite meeting it.

Using this technique, I have produced:

1. a *walking* character whose travel direction can be controlled (see Figure 1.5b).

2. a *running* character whose travel direction can be controlled (see Figure 6.12).

3. a *cartwheeling* character whose travel direction can be controlled (see Figure 6.13). Note that the cartwheeling character is an interesting one because the character only knows how to turn to the right. So, when the character is asked to turn to the left, he makes two large right hand turns. This reaction is not "programmed" into the control structure, instead it happens naturally because of the way parametric motion graphs work.

4. a character who can either *run or walk* in a desired travel direction (see Figure 6.14).

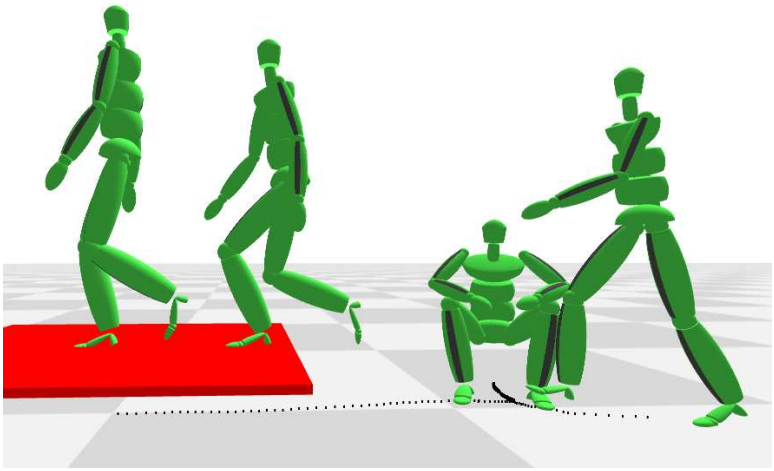


Figure 6.16 An interactively controllable character using parametric motion graphs. The character has just stepped up onto a platform after sitting down in a chair.

5. a *boxing* character that is able to change facing direction while "dancing", punch towards specified 3D locations, and duck below a specified height (see Figure 6.15).

6. a character that can perform *everyday actions* - walking or running in a desired direction, stepping onto and off of platforms, sitting down and standing up from chairs, and leaping over distances (see Figure 6.16)

6.2.3 Comparison with Fat Graphs

Parametric motion graphs are similar to another method for constructing structured motion graphs called fat graphs [SO06]. As described in Section 2.2.3, a fat graph is constructed by first identifying key poses within a motion database that appear many times. These poses are then represented as "hub" nodes within the graph. The edges represent parametric motion spaces of motions that can transition from the same two key poses. Like parametric motion graphs, this structure explicitly combines parametric synthesis with synthesis-by-concatenation methods to produce a structured representation of motion transitions that can be used efficiently at runtime to accurately control a human character.

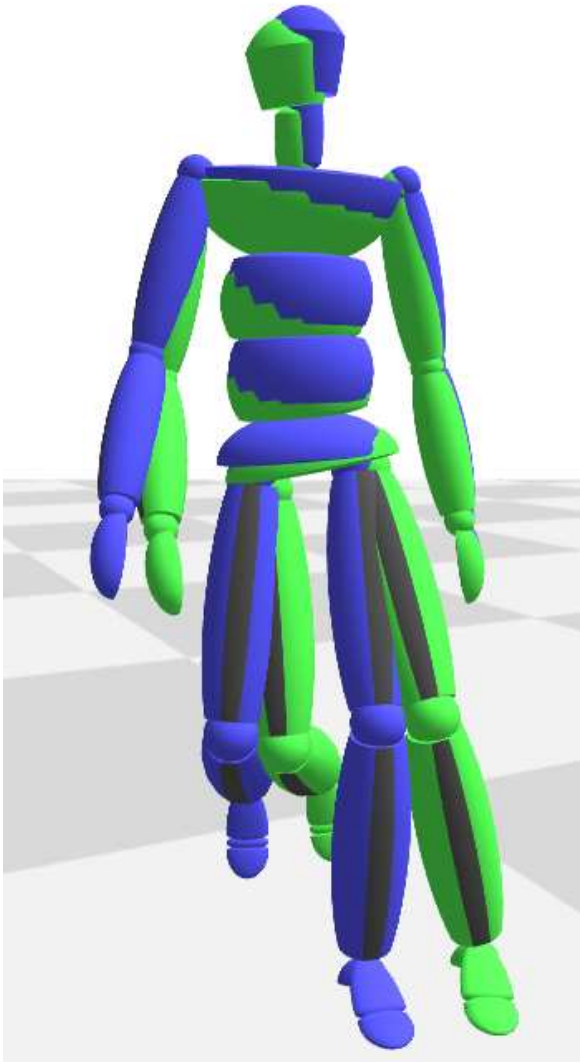


Figure 6.17 At the transition point between two motions of a character turning towards the right, the character using a parametric motion graph remains leaning into the turn (as shown in green) while the character using a fat graph must return to the common transition pose with no lean (as shown in blue), causing the character to “bob” as it goes around the turn.

Yet, parametric motion graphs have a number of advantages over fat graphs. Because all motions representing the same logical action, such as walking or dodging, are explicitly grouped together, parametric motion graphs provide additional logical structure to the graph. A graph author can easily see the logical connections between motion types, allowing the graphs to be designed easily for specific applications.

Parametric motion graphs also represent continuously changing transition points and ranges within a single type of motion. Like Snap-Together Motion, the technique fat graphs are based on (see Section 2.2.3), a fat graph must use more than one "hub" node in order to capture some of this complexity. For instance, in a fat graph representation, a walking parametric motion space might be divided across three parametric motion spaces in order to avoid transitioning from a sharp righthand curvature walking motion to a sharp lefthand curvature walking motion: one where the character is curving a lot towards the right, one where the character is curving a lot towards the left, and one where the character is curving mostly forward. And even if these motions are grouped into these three separate parametric motion spaces, each motion within a single parametric motion space will have exactly the same possible target transition motions. Parametric motion graphs represent continuously changing transition points using sampling.

Fat graphs are also limited in the quality of their results by the use of "hub" nodes; motions are constantly forced to return to the same average pose at each transition point. For instance imagine a character who is walking at a curvature that is very sharp to the right; when that character reaches the "hub" node, he must transition to the average pose of all of the walking motions represented by the parametric graph edge, even if the character continues to walk at a curvature that is very sharp to the right. By forcing motions to return to an average pose at "hub" nodes, motion streams often exhibit repetitive "bobbing" artifacts, as illustrated in Figure 6.17. On the other hand, parametric motion graphs handle natural variations in the transition poses of similar motions.

6.2.4 Algorithm Performance

In this section, I describe how parametric motion graphs perform in each of the six categories described in Section 1.1.

Node Name	Mean Clip Length	Min Clip Length	Max Clip Length
Walking	1.6s	1.4s	1.8s
Running	1.0s	0.8s	1.1s
Cartwheeling	2.9s	2.5s	3.5s
Stepping Up	1.9s	1.9s	2.0s
Stepping Down	1.9s	1.8s	1.9s
Sitting Down	4.8s	4.3s	5.3s
Standing Up	3.0s	2.9s	3.1s
Jumping	1.5s	1.4s	1.6s
Punching	1.1s	0.5s	1.9s
Ducking	1.6s	1.1s	2.6s
Boxing Dance	0.9s	0.6s	1.1s

Table 6.2 Example motion length information for each parametric motion space. Each line in the table contains the name of the parametric motion space, the average length of a motion clip in the parametric motions space, the minimum length of a motion clip in the parametric motion space, and the maximum length of a motion clip in the parametric motion space.

Efficient Synthesis The examples in this paper were computed on a laptop computer with a $1.75GHz$ Pentium M Processor, $1GB$ of RAM, and an ATI Mobility Radeon X300 graphics card. All of the generated motions were sampled at $30Hz$. Each of the generated parametric motion graphs can synthesize and render streams of motion at more than 180 frames per second, consistently. Because I use k-nearest neighbor interpolation and the weighting algorithm presented in Section 6.1.2, synthesis times are nearly independent of the number of example motions in the database. It should be noted however that computation time is dependent on the number of examples being blended together. Because Kovar and Gleicher's method for blending-based parametric synthesis [KG04] limits the number of motions that can be blended together at each of the sample points in a parametric motion space, this time is effectively bounded.

Efficient Data Storage Even for my largest graph, it is possible to store the graph's structure and edge information in a plain text file requiring less than $50KB$ of space. The storage required for this graph structure scales linearly with n_s, or the number of samples from the source motion space (see Section 6.1.1.2). Thus, it is useful to keep n_s small. Since it is unnecessary to densely sample the source nodes, in general n_s can remain low; however, n_s scales exponentially in relation to the dimensionality of the space being sampled [1]. Though, because blending-based parametric synthesis requires a more densely sampled space, it will fail due to high-dimensionality well before the limit of n_s is met for a parametric motion graph. In Section 6.3, I will discuss a possible way to alleviate the dimensionality problem using decoupling methods.

Low Latency or Response Time In terms of responsiveness, my method is limited by my ability to transition between motions only at one point near the end of a clip. Similarly, I do not adjust the parameter vector while generating a motion. These limitations mean that for motion spaces that represent long motions, it may take time for the character to react to

[1] This phenomenon of an exponentially increasing volume caused by adding dimensions to a mathematical space is a well studied problem in statistics. It is often referred to as the *curse of dimensionality*, a term coined by Richard Bellman [Bel57]

user requests. This problem can be lessened by choosing parametric motion spaces that represent short motion clips, as the maximum length of an example clip acts as an upper bound on the amount of time that the user must wait for a transition to a new motion (see Table 6.2 for detailed timing information about the length of the example clips used for this dissertation). Another possible way to improve the response time for some motions is to use a representation of blending-based parametric motions that is specifically designed for continuous, parameter vector changes, such as that in [TLP07].

I advocate improving response times for motions that are necessarily longer in natural ways. For instance, while it may take time for a human to begin accelerating, a gaze cue, such as those that can be generated using my technique in Chapter 5, can be used to indicate that the character is about to start moving; a simple change of gaze can easily be perceived as a response to a motion request.

Accurate Motion Generation and Visual Quality Because the motion clips in a parametric motion graph are generated using existing blending-based parametric synthesis methods, it is possible to produce motion clips that are of high-quality and that accurately meet user requests. However, a tradeoff exists between the quality of the transitions produced between motion clips and the accuracy of those clips. By setting T_{GOOD} to be high, synthesized motions are more likely to be accurate since the graph edge will allow a much larger range of transitions, but this might cause the transitions to look less good. For instance, by setting T_{GOOD} high when building the walking parametric motion graph, it might be possible for a character to transition from any walking motion to any other, allowing very fast changes in curvature. The synthesized motion streams would react accurately to each request for a curvature change as all curvature changes are possible, but the motion would look unrealistic during the transition. By allowing a user to set T_{GOOD} and T_{BAD} manually, I allow the user to explicitly manage this tradeoff.

Automated Authoring The process of authoring a parametric motion graph is highly automated. As described at the beginning of this section, an author must simply choose which existing

parametric motion spaces to connect together, set the tunable thresholds, T_{GOOD} and T_{BAD}, for determining the tradeoff between motion quality and accuracy, and decide how many samples to take from the source and target nodes. It is also possible for my system to identify all possible links between chosen parametric motion spaces automatically, but in practice, the information an author supplies about which types of motions can transition to which other types of motions is invaluable for minimizing the complexity of the graph, allowing more efficient synthesis of controllable motion streams.

6.3 Discussion

As presented, parametric motion graphs are able to produce seamless, controllable motion streams in realtime. The authoring process is highly automated, making parametric motion graphs useful for interactive applications that would not normally have the resources to build the structures necessary for accurate character control.

While I use the method of Kovar and Gleicher [KG04] to produce parametric motion spaces, my methods do not require that motions be generated with any particular parametric motion synthesis method. However, parametric motion graphs do require smooth parametric motion spaces (see Section 3.5); my sampling and interpolation methods depend on nearby motions in parameter space looking similar (see Section 6.1.1). While I have not provided an example, my method should work just as well using a procedural parametric synthesis method, as long as it produces smooth motion spaces.

One limitation to the technique in this chapter is that it cannot represent transitions between two nodes if there is any motion in the source node that cannot transition to the target node (see Section 6.1.1.2). For example, consider two nodes that represent a person walking at different curvatures where the first allows a much wider range of curvatures than the other. Because the extreme motions of the first node do not look like any of the motions in the second node, I will be unable to create an edge between the nodes.

One possible solution to the problem of building edges between partially compatible nodes is to dynamically add additional nodes to the graph when large enough continuous pieces of a source

node can transition to the target node. This new node would represent the same parametric motion space as the first except that its range would be limited to the range of parameters that have valid transitions to the target node. This solution has the drawback of adding greater complexity to the parametric motion graph, a characteristic that should be avoided in order to facilitate fast decision making at runtime. But there may be a way to balance the tradeoff between graph complexity and overall transition representation.

Another extension to this work that could lead to better methods for interactive control is to develop better local search methods than the greedy one in Section 6.2.2.2. Planning a long motion that consists of a series of motion clips using a parametric motion graph is at the moment an unexplored area of research. This lack of planning is in part designed into parametric motion graphs; the ability to make quick decisions without planning allows the graph to react to changing user requests efficiently and often. However, in some circumstances, it could be useful to know how to reach a specific motion outcome within the graph that requires synthesizing multiple motion clips before the outcome can be reached.

This chapter shows that motions for interactive characters can be designed in an automated way, allowing fast, accurate, high-fidelity motion generation in realtime. My method gains the benefits of accurate motion generation using parametric synthesis as well as the ability to make good transitions between clips using a continuous representation of transitions between parameterized spaces of motion. This technique can decrease the amount of time it takes to author interactive characters, increase the accuracy and efficiency of these characters at runtime, and provide high-fidelity motion in a reliable way.

Chapter 7

Discussion

The problem of synthesizing quality human motion is a difficult one not only because of the intricacies of the human body but also because humans are good at being able to recognize when a motion is flawed. For interactive applications, the problem becomes even harder; motions must necessarily be generated under much tighter processing and storage requirements while at the same time accurately meeting the ever-changing requests of a user. Example-based methods for human motion synthesis have aided the goal of producing natural looking, accurate motions; these techniques use nuanced example motions captured in a motion capture studio to synthesize new motions that are as nuanced as the originals. However, most of these example-based synthesis methods are not used in practice due to two drawbacks:

1. the exponentially large number of example motions needed to control many different motion parameters simultaneously

2. the processing time and latency necessary for synthesizing a stream of controllable motion

This dissertation has addressed both of these problems by decreasing the number of example motions needed for motion clip synthesis through parameter decoupling and by increasing synthesis and response time through motion clip organization. Taking a practical approach, I focused on developing methods for decoupling commonly used motion parameters and automatically constructing highly structured control mechanisms for realtime character control. Specifically, I made the following technical contributions:

1. **Runtime Method for Splicing Upper-Body Actions with Locomotion.** Chapter 4 introduced a new method for splicing the upper body action of one motion with the lower

body locomotion of another. Unlike prior techniques for performing upper-body, lower-body splicing, my method explicitly considers the natural correlations between the upper and lower body. My approach of using temporal and spatial alignment to retain correlations produces high-quality results that exhibit physical as well as stylistic characteristics seen in the original two example motions. And because the method decouples motion parameters, it considerably reduces the storage requirements for interactive applications that need many different action/locomotion combinations at runtime. Since upper-body/lower-body splicing is already commonplace in the video game industry using naïve methods, my technique has potential to influence real world motion synthesis in interactive applications.

2. **Runtime Method for Adjusting the Gaze Direction of a Character.** In Chapter 5, I presented another decoupling method. This method decouples gaze direction from overall body motion. By basing my method on studies in the biological and psychological sciences, I have been able to produce a simple, efficient model for human gaze that uses little storage space and can be easily applied to any base motion at runtime. The ability to control gaze at runtime in a realistic way not only has the potential to increase the feeling of connectedness between a character and its environment, but it also has the potential for increasing character responsiveness through use of subtle gaze cues.

3. **Method for Automated Authoring of Parametric Motion Graphs.** My method for constructing a parametric motion graph as presented in Chapter 6 is highly-automated. The technique tackles the difficult problem of defining good transitions between entire spaces of motions using sampling. Without using an automated graph construction method, an author would be hard pressed to manually define these transitions without carefully hand-tweaking the motion data itself.

4. **Runtime Method for Using Parametric Motion Graphs.** Chapter 6 also shows how the structure of parametric motion graphs makes them a natural mechanism for controlling interactive characters in realtime. The synthesized motion streams exhibit quality transitions without pauses due to processing. In fact, processing times are extraordinarily quick, even

using my unoptimized, prototype system. This ability to synthesize controllable motion streams in realtime for interactive characters makes it possible for even low-budget applications to explore higher-quality methods for motion synthesis.

With these specific technical contributions, I have shown that example-based motion synthesis methods can be used to effectively synthesize quality human motion that accurately meets user requests in an interactive application.

While the methods presented in this dissertation either tackle the problem of motion clip synthesis through decoupling or focus on the problem of motion stream synthesis using highly structured control mechanisms, these methods could be combined to gain the advantages of both types of techniques. One possibility with great potential is to use a combination of motion clip synthesis methods within each node of a parametric motion graph (Chapter 6) in order to decouple motion parameters during motion clip generation. For instance, one node in a graph might represent all motions where a person walks at a particular curvature and looks in a particular direction. Internally, this node could use blending-based parametric synthesis to synthesize a base walking motion and then use the technique presented in Chapter 5 to layer the gaze direction of the character onto the base motion. The algorithms presented in this dissertation would not change even though the clips produced by each node of the parametric motion graph might be synthesized using a layering of decoupled methods; any smooth parametric motion space can be sampled no matter how the synthesized motions are produced.

To develop the algorithms presented in this dissertation, I built a custom testbed where motion clips and motion steams could be synthesized both online and offline. My system allows the upper-body action of one motion to be spliced onto the lower-body locomotion of another, allows a parametric gaze map to be applied to any base motion, and supports the interactive control of virtual human characters using parametric motion graphs. Because of the dependence on example motions, the effective use of these techniques in real world situations would not only require more optimized implementations of these algorithms but also tools for analyzing and browsing a motion database. Additional artist-oriented tools would also need to be developed to allow the authoring and tweaking of parametric gaze maps (Chapter 5) and parametric motion graphs (Chapter 6).

The rest of this chapter describes the applications for the methods I have introduced in this dissertation, the limitations of the overall approach of my work, and some potential directions for future research related to human motion synthesis for interactive applications.

7.1 Applications

The main focus of my work is on synthesizing human motion for interactive applications. The industry that often drives technology development in this realm is the video game industry. And the needs of the video game industry have strongly inspired the algorithms presented in this dissertation. My methods and insights have the potential to improve the quality and accuracy of human motions in a video game. Video games need the ability to decouple upper-body action from lower-body locomotion and currently fulfill this need using a simple naïve DOF replacement algorithm. My technique could be used as a direct replacement for these methods in order to improve splices in next-generation video games. And now gaze control is becoming a part of video games, again using simple, yet efficient methods that sacrifice motion quality. However, my gaze control model could be applied at runtime in order to perform more realistic looking gaze adjustments. Finally, the idea of combining blending-based parametric synthesis and synthesis-by-concatenation using sampling can increase the accuracy and quality of the motion streams synthesized in modern video games and greatly reduce the amount of time it takes to author controllable human characters.

My work also applies well to the area of online or offline crowd simulation. Crowd simulation is also restricted to using efficient methods for motion synthesis because of the sheer number of motions that must be synthesized. My decoupling methods can be used to efficiently synthesize a large variation of different motions using the same small set of example motions. And parametric motion graphs can allow both high-quality random motion synthesis (see Section 6.2.2.1) as well as motion synthesis that is guided by high-level goals (see Section 6.2.2.2), like those that an agent in a crowd might be required to meet. Using my techniques, a crowd might exhibit greatly variability as well as more accurate interactions with the environment.

Finally, my work applies well to synthesizing motions for training simulations that can aid in preparing workers for their jobs or teaching students about new subjects. The ability to control

gaze in an interactive application could lend a greater sense of believability to the actors in a simulation, making it easier to connect the lessons learned with the real world. Furthermore, as with video games, the ability to divide upper-body action from locomotion can increase flexibility in the motions that can be mixed-and-matched at runtime. Finally, because of the ease with which controllable characters can be produced using parametric motion graphs, even low-budget training simulations can begin to experiment with how motion quality affects the transference of the lessons being taught within the simulation. The impact of my work on training simulations could be significant in increasing the realism of these simulations while reducing the time and effort needed to author believable human characters.

7.2 Limitations

While the work presented in this dissertation meets my goal of increasing the utility of example-based motion generation methods for interactive applications by supplying insight into techniques that provide efficiency, low latency, high accuracy, quality, and automated authoring, there are still limitations. This section describes some of the most important limitations to consider when applying my approach to real applications.

7.2.1 Data

The effectiveness of any example-based human motion synthesis technique is necessarily tied to the quality and consistency of the motion database. It is important that capture motions are of high-quality as example-based techniques make this assumption. Furthermore, example motions must be captured in such a way as not to exhibit motion variations that are not intended. For instance, it is easy to capture 30 example motions of a person walking at various curvatures; it is much harder to ensure that the actor does not begin to exhibit subtle motion characteristics associated with fatigue towards the end of the motion capture shoot. It is usually not desirable to produce an interactive character that appears “tired” whenever he curves sharply to the right, for example.

All of the methods presented in this dissertation assume that the motions being blended or spliced are compatible with one another. The motions cannot come from two different actors with drastically different proportions or be mapped to different skeletal hierarchies. Some of the problems associated with data set compatibility can be overcome using motion retargeting methods, like those in [Gle98, SLGS01], but these types of techniques cannot address more stylistic compatibility issues; for instance, it might be possible to splice the upper body of a drunk person walking with loose joint control and the lower body of another person walking with a rigid, straight-legged waddle, but these motions are not necessarily compatible with one another.

And, as with many other example-based motion synthesis methods, the techniques presented in this dissertation depend on having enough example motions to adequately cover all possible motions that might be synthesized. My techniques are incapable of producing motion that is considerably different from the motions in the database. This includes transitions between motions. My reliance on linear blend transitioning rather than a more complex method means that I limit the motions that can be appended together to those that already appear similar. However, the problem of generating a quality transition between any two motion clips is an open problem that may be intractable.

7.2.2 Generalization

With the methods presented in this dissertation, I have shown the utility of decoupling and structure by developing methods that directly address some specific problems in motion synthesis for interactive character control. While these methods illustrate the advantages of my general approach, the methods themselves do not directly generalize. In particular, the methods for decoupling motion parameters during motion synthesis do not provide a method for decoupling any general parameter from any other.

Furthermore, all of the methods presented are primarily designed for the control of full body motion. Techniques for better control of facial features or hands, either independent of or in conjunction with full body motion, are not addressed.

7.2.3 User Input

While a goal of my work is to automate the process of authoring controllable human characters for interactive applications, my methods do depend on goals or hints supplied by a user. For instance, while my technique for building a parametric motion graph automates the process of constructing transition mappings from one parametric motion space to another, it still depends on user input to design the global shape of the graph (i.e., which nodes connect to which other nodes). The process of building an edge between parametric motion spaces also requires that a user specify a small number of tunable parameters that effectively define the tradeoff between motion quality and accuracy. Similarly, the automated methods for processing example motions of a person adjusting their gaze require that a user specify values for several tunable parameters; these parameters allow the algorithms to be adjusted for different motion capture environments by depending on the knowledge of the user.

Additionally, all of the methods presented depend on a user to supply the high-level information about what motion to synthesize. In truth, this ability to supply a high-level goal is the point of interactive motion synthesis, but in some cases, such as for gaze control, there may be automated methods that could help identify a plausible goal given the configuration of the environment and the current motion of the character.

7.3 Future Work

Considering the current trend towards using video games and training simulations to supplement classroom learning and on-the-job training, the need for quality, interactive human motion synthesis will only grow. And yet, there are still many open problems in the area of human motion synthesis:

Better Understanding of Human Perception: One area of human motion research that is only just starting to be explored is the area of human perception. When synthesizing human motion, techniques often introduce artifacts, such as foot-sliding, knee-popping, and bone-stretching. The goal of many researchers is to limit the introduced artifacts to ones that are

not offensive. Yet few papers like [HRvdP04], which studies the perceptual effects of bone-stretching, have appeared. More work is needed to determine which artifacts humans can perceive and to what extent. Using careful experimental design, it may be possible to study the perception of human motion to help inform motion synthesis techniques.

Better Motion Quality Metrics: One important characteristic of a virtual character motion is its quality. In this dissertation, I have sought to provide only high quality motion synthesis methods. However, as stated throughout this document, quality is a partially subjective characteristic and there is no standard way to measure motion quality directly. My work focuses on ensuring only some quality characteristics, such as artifact appearance and continuity. Yet, in the future, better metrics and processes for measuring the quality of a human motion could greatly impact not only my own work but also the work of others in the motion synthesis community.

Better Physical Models: One downside to example-based human motion synthesis is that the motions generated using these techniques might break some of the laws of physics. Some researchers have begun studying physically based models for motion generation, but these models fail to recognize the importance of characteristics, such as personality, that a purely physical approach cannot capture. Future work on ways to combine example-based approaches with physically based approaches in order to create realistic motions with desirable physical as well as stylistic properties could have great potential to the interactive application community.

Motion Browsing: The proliferation of example-based motion synthesis techniques means that the number of motions in a typical motion database is bound to grow. Large numbers of motions not only present a problem for motion synthesis techniques, but large number of motions also make it difficult to browse a motion database. The community is in need of better ways to present the data in a motion collection so that a human can browse it rapidly. Solutions to the problem of motion browsing might include better methods for automatically

identifying motion characteristics as well as better methods for “summarizing” a motion in a single thumbnail image.

I hope that my work on efficiently generating accurate, quality motion using an example-based approach will increase the quality and accuracy of human motion in real world applications and inspire others to develop practical yet quality-oriented algorithms for motion synthesis in interactive applications.

REFERENCES

[ACP02] B. Allen, B. Curless, and Z. Popovic. Articulated body deformation from range scan data. *ACM Transactions on Graphics*, 2002.

[AF02] O. Arikan and D. A. Forsythe. Interactive motion generation from examples. In *Proceedings of ACM SIGGRAPH*, pages 483–490, 2002.

[AFO03] O. Arikan, D. A. Forsyth, and J. O'Brien. Motion synthesis from annotations. In *Proceedings of ACM SIGGRAPH*, pages 402–408, 2003.

[AGH03] N. Al-Ghreimil and J. Hahn. Combined partial motion clips. In *The 11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*. Eurographics, February 2003.

[Ame04] *The American Heritage Dictionary of the English Language*. Houghton Mifflin Company, fourth edition, 2004.

[AW00] G. Ashraf and K. C. Wong. Generating consistent motion transition via decoupled framespace interpolation. *Computer Graphics Forum*, 2000.

[BB98] R. Bindiganavale and N. I. Badler. Motion abstraction and mapping with spatial constraints. In *Proceedings of CAPTECH*, 1998.

[BBK01] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, pages 322–373, 2001.

[BBM+01] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen. Unstructured lumigraph rendering. In *Proceddings of ACM SIGGRAPH*, pages 425–432, 2001.

[BC89] A. Bruderlin and T. Calvert. Goal-directed dynamic animation of human walking. In *Proceedings of ACM SIGGRAPH*, pages 233–242, 1989.

[BC96] A. Bruderlin and T. Calvert. Knowledge-driven, interactive animation of human running. In *Proceedings of Graphics Interface*, pages 213–221, 1996.

[Bel57] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[BH00] M. Brand and A. Hertzmann. Style machines. In *Proceedings of ACM SIGGRAPH*, 2000.

[Bod00] P. Bodik. Automatic footplant detection: inside flmoview. As Summer Project Web Page: http://www.cs.wisc.edu/graphics/Gallery/PeterBodik/, 2000.

[Bow00] R. Bowden. Learning statistical models of human motion. In *Proceedings of IEEE CVPR*, 2000.

[BRRP97] B. Bodenheimer, C. Rose, S. Rosenthal, and J. Pella. The process of motion capture: Dealing with the data. *Computer Animation and Simulation*, 1997.

[BW95] A. Bruderlin and L. Williams. Motion signal processing. In *Proceedings of ACM SIGGRAPH*, pages 97–104, August 1995.

[CHP07] S. Cooper, A. Hertzmann, and Z. Popovic. Active learning for real-time motion controllers. In *Proceedings of ACM SIGGRAPH*, 2007.

[CLS03] M. G. Choi, J. Lee, and S. Y. Shin. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics*, 22(2):182–203, 2003.

[DYP03] M. Dontcheva, G. Yngve, and Z. Popovic. Layered acting for character animation. In *Proceedings of ACM SIGGRAPH*, July 2003.

[FS00] E. G. Freedman and D. L. Sparks. Coordination of the eyses and head: movement kinematics. *Experimental Brain Research*, 2000.

[FvdPT01a] P. Faloutsos, M. van de Panne, and D. Terzopoulos. Composable controllers for physics-based character animation. In *Proceedings of ACM SIGGRAPH*, page 251260, 2001.

[FvdPT01b] P. Faloutsos, M. van de Panne, and D. Terzopoulos. The virtual stuntman: Dynamic characters with a repetoire of autonomous motor skills. *Computers and Graphics*, pages 933–953, 2001.

[GJH01] A. Galata, N. Johnson, and D. Hogg. Learning variable length markov models of behaviour. *Computer Vision and Image Understanding*, 2001.

[Gle98] M. Gleicher. Retargetting motion to new characters. In *Proceedings of ACM SIGGRAPH*, 1998.

[Gle01] M. Gleicher. Motion path editing. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics*, 2001.

[GMHP04] K. Grochow, S. Martin, A. Hertzmann, and Z. Popovic. Style-based inverse kinematics. In *Proceedings of ACM SIGGRAPH*, 2004.

[Gra98] F. S. Grassia. Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools*, 3(3):29–48, 1998.

[GSKJ03] M. Gleicher, H. J. Shin, L. Kovar, and A. Jepsen. Snap-together motion: Assembling run-time animation. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics*, April 2003.

[GV87] D. Guitton and M. Volle. Gaze control in humans: eye-based coordination during orienting movements to targets within and beyond the oculomotor range. *Journal of Neurophysiology*, 1987.

[HGP04] E. Hsu, S. Gentry, and J. Popovic. Example-based control of human motion. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004.

[Hor87] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4:629–642, 1987.

[HP97] J. Hodgins and N. Pollard. Adapting simulated behaviors for new characters. In *Proceedings of ACM SIGGRAPH*, page 153162, 1997.

[HPP05] E. Hsu, K. Pulli, and J. Popovic. Style translation for human motion. In *Proceedings of ACM SIGGRAPH*, pages 1082–1089, July 2005.

[HRvdP04] J. Harrison, R. Rensink, and M. van de Panne. Obscuring length changes during animated motion. In *In Proceedings of ACM SIGGRAPH*, 2004.

[HW98] J. K. Hodgins and W. L. Wooten. Animating human athletes. *Proceedings of Robotics Research: The Eighth International Symposium*, pages 356–367, 1998.

[HWBO95] J. Hodgins, W.Wooten, D. Brogan, and J. O'Brien. Animating human athletics. In *Proceedings of ACM SIGGRAPH*, pages 71–78, August 1995.

[HWG07] R. Heck, M. Wallick, and M. Gleicher. Virtual videography. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCAPP)*, 2007.

[IAF06] L. Ikemoto, O. Arikan, and D. Forsyth. Knowing when to put your foot down. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics*, 2006.

[IAF07] L. Ikemoto, O. Arikan, and D. Forsyth. Quick transitions with cached multi-way blends. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2007.

[IF04] L. Ikemoto and D. A. Forsyth. Enriching a motion collection by transplanting limbs. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, August 2004.

[JT81] O. Johnston and F. Thomas. *Disney Animation: The Illusion of Life*. Abbeville Press, 1981.

[KB96] H. Ko and N. Badler. Animating human locomotion with inverse dynamics. *IEEE Computer Graphics and Application*, 16(2):58–59, 1996.

[KG03] L. Kovar and M. Gleicher. Flexible automatic motion blending with registration curves. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, July 2003.

[KG04] L. Kovar and M. Gleicher. Automated extraction and parameterization of motions in large data sets. In *Proceedings of ACM SIGGRAPH*, pages 559–568, 2004.

[KGM07] K. Han Kim, R. Brent Gillespie, and Bernard J. Martin. Head movement control in visually guided tasks: Postural goal and optimality. *Computers in Biology and Medicine*, pages 1009–1019, 2007.

[KGP02] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *Proceedings of ACM SIGGRAPH*, 2002.

[Kov04] L. Kovar. Automated methods for data-driven synthesis of realistic and controllable human motion. University of Wisconsin-Madison, Ph.D. Thesis, 2004.

[KP06] P. G. Kry and D. K. Pai. Interaction capture and synthesis. In *Proceedings of ACM SIGGRAPH*, 2006.

[KPS03] T. Kim, S. Park, and S. Shin. Rhythmic-motion synthesis based on motion-beat analysis. In *Proceedings of ACM SIGGRAPH*, pages 392–401, 2003.

[KS05] T. Kwon and S. Y. Shin. Motion modeling for on-line locomotion synthesis. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, July 2005.

[KSG02] L. Kovar, J. Schreiner, and M. Gleicher. Footskate cleanup for motion capture editing. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2002.

[LBB02] S. P. Lee, J. B. Badler, and N. I. Badler. Eyes alive. In *Proceedings of ACM SIGGRAPH*, 2002.

[LCL06] K. H. Lee, M. G. Choi, and J. Lee. Motion patches: Building blocks for virtual environments annotated with motion data. In *Proceedings of ACM SIGGRAPH*, 2006.

[LCR+02] J. Lee, J. Chai, P. Reitsma, J. Hodgins, and N. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings of ACM SIGGRAPH*, pages 491–500, 2002.

[LK06] M. Lau and J. Kuffner. Precomputed search trees: Planning for interactive goal-driven animation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2006.

[LL04] J. Lee and K. H. Lee. Precomputing avatar behavior from human motion data. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pages 79–87, 2004.

[LP02] C. K. Liu and Z. Popovic. Synthesis of complex dynamic character motion from simple animations. *ACM Transactions on Graphics*, page 408416, 2002.

[LS99] J. Lee and S. Y. Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of ACM SIGGRAPH*, 1999.

[LvdP96] A. Lamouret and M. van de Panne. Motion synthesis by example. In *Proceedings of the Eurographics workshop on Computer animation and simulation*, pages 199–212, 1996.

[LvdPF96] J. Laszlo, M. van de Panne, and E. Fiume. Limit cycle control and its application to the animation of balancing and walking. In *Proceedings of ACM SIGGRAPH*, pages 155–162, 1996.

[LWS02] Y. Li, T. Wang, and H. Y. Shum. Motion texture: A two-level statistical model for character motion synthesis. In *Proceedings of ACM SIGGRAPH*, 2002.

[LZWP03] F. Liu, Y. Zhuang, F. Wu, and Y. Pan. 3d motion retrieval with motion index tree. *Computer Vision and Image Understanding*, pages 265–284, 2003.

[Mac90] A. A. Maciejewski. Dealing with the ill-conditioned equations of motion for articulated figures. *IEEE Computer Graphics and Applications*, 1990.

[MBBT00] J. S. Monzani, P. Baerlocher, R. Boulic, and D. Thalmann. Using an intermediate skeleton and inverse kinematics for motion retargeting. In *Proceedings of Eurographics*, 2000.

[MBC01] M. Mizuguchi, J. Buchanan, and T. Calvert. Data driven motion transitions. In *Eurographics Short Presentations*, September 2001.

[MC07] M. K. McCluskey1 and K. E. Cullen. Eye, head, and body coordination during large gaze shifts in rhesus monkeys: Movement kinematics and the influence of posture. *Journal of Neurophysiology*, 2007.

[Men00] A. Menache. *Understanding Motion Capture for Computer Animation and Video Games*. Academic Press, 2000.

[MFCD99] F. Multon, L. France, M.-P. Cani, and G. Debunne. Computer animation of human walking: a survey. *The Journal of Visualization and Computer Animation*, 10:39–54, 1999.

[MK05] T. Mukai and S. Kuriyama. Geostatistical motion interpolation. In *Proceedings of ACM SIGGRAPH*, 2005.

[MP07] J. McCann and N. S. Pollard. Responsive characters from motion fragments. In *Proceedings of ACM SIGGRAPH*, 2007.

[MZF06] A. Majkowska, V. Zordan, and P. Faloutsos. Automatic splicing for hand and body animations. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2006.

[MZH+07] R. A. Metoyer, V. B. Zordan, B. Hermens, C. C. Wu, and M. Soriano. Psychologically inspired anticipation and dynamic response for impacts to the head and upper body. *Transactions on Visualization and Computer Graphics*, 2007.

[NF02] M. Neff and E. Fiume. Modeling tension and relaxation for computer animation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 8188, 2002.

[OBBH00] J. F. O'Brien, B. Bodenheimer, G. Brostow, and J. K. Hodgins. Automatic joint parameter estimation from magnetic motion capture data. In *Proceedings of Graphics Interface*, 2000.

[PB00] K. Pullen and C. Bregler. Animating by multi-level sampling. In *Proceedings of IEEE Computer Animation Conference*, pages 36–42. CGS and IEEE, May 2000.

[PB02] K. Pullen and C. Bregler. Motion capture assisted animation: Texturing and synthesis. In *Proceedings of ACM SIGGRAPH*, July 2002.

[Per95] K. Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, March 1995.

[PG96] K. Perlin and A. Goldberg. Improv: A system for scripting interactive actors in virtual worlds. In *Proceedings of ACM SIGGRAPH*, August 1996.

[PL06] J. Pettre and J. P. Laumond. A motion capture-based control-space approach for walking mannequins: Research articles. *Computer Animation and Virtual Worlds*, pages 109–126, 2006.

[PSS02] S. I. Park, H. J. Shin, and S. Y. Shin. On-line locomotion generation based on motion blending. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, July 2002.

[PZ05] N. Pollard and V. Zordan. Physically based grasping control from example. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005.

[RCB98] C. Rose, M. Cohen, and B. Bodenheimer. Verbs and adverbs: multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5), 1998.

[RGBC96] C. Rose, B. Guenter, B. Bodenheimer, and M. Cohen. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of ACM SIGGRAPH*, pages 147–154, August 1996.

[RP04] P. S. A. Reitsma and N. S. Pollard. Evaluating motion graphs for character navigation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pages 89–98, 2004.

[RP07] P. Reitsma and N. Pollard. Evaluating motion graphs for character animation. *ACM Transactions on Graphics*, 2007.

[RSC01] C. Rose, P. Sloan, and M. Cohen. Artist-directed inverse-kinematics using radial basis function interpolation. *Computer Graphics Forum*, 20(3), 2001.

[SDO+04] M. Stone, D. DeCarlo, I. Oh, C. Rodriguez, A. Stere, A. W. Lees, and C. Bregler. Speaking with hands: Creating animated conversational characters from recordings of human performance. In *Proceedings of ACM SIGGRAPH*, 2004.

[SH05] A. Safonova and J. Hodgins. Analyzing the physical correctness of interpolated human motion. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005.

[SH07] A. Safonova and J. K. Hodgins. Construction and optimal search of interpolated motion graphs. In *Proceedings of ACM SIGGRAPH*, 2007.

[Sho85] K. Shoemake. Animating rotation with quaternion curves. In *Proceedings of ACM SIGGRAPH*, pages 245–254, July 1985.

[SKF07] A. Shapiro, M. Kallmann, and P. Faloutsos. Interactive motion correction and object manipulation. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2007.

[SKG03] H. SHIN, L. KOVAR, and M. GLEICHER. Physical touch-up of human motions. In *Proceedings of Pacific Graphics*, October 2003.

[SKG05] M. Sung, L. Kovar, and M. Gleicher. Fast and accurate goal-directed motion synthesis for crowds. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005.

[SLGS01] H.-J. Shin, J. Lee, M. Gleicher, and S.-Y. Shin. Computer puppetry: an importance-based approach. *ACM Transactions on Graphics*, 20(2):67–94, April 2001.

[SM01] H. Sun and D. Metaxas. Automating gait animation. In *Proceedings of ACM SIGGRAPH*, 2001.

[SMM05] M. Srinivasan, R. A. Metoyer, and E. N. Mortensen. Controllable character animation using mobility maps. *Graphics Interface*, 2005.

[SNI06] T. Shiratori, A. Nakazawa, and K. Ikeuchi. Dancing-to-music character animation. In *Proceedings of Eurographics*, 2006.

[SO06] H. J. Shin and H. S. Oh. Fat graphs: Constructing an interactive character with continuous controls. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2006.

[SSSE00] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa. Video textures. In *Proceedings of ACM SIGGRAPH*, pages 489–498, 2000.

[TH00] L. M. Tanco and A. Hilton. Realistic synthesis of novel human movements from a database of motion capture examples. In *Proceedings of IEEE Workshop on Human Motion*, 2000.

[TLP07] A. Treuille, Y. Lee, and Z. Popovic. Near-optimal character animation with continuous control. In *Proceedings of ACM SIGGRAPH*, 2007.

[TM04] S. C. L. Terra and R. A. Metoyer. Performance timing for keyframe animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 253–258, 2004.

[TSK02] S. TAK, O. SONG, and H. KO. Spacetime sweeping: An interactive dynamic constraints solver. *Computer Animation*, pages 261–270, June 2002.

[WB03] J. Wang and B. Bodenheimer. An evaluation of a cost metric for selecting transitions between motion segments. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003.

[WDAC06] J. Wang, S. M. Drucker, M. Agrawala, and M. F. Cohen. The cartoon animation filter. In *Proceedings of ACM SIGGRAPH*, pages 1169–1173, 2006.

[WH97] D. Wiley and J. Hahn. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications*, 17(6), 1997.

[WH00] W. Wooten and J. Hodgins. Simulation of leaping, tumbling, landing, and balancing humans. In *Proceedings of IEEE International Conference on Robotics and Animation*, 2000.

[WP95] A. Witkin and Z. Popovic. Motion warping. In *Proceedings of ACM SIGGRAPH*, 1995.

[YKH04] K. Yamane, J. Kuffner, and J. Hodgins. Synthesizing animations of human manipulation tasks. In *Proceedings of ACM SIGGRAPH*, 2004.

[YLvdP07] K. Yin, K. Loken, and M. van de Panne. Simbicon: Simple biped locomotion control. In *Proceedings of ACM SIGGRAPH*, 2007.

[ZH03] V. B. Zordan and N. C. Van Der Horst. Mapping optical motion capture data to skeletal motion using a physical model. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 245–250, 2003.