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.