Chapter 2

Related Work

This chapter reviews research related to human motion synthesis. We first present a general overview of common strategies, and then for each of the specific topics covered in this dissertation we provide a detailed review of previous work.

2.1 Motion Synthesis Methods

Existing motion synthesis methods can be divided into three categories: manual synthesis, physically-based synthesis, and data-driven synthesis. In this section we discuss specific methods that fall within these categories and summarize some strengths and weaknesses of each approach.

2.1.1 Manual Synthesis

The oldest and most basic synthesis method is to manually specify a character's degrees of freedom (DOFs) at individual points in time, which are called keyframes. DOFs in between keyframes are then computed through simple interpolation methods, such as cubic spline interpolation. Since this interpolation rarely produces realistic results unless the keyframes are quite dense, the artist must manually construct a substantial fraction of the character poses that appear in the motion. This is arduous and time consuming, not only because many poses must be created (animations typically run at 24 frames per second), but also because it is challenging to craft these poses such that they yield a convincing motion when played in sequence. On the other hand, keyframing

provides complete control over every detail of a character's movement, and it remains popular in applications (such as cartoons) where motion need not be realistic as long as it is expressive.

An alternative to keyframing is to handcraft algorithms that procedurally replicate particular motions. This allows an artist to create entire motions at once, rather than one pose at a time, and motions can be altered on the fly to allow for online, interactive character animation. Perlin [69] and Perlin and Goldberg [70] have demonstrated that a variety of motions can be generated with simple and computationally efficient algorithms. However, these algorithms can be quite difficult to design, a new algorithm must be developed whenever a new kind of motion is to be animated, and the generated motion rarely attains the same level of realism as motion capture.

2.1.2 Physically-Based Synthesis

Since real human movement is governed by the laws of physics, physical simulation is a natural strategy for animating motion. Given the mass distribution for each body part and the torques generated by each joint, Newton's laws provide a system of ordinary differential equations that can be integrated to yield joint trajectories. However, while average mass distributions can be obtained from the biomechanics literature [97], finding joint torques that yield a particular motion is considerably more difficult. Hodgins et al. [39] proposed handcrafting joint controllers based on finite state machines and proportional-derivative servos. They demonstrated this approach on several motions, including running, bicycling, and vaulting. Similar methods were employed by Wooten and Hodgins [99, 100] to generate gymnastic motions like flipping or tumbling and by Faloutsos et al. [24] to generate motions for preserving balance and recovering from a fall. While these efforts have successfully produced certain kinds of motion, controller design is a difficult task, and a given controller can only yield a narrow range of motion. On the other hand, once a controller has successfully been created, one can attempt to adapt it to new circumstances. For cyclic motions like walking, Laszlo et al. [51] applied limit cycle control to adjust the underlying joint controller so as to yield a desired variation of the original motion. To adapt a controller to a new body, Hodgins and Pollard [38] computed new controller parameters by first applying approximate scaling strategies and then tuning the results with simulated annealing. Finally, Faloutsos et

al. [23] used support vector machines to learn the conditions under which controllers for different actions can be composed, providing a meta-controller that switches control strategies to transition between actions.

Rather than explicitly constructing joint controllers, some previous work has employed constrained optimization to convert a small number of user-provided keyframes into a full motion that obeys physical laws. Liu and Popović [58] considered the case of ballistic motions, such as leaping or hopping. Given a small number of keyframed poses, a rough motion was first computed via spline interpolation. An optimization was then performed to find a minimal deviation from this initial motion that obeyed calculated trajectories for the linear and angular momentum of/about the center of mass. When the character was airborne, these trajectories were derived from Newton's laws, and when the character was on the ground, they were governed by a biomechanically-inspired model of momentum transfer. To make optimization more efficient, Fang and Pollard [25] demonstrated that physical constraints expressed in terms of aggregate force and torque (rather than the force and torque at each joint) can be differentiated in time linear in the number of character DOFs, whereas in general derivative computation is of quadratic complexity.

Although physically based motion synthesis can automatically generate physically realistic motions, a motion that obeys the laws of physics can appear devoid of personality. Neff and Fiume [65] proposed creating more natural looking motion through an antagonistic joint control model, where opposing muscle forces varied the amount of tension and relaxation. Finding appropriate tension parameters, however, can be a nontrivial task. To create more realistic motions through physically-based optimization, Safonova et al. [80] applied principal components analysis to captured examples of a motion to extract a set of basis poses, which were then used to restrict the space of possible skeletal postures. While this can improve convergence and make individual poses appear more natural, the *sequence* of poses generated by the optimization still does not approach the fidelity of captured motion. More generally, at present physical simulation can only provide a relatively narrow range of realistic motion, and for this reason we use motion capture as our source of example motions. However, our methods would work just as well when applied to physically simulated or hand-animated motion.

2.1.3 Data-Driven Synthesis

Motion capture technology provides a source of highly realistic example motions that can serve as raw material for a variety of data-driven synthesis algorithms, including those introduced in this dissertation. In this section we will discuss previous work in data-driven synthesis that is *not* directly related to motion graphs, motion blending, or parameterized motion. These topics will be deferred until Sections 2.3–2.5. Also, we note that while historically the availability of motion capture data has made data-driven synthesis practical, example motions could also potentially come from keyframe animation, physical simulation, or even another data-driven synthesis algorithm. In this respect the techniques mentioned here are complimentary to our own, as they could readily be applied to motions generated with our methods.

A simple way of creating motion is to apply signal processing operations independently to each degree of freedom of an example motion. Bruderlin and Williams [15] identified several potentially useful operations, including multiresolution filtering, waveshaping, and adding smooth displacement maps. Witkin and Popovic [98] introduced a variant of displacement mapping called motion warping, and Gleicher [32] adapted displacement mapping to provide a simple interface for interactively editing the path traversed by a character. These algorithms are fast and easy to implement, but because they operate on each DOF independently, they are not well suited for adjustments that involve coordinated movement. For example, moving the hand to a particular location can require simultaneous adjustments to the elbow, shoulder, and spine. Gleicher [30, 31], Lee and Shin [54], and Shin et al. [83] coherently adjusted multiple DOFs by using optimization to minimally adjust a motion such that user-specified constraints (e.g., hand position at certain times) were satisfied.

Several researchers have proposed editing operators designed to alter a motion's aesthetic properties. Unuma et al. [90] combined cyclic motions by linearly combining the Fourier coefficients of each DOF, and they reported that in some cases abstract properties such as a walk's "briskness" could be controlled. Given two examples of the same action, one performed neutrally and one with a particular emotion (e.g., angrily), Amaya et al. [4] represented the difference as a warp applied

to the timing and amplitudes of the neutral motion. These warps could then be applied to a different neutral motion, with the goal of adding similar emotional content. Chi et al. [19] applied Laban Movement Analysis to control the expressive content of gestures. Neff and Fiume [66] presented operations for adjusting a motion's succession (how movement spreads from the hip to the extremities), amplitude (total joint range of motion), and extent (proximity of hands and arms to body).

The preceding algorithms all adjust only kinematic properties, but other algorithms are designed to preserve physical correctness. Tak et al. [47, 86] used optimization to ensure that the body's zero moment point remained inside the support polygon, which is a requirement of physical validity. Popović and Witkin [71] presented an editing framework based on physically-based optimization that improved efficiency by first fitting the original motion onto a hand-tailored simplified model that still captured essential dynamical features. Abe et al. [1] employed an optimization framework similar to [58] to adjust captured ballistic motions (such as jumping up and spiking a volleyball) while obeying restrictions on angular and linear momentum. Zordan and Hodgins [102] mapped motion capture data onto a physical simulation by using proportional-derivative servos to produce joint torques that tracked the measured joint angles. A motion could then be edited either by applying new external forces (e.g., to simulate being hit) or by kinematically adjusting the original motion and then tracking the result as closely as possible in the simulation.

2.1.4 Comparison of Motion Synthesis Methods

Manual, physically-based, and data-driven synthesis each have advantages and disadvantages. Manual synthesis provides the greatest degree of control over motion, as it is constrained neither by visual realism nor by physical accuracy. At the same time, manual synthesis requires an extraordinary investment of time and talent, and realistic human motion in particular can only be created with considerable effort from the most skilled of artists. Physically-based synthesis eliminates a great deal of manual labor by, roughly speaking, using physical laws to automatically fill in the details of a character's motion. However, while this approach guarantees that motions are physically accurate (within the limitations of the rather crude approximation that the human body

is a rigid skeleton), physical accuracy does not imply visual realism. For highly dynamic motions, such as a standing broad jump, the laws physics sufficiently constrain the range of possible motion that physically-based synthesis yields convincing results. For more nuanced motions, such as walking, picking up a glass, or smelling a flower, physics provides only a loose set of constraints, and physically-based synthesis yields motions that look awkward and unnatural. Moreover, physically-based synthesis is computationally expensive: current algorithms [25, 58, 80] on modern hardware typically require several minutes to produce a few seconds of motion. Data-driven synthesis, in contrast, can yield highly realistic animations of effectively any motion that a real person can perform, and most algorithms (including those is this dissertation) can produce motion in real time. However, data-driven synthesis requires access to relatively expensive special-purpose motion capture hardware. Also, while in principle data driven synthesis could be applied to any creature whose movement can be captured, in practice it is primarily applicable to humans. Manual and physically-based synthesis, in contrast, can be readily applied to a wide range of body structures.

2.2 Enforcing Kinematic End Effector Constraints

Enforcing kinematic constraints on end effectors is a special case of the *inverse kinematics* (IK) problem, which is to adjust an articulated figure so one or more of its joints are at specified positions and/or orientations. The use of IK in animation dates back to some of the earliest systems [48, 29]. Fundamentally, IK is about finding the solution to a system of nonlinear equations, and IK algorithms may be divided into two classes: numerical and analytical. Numerical algorithms typically arrive at a solution by linearizing the problem about an initial skeletal configuration, moving towards the goal configuration, and repeating until convergence; see Welman [95] for a survey of computational methods. These algorithms can handle sophisticated constraints on complex skeletons, but are considerably more computationally expensive than analytical algorithms. Also, since the equations are ill-conditioned in certain character configurations [60], it is difficult to guarantee that numerical methods will yield smooth changes to a character's DOFs, even when smooth

changes are made to end effector trajectories. Analytical algorithms, in contrast, find efficient closed-form solutions to the IK equations and typically provide smoothness guarantees. However, there is no general method for constructing an analytical IK solver for a given skeletal topology, and in general these solvers only exist for relatively simple topologies with far fewer DOFs than a full-body skeleton.

This dissertation introduces a new analytical IK algorithm that can be used to place wrists and ankles in specified positions and orientations. Our algorithm incorporates a specialized IK solver designed for individual limbs. This single-limb solver is similar to one developed by Tolani et al. [89], except we take measures to avoid "popping" artifacts that can occur when a limb is near full extension. Previous full-body IK algorithms have also treated individual limbs independently [54, 83].

In general, a motion will contain multiple distinct constraints that each last for multiple frames and may or may not overlap in time. Ideally, these constraints will be satisfied by adding a smoothly varying sequence of per-frame adjustments. Performing IK independently on each frame [76] will not in general yield a smooth result, since constraints can discontinuously switch on and off. Monzani et al. [64] smoothly dissipated adjustments into unconstrained frames, but they employed a general numerical IK solver and did not ensure that smooth changes would be made when the number of constraints affecting a character changed (e.g., on one frame both ankles might be constrained, and on the next frame just one ankle might be constrained). We ensure that only smooth changes are made regardless of constraint timing. Gleicher [30, 31] enforced smoothness by representing the total adjustment made to each DOF with a cubic spline and optimizing the knot values. Since the optimization was performed over the entire motion, this process was offline and could scale poorly to long motions, and the restriction that adjustments be represented as a spline means that in general constraints could only be approximately enforced. Our method uses a fixed amount of computation per frame, requires a limited amount of lookahead, and satisfies constraints exactly. Lee and Shin [54] satisfied kinematic constraints with a hierarchical sequence of adjustments. At each stage a hybrid numerical/analytical IK solver was applied independently to each frame and a spline was fit to the resulting displacements, with the knot spacing growing

smaller at later stages of the algorithm. This required IK to be performed on each frame multiple times, whereas our algorithm executes a single per-frame IK calculation and uses a purely analytic IK algorithm.

Our work is related to the task of fitting a skeleton to measured marker positions, such as would be derived from an optical motion capture system. Each marker is assumed to be rigidly attached to a single joint, and sufficiently many markers exist so all of a joint's DOFs can be reconstructed from the markers attached to it. However, some error will be introduced since the human body is not truly a rigid skeleton, and these errors will accumulate if the orientation of each joint in a kinematic chain is independently reconstructed in sequence. This can cause reconstructed end effector positions to deviate substantially from their measured locations. Several methods exist for solving this problem. Bodenheimer et al. [11] used an offline optimization-based numerical IK algorithm (related to that of Zhao and Badler [101]) wherein the user could control the tradeoff between accurate joint orientations and accurate joint positions. Choi and Ko [20] presented a similar online algorithm based on inverse rate control. Shin et al. [83] used a simplified IK solver to perform online skeletal reconstruction where the importance of matching measured end effector positions was tied to their proximity to important objects in the environment (e.g., a doorknob). All of these algorithms rely on having measured end-effector positions, which are not available when motions are generated through data-driven synthesis methods.

2.3 Motion Graphs

The video game industry has long built motion graphs (sometimes called "move trees") for the purpose of character control, although it is only relatively recently that this practice has been published [62]. These motion graphs have historically been constructed manually in the sense that a user explicitly decided which motion clips could be connected. Early research involving motion graphs also built the graphs manually [69, 76]. We show how a motion graph can be automatically constructed by identifying places where motions are locally similar.

Automated graph-based motion synthesis has been the subject of a considerable amount of previous work. Several researchers have built graph-based statistical models wherein each node is a simple generator of poses and edges represent transitions between different kinds of poses [13, 28, 14, 57]. Nodes were formed by clustering poses from the original data, and transition probabilities at edges were computed based on the sequence of poses in the input motions. While statistical models provide a natural means of adding variability to a motion, they also place looser guarantees on motion quality since the true statistics of human motion are unknown (e.g., pose distributions at a given instant are not as simple as a gaussian). Also, these efforts have not focused on controlling the properties of generated motion. Molina-Tanco and Hilton [63] constructed a hidden Markov model wherein the observables corresponded to clusters of motions segments (generated with a k-means algorithm), the hidden states were individual motion segments, and transition probabilities took on one of two values depending on whether the segments of motion were contiguous in the original data. A user could require a motion to start and end in two particular poses, and a dynamic programming algorithm found a maximum-likelihood sequence of connecting motion segments. This work is similar to our own in that generated motion consisted of spliced segments of captured motions, as opposed to the output of a probabilistic model. However, we use a similarity-based cost metric for transitions that can take on a continuum of values, rather than one of two values, and we consider more general constraints on motion synthesis. Pullen and Bregler [72] "stylized" simple keyframed animations by breaking them into smaller pieces and replacing each piece with captured motion. To control the process, a user selected skeletal parameters particularly important to the motion and specified a characteristic time scale. The system then found clips of motion data which matched keyframed parameters at the appropriate frequency band. Our method does not require a keyframed example motion, although one could be supplied as part of an objective function.

Graph-based synthesis has also been discussed in research involving low-DOF characters [49] and motion generated from procedural methods [69], physical simulation [23, 41], and video [82, 81]. These different motion representations require different methods for identifying and creating transitions.

Our work was performed concurrently with (and independently of) that of Lee et al. [52] and Arikan and Forsyth [5], both of whom also automatically identified transition locations based upon a distance metric and then used search algorithms on the resulting graph to extract motions that satisfied user-defined constraints. Aside from technical distinctions in how distances were defined and how transitions were created, the primary differences with our work were in the applications. We consider constraints on the path followed by the character throughout the duration of the motion, with the style of the motion (represented with an annotation, such as "ballet") optionally restricted on different parts of the path. Arikan and Forsyth [5] allowed constraints on the pose, position, and orientation of the character on the first and last frames of animation, with no restrictions on intermediate frames. Lee et al. [52] provided path constraints without style restrictions, generated motion that best fit a video sequence, and presented an interface where a user could directly select what motion was to take place next.

The generation of transitions is an important part of building motion graphs. Our work uses a simple linear blending method similar to that of Perlin [69], except that we propogate kinematic constraints from the original data into the transitions. Other transition methods are possible, such as displacement maps [5, 52, 72], the torque-minimization algorithm of Rose et al. [77], or the blending method introduced in Chapter 5.

There has been much recent interest in graph-based motion synthesis, and a number of extensions have been made to our work and that of Lee et al. [52] and Arikan and Forsyth [5]. To help users build motion graphs with a simpler and more understandable structure, Gleicher et al. [33] developed a system for automatically detecting skeletal poses that appear frequently in a data set. Multi-way transitions were created around these poses, forming high-connectivity graphs with a small number of nodes. Arikan et al. [6] presented an approximate dynamic programming algorithm for extracting motions that satisfy annotation constraints. For the special case of motion with a clear rhythmic structure (such as dancing), Kim et al. [46] automatically segmented motion data using beat analysis, clustered these motion segments with a k-means algorithm, and then built a Markov model representing transition probabilities between different clusters. New motions could then be synthesized that preserved the original data's rhythmic structure while satisfying

constraints on the path traversed by the character. Reitsma and Pollard [75] empirically evaluated the ability of a particular motion graph to perform navigation tasks (i.e., directing a character to a particular position and orientation) by embedding the graph in the environment of interest and calculating 1) the degree to which different portions of the environment could be connected with a motion and 2) the ability of synthesized motions to follow the shortest path between two points. Given a data set containing two coupled motions and a new "control" motion, Hsu et al [40] used an approximate dynamic programming algorithm to construct the complementary motion. For example, given data of two people dancing and a new motion for the leader, an appropriate motion for the follower could be generated automatically. Finally, Lee and Lee [53] controlled motion synthesis in real time by using dynamic programming to precompute a lookup table indicating the optimal transition to take from any node given one of a finite set of goal configurations.

2.4 Motion Blending

Motion blending has been an integral part of several research efforts. One of the earliest is the real-time procedural animation system of Perlin [69]. To animate a character, a user manually constructed a set of base motions and then used blending operations to transition between motions and to create new ones via interpolation. The language for constructing motions contained built-in timing and constraint models, simplifying the related blending issues. Several systems have used multitarget interpolation to create parameterized motions [96, 78], and the Verbs and Adverbs system of Rose et al. [76] combined this with linear blend transitions to create a motion graph containing parameterized motions. Multiple research groups have used general blending (that is, blending with arbitrary weight functions) to provide continuous control of locomotion [35, 7, 68]. Registration curves support all of these operations (interpolation, transitioning, and continuous control) and could be used as a back end in these systems for computing blends.

2.4.1 Timing

Motions with different timing may be *timewarped* such that logically related events occur simultaneously; we refer to this mapping between corresponding frame indices as a *timewarp curve*. We build timewarp curves automatically and for arbitrarily many motions. Timewarping has also been an important part of previous work on motion blending. Some systems required the user to mark sparse "key times" in the input motions to acquire time correspondences [76, 68]; for example, for a set of punches these might be the initiation, apex, and retraction of each punch. Our technique requires no user intervention. Ashraf and Wong [7] semi-automatically labelled motions with sparse timing information, such as zero-crossings of the second derivative of user-specified joint angles. Timewarp curves were generated through a greedy algorithm that matched identical labels (e.g., the same joint angle had experienced a zero-crossing). Our algorithm generates dense correspondences and uses a dynamic programming technique with stronger optimality properties. The method of Bruderlin and Williams [15] is the most similar to our own, as it also is based on dynamic programming. However, their algorithm does not yield one-to-one timewarp curves (and hence does not provide an invertible mapping between frames), and it is only designed for two input motions. We extend this method to produce timewarp curves that provide one-to-one mappings for example sets containing an arbitrary number of motions.

2.4.2 Root Blending

Most previous work has used linear interpolation to compute blended root parameters, which can cause the blended root trajectory to collapse if the input trajectories are not quite similar. A notable exception is the work of Park et al. [68], where the root was positioned and oriented according to a user-specified path (Gleicher [32] used a similar method to interactively edit individual motions). Blending was then used to determine individual skeleton poses appropriate for the local speed and curvature of the root trajectory. We blend root parameters directly, so there is no need to specify the root path. Also, Park et al.'s method was based on the assumption that the input motions were sufficiently smooth that the root path could be well approximated by a circular

arc. Our method is applicable to motions with sharply varying path curvatures (see, for example, Figure 5.17 of Chapter 5).

2.4.3 Automatic Constraint Annotation

Bindiganavale and Badler [10] automatically detected contact constraints between end effectors and other objects by identifying when acceleration zero-crossings coincided with close proximity to items of interest. This method is suitable only for finding constraints that actually occurred in a captured motion, whereas for blends we are interested in the constraints that *should* be satisfied. Rose et al. [76] determined constraints on a blend by manually specifying corresponding constraints in the input motions and blended the boundaries over which these constraints were active. Our method is similar, except corresponding constraints are identified automatically. Ashraf and Wong [8] copied the constraints from the input motion with the currently highest blending weight. While this approach works well for the special case of transitions, it can produce artifacts with other blending operations such as interpolation. In particular, when the blend weights are nearly equal a small change in the weights may produce large changes in the constraints. Our technique ensures that constraints change smoothly if the blend weights change smoothly.

2.5 Parameterized Motion

Wiley and Hahn [96] and the Verbs and Adverbs system of Rose et al. [76] pioneered the technique of building parameterized motions from interpolations of captured examples, and the Verbs and Adverbs system in particular introduced the now-popular approach of applying scattered data interpolation methods to approximate the mapping between motion parameters and interpolation weights. This dissertation builds upon this work by introducing a new search algorithm for extracting the initial example motions from a data set and by providing a more efficient and robust method for building very general kinds of parameterizations. The remainder of this section expands on previous research relating to searching motion data sets and parameterizing the space of interpolated motions.

2.5.1 Searching for Example Motions

Searching a motion data set for segments similar to an example motion is related to the time sequence retrieval problem, which has been studied by the database community for over a decade. Given a distance metric and a query time sequence, the task is to search a database for time sequences whose distance to the query is either below a threshold ϵ or among the k smallest. Most proposed solutions follow the GEMINI framework proposed by Faloutsos et al. [22]. First, a low-dimensional approximation is extracted from each time series in the database. Example approximations include the first few coefficients of a Fourier [2] or wavelet [18] transform, the average values in adjacent windows [43], and bounding boxes [92]. Next, a distance metric is defined over this approximation that underestimates the true distance between the time series. Finally, the approximated signals are stored in a spatial data structure such as an R-tree [36].

This approach provides efficient pruning of portions of the database that are distant from the query while keeping dimensionality low enough that spatial access methods remain viable [12]. However, it also determines similarity through direct numerical comparison, comparing a computed distance against a user-supplied threshold. This makes it impossible to distinguish unrelated motion segments from variations of the query that are numerically quite different — the only discriminant used in the search process is the distance measure, and both kinds of motion are distant from the query. In particular, finding increasingly different variations of the query requires using higher and higher distance thresholds, leading to a continually growing collection of spurious results that must be manually discarded by the user. Our strategy is instead to find close motions and then use them as new queries, allowing one to ultimately find motions distant from the query without resorting to higher thresholds. This strategy is inspired by manifold learning algorithms that use local neighborhoods of points to infer the structure of a low-dimensional manifold embedded in a high-dimensional space [79, 87]. Jenkins and Matarić [42] used a similar strategy to extract motion primitives from human motion data, although their focus was on controlling robot motion, rather than producing high-fidelity animation. A second important problem with the GEMINI framework is that in general it does not provide precise start and end frames for each match, but rather returns a larger range of frames that surrounds the "true" match. This is because perturbing

the boundaries of a motion segment will usually produce only a small change in distance to the query, and unless the initial threshold is quite tight these perturbations will also be returned by the search algorithm. Our search method effectively returns "local minimum" matches that are optimal under perturbation, preventing the user from having to tune boundary frames manually or hunt through nearly identical results. A final limitation of existing search algorithms is that they assume the distance between individual data elements (i.e., skeletal poses) can be computed with an L_p norm. Frame distance metrics that represent orientations with quaternions [52] fail this criterion. We allow arbitrary distance metrics to be used for individual frames.

Content-based database retrieval has appeared in a number of graphics contexts, including images [17], video [91], and 3D models [27]. To search motion capture data sets, Cardle et al. [16] and Keogh et al. [44] used variants of the GEMINI framework. Liu et al. [59] automatically extracted keyframes for each motion in a database and used these keyframes to construct a hierarchical tree of clusters of motions, with deeper levels of the tree corresponding to joints deeper in the skeletal hierarchy. To process a query, the closest leaf cluster was found and its motions were directly compared against the query. This algorithm also uses a direct numerical comparison to determine similarity, and it is designed to compare entire motions against a query, whereas our algorithm is also able to compare subsections of motions.

The problem of searching for motions is related to that of generating descriptive labels. Arikan et al. [6] used support vector machines to automate the process of annotating a motion data set. Other researchers have developed automated methods for clustering and segmenting [63, 46, 9] motion data based on behavioral similarity, which may also be viewed as mechanisms for generating labels. While none of these techniques are designed to provide precise boundaries between different actions, even a coarse set of annotations can make database search more efficient and robust (see Section 6.2 of Chapter 6).

2.5.2 Parameterizing Interpolations

Many methods for building parameterized motions are inaccurate in the sense that they do not ensure that the actual parameters of synthesized motions are the same as the desired parameters.

This is reasonable for qualitative properties like "happiness", where strict accuracy is neither quantifiable nor necessary [90, 76]. Similarly, strict accuracy is unnecessary if motion interpolation is not being used to directly control parameters of interest. For example, while Park et al. [68] used motion interpolation to create locomotion with specified speed and curvature, the actual path of the root was determined through a user-specified trajectory. Nonetheless, in many cases accurate parameterizations are essential. Rose et al. [78] improved the accuracy of scattered data interpolation by adding additional samples to parameter space. Specifically, they identified sets of target parameters for which the accuracy was particularly poor and used gradient descent to find interpolation weights that yielded those parameters. We offer an alternative solution based on more directly sampling the space of interpolations. The general strategy of sampling the space of interpolations was previously used by Wiley and Hahn [96] to obtain regular samplings of parameter space and by Zordan and Hodgins [102] to generate dense sets of example motions as an aid for inverse kinematics tasks, although these efforts were not focused on improving the accuracy of parameterization. Also, we expand on this previous work by showing how to restrict interpolation weights to reasonable values and how to sample in a way that scales well to large numbers of examples.

Previous work on parameterized motions has performed scattered data interpolation by computing a best-fit linear map between interpolation weights and motion parameters and then adding radial basis functions centered on each example [76, 68, 78]. This can produce interpolation weights containing large negative weights [3], and if the user requests parameters far from the examples, interpolation weights are based purely on the linear approximation and hence are effectively arbitrary. Also, the run time of this algorithm is $O(n)$ for n example motions. We propose instead using k-nearest-neighbors interpolation, as suggested by Allen et al. [3]. This allows us to explicitly constrain interpolation weights to reasonable values, project points outside the accessible region of parameter space back onto it, and compute interpolations in time that is nearly independent of the number of example motions.