Chapter 1

Introduction

Animated humans are an important part of a diverse range of media, and they are commonplace in entertainment, training, and visualization applications. At present they are used as characters in games and for special effects in movies; they are part of simulations used by the military to prepare soldiers and by industry to instruct workers in using equipment; and they are used as visualization aids for medical analysis (studying an injured person's gait) and equipment design (determining if controls can be comfortably accessed). Moreover, there is every reason to believe that the demand for animated humans will grow in the future. Because much of our lives are spent observing and interacting with other people, animated humans are a natural and essential part of any visual medium designed to tell stories or simulate real world events.

While animated humans may take the form of cartoons or caricatures, realism is often desirable: more realistic characters can make entertainment applications more engaging and simulations more immersive. However, realistic human motion is quite difficult to animate. The human body is an extremely complicated structure that is capable of actions ranging from subtle and nuanced (such as a shrug) to highly dynamic (such as a back flip). At the same time, people are natural experts on human motion. Having seen other people move our entire lives, we are highly attuned to the subtleties of human movement. For example, a person's mood can be judged purely from body language, and people can identify acquaintances from a distance simply by observing how they walk. As a result, even casual viewers can readily identify inaccuracies in animated motion,

and animations must be of considerable fidelity if they are to be considered realistic by human observers.

The challenge of creating realistic motion is compounded by the fact that one typically needs not just one motion, but many motions. A synthetic actor in a movie, for example, must alter its motion if the director decides to change the script. An agent in a training simulation must adapt its actions to reflect the user's behavior. A character in a game must respond to user input and to changes in the environment. In each case one needs characters with a wide range of possible motions, and these motions must not just be realistic but also *controllable* in the sense that they can be tailored to high-level directions.

In principle, realistic motion can be animated by manually specifying a sequence of character poses, which is known as keyframing. However, this requires an investment of time and artistic talent that is prohibitive for most applications — just a few seconds of high-quality motion may take a skilled artist days to animate. A second possibility is to physically simulate the movement of the human body and compute joint motions that correspond to the desired action [23, 24, 25, 38, 39, 51, 58]. While this approach guarantees that motions are physically plausible (at least within the limitations of the underlying body model), motions that are physically plausible may nonetheless appear unnatural, awkward, or robotic. This is because important properties of real human motion, such as "personality", are neither described nor enforced by physical laws. More generally, the qualities that make motion look human are not readily expressed through sets of equations.

An increasingly popular alternative to keyframing and physical simulation is to record the movement of a live performer, a process known as *motion capture*. Example motion capture technologies include mechanical armatures, magnetic sensors, and specialized multi-camera systems; see Sturman's tutorial [85] for a brief history and Menache's book [61] for a more in-depth introduction. Motion capture data can be used to create high-fidelity animations of effectively any motion that a real person can perform, and it has become a standard tool in the movie and video game industries. However, because motion capture can only be used to reproduce a recorded movement, it offers little control over a character's actions. Imagine, for example, that we want to animate a character walking around in a virtual environment. The character must be able to travel at different

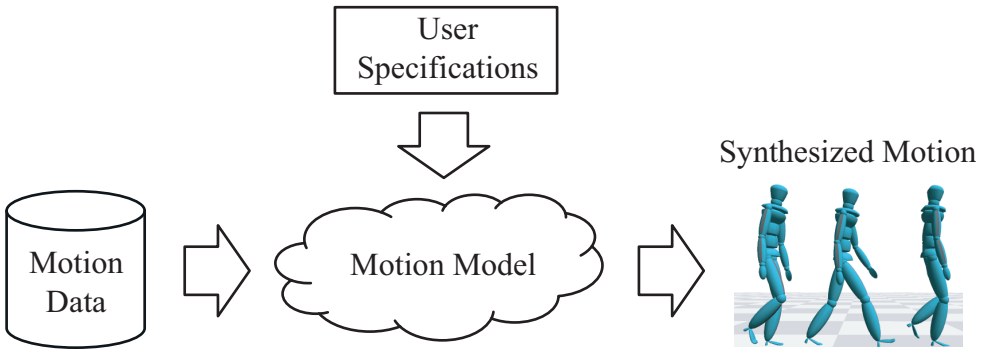

Figure 1.1: Schematic for data-driven motion synthesis.

speeds, follow both smoothly and sharply changing paths, and avoid obstacles. The data, however, only allows the character to perform fixed motions from a limited pool of options. Clearly, it is not possible to capture every variation of every possible way of walking that might be needed, nor is it possible to capture every sequence of steps that might be taken. Similarly, it is infeasible to adjust an animation by directly editing motion data. Just a few seconds of data involves over a thousand parameters, and these parameters are coupled in complicated ways. Manually adjusting data values is analogous to editing a video by changing the colors of individual pixels.

While it is impractical to edit motion data directly, the basic idea of using captured motions to make new ones is nonetheless sound, provided that the data is first converted to a more useable form. *Data-driven motion synthesis* uses example motions to build generative models that can synthesize new, realistic motions through a high-level user interface (Figure 1.1). The goal of data-driven synthesis is to preserve the quality of the original examples while providing intuitive control over a character's actions.

Although the capabilities of data-driven synthesis depend on the underlying motion model, in general this approach can only reliably produce motion that is reasonably similar to the examples. If one wants a flexible motion model that can animate characters with a wide range of movement, then one must start with a large data set. However, previous research has focused almost exclusively on small data sets consisting of a single motion or a handful of motions. This is mainly for practical reasons, as it is only in recent years that it has become economical to acquire larger amounts of data. Nonetheless, it is now common for industry-scale projects to incorporate data sets

containing hundreds or thousands of individual actions. While this makes it possible to animate characters with expressive ranges of motion, it also necessitates a much greater investment of time and effort, since existing methods for building motion models from raw data require considerable labor from the user.

The thesis of this dissertation is that highly automated methods can be used to build generative motion models that offer high-level control and preserve motion realism. Automation is crucial for working with the larger data sets needed for flexible generative models. However, automation also makes it more difficult to guarantee controllability and realism, because a user will no longer be directly overseeing every aspect of a model's construction. To provide control, we develop models that allow users to create motion just by specifying high-level properties, rather than finer details. For example, a walk might be generated by specifying the path that is to be traversed, rather than the location of each individual footstep, and a reach might be created by supplying the location of the target object, as opposed to a detailed trajectory that the hand is to follow. This allows users to direct motion without having to understand the details of the underlying model. To preserve the realism of the captured motions, our models explicitly limit the degree to which generated motion can deviate from the original data, and the user can control this limit to make an appropriate tradeoff between motion quality and model flexibility. Our models also guarantee that synthesized motion preserves the smoothness (C_1 continuity) of the captured motions. This is important because sharp or jittery changes to motion are almost always unrealistic, since real people cannot change their pose instantaneously. Finally, our models infer and enforce kinematic constraints on synthesized motions, which helps ensure that important interactions with the environment are maintained.

Different motion models provide different means of controlling motion. We focus on two motion models that offer complementary forms of control (Figure 1.2): *motion graphs*, which control the sequencing of multiple actions, and *motion blends*, which control the properties of individual actions. A *motion graph* is a directed graph where edges correspond to motion clips and nodes indicate choice points where different clips can seamlessly connect. Motion graphs allow arbitrarily long motions to be constructed from shorter pieces, and they can be used both

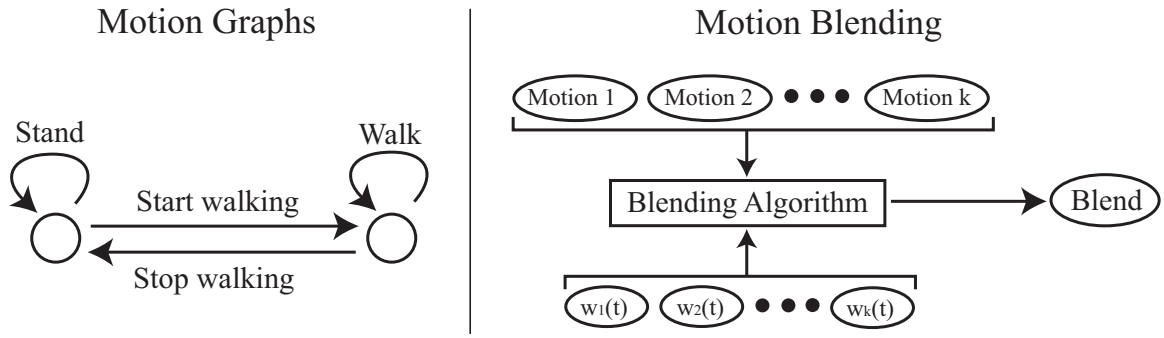

Figure 1.2: Schematic representations of motion graphs and motion blending.

for local control (choosing the next action based on current circumstances) and global control (creating a sequence of actions with desired properties). *Motion blending* combines input motions according to time-varying weights, creating new motions "in-between" the examples. By choosing appropriate weight functions, one can smoothly transition between motions, interpolate motions, and exert continuous control over a motion. Also, given multiple variations of the same action, blending can be used to construct a *parameterized motion*, which is a continuous family of related actions parameterized on high-level features. For example, a parameterized punch might allow an animator to create a specific punch simply by specifying its speed and target location. We have chosen to focus on graph-based and blending-based synthesis because these models are by far the most commonly used, and they appear frequently in previous research [14, 63, 68, 69, 76, 78, 96] and have proven successful in industry [62, 88]. The extension of our methods to other models (in particular, hybrid graph/blending-based models [76]) is beyond the scope of this dissertation, and is left for future work.

This dissertation provides automated methods for building graph-based and blending-based motion models from sets of example motions. The automation not only significantly reduces the time and effort needed to build a model, but also makes it feasible to work with the large data sets needed for expressive models. When combined with motion capture technology as a source of large numbers of high-quality examples, our methods provide a general mechanism for quickly creating realistic motions that meet specific requirements, whether they be provided by a human or by an artificial intelligence module. Four main technical contributions are made:

1. **Efficient and smooth enforcement of kinematic end effector constraints (Chapter 3).** We introduce a constraint enforcement algorithm that can be used to automatically adjust motions synthesized through motion graphs and motion blends so they preserve interactions with the environment, effectively expanding the range of motion that these models can successfully produce.

2. **Automated motion graphs (Chapter 4).** We present a method for automatically building motion graphs from raw motion data, and we further present an automated search framework for extracting motions from these graphs that satisfy high-level constraints.

3. **Improved automatic motion blending (Chapter 5).** We develop an automated blending algorithm based on a novel data structure called a *registration curve*. Registration curves encapsulate relationships between the input motions that currently must be specified manually [68, 76, 78], plus additional information that is important for creating blends. This allows us not only to automate previous blending algorithms, but also to expand the range of motions that can be successfully blended.

4. **Automated construction of parameterized motions (Chapter 6).** To help users collect different variations of the same action, we present a search algorithm for extracting sets of logically similar motion segments from a data set. We also introduce a technique for building accurate parameterizations that is more efficient and stable than existing methods.

Figure 1.3 depicts how these chapters relate to one another. While not directly concerned with motion graphs or motion blends, the constraint enforcement algorithm of Chapter 3 is a vital component of our strategy for preserving motion quality during data-driven synthesis, and hence it is an essential part of this dissertation. Graph-based synthesis is considered in Chapter 4, which may be read independently of the remaining two technical chapters. Both Chapter 5 and Chapter 6 address blending-based synthesis, with Chapter 5 discussing the automation of blending itself and Chapter 6 focusing on parameterized motion as an important application of blending.

Although we will use practical applications as motivation, the methods in this dissertation are not restricted to any particular domain (e.g., movies versus games). However, different applications

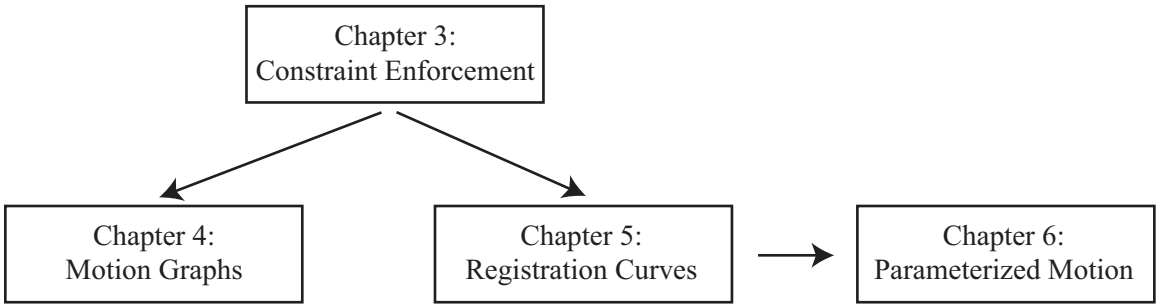

Figure 1.3: Logical relationships among Chapters 3–6.

will typically have very different requirements in a motion model. For example, a lead movie character that fills most of the screen must adhere to very high quality standards, but may need a only a modest range of motion in order to fine-tune a scripted set of actions. In contrast, characters buried within a crowd of thousands can typically accept lower fidelity motion, but they must also have the flexibility to adapt to complex, changing surroundings. To accommodate these different needs, our motion models are equipped with a small number of thresholds that, roughly speaking, determine how much the original example motions may be altered during synthesis, allowing a user to determine an appropriate tradeoff between model flexibility and motion quality.

The remainder of this chapter presents some preliminary technical material and a brief overview of the methods that will be developed in subsequent chapters.

1.1 Preliminaries

1.1.1 Motion Representation

For our purposes, the human body is represented as an articulated figure, or *skeleton*, consisting of a set of ball-and-socket joints arranged in a tree-like hierarchy (Figure 1.4). Each joint has exactly one parent and is associated with a coordinate system defined relative to that of its parent. The exception is the joint at the top of the hierarchy, called the *root*, which has no parent and is associated with a coordinate system defined relative to a (fixed) world coordinate system. A skeleton is constructed by specifying the number and connectivity of its joints plus the offset $\mathbf{o}_i$ of each joint other than the root, where a joint's offset specifies the location of the origin of its

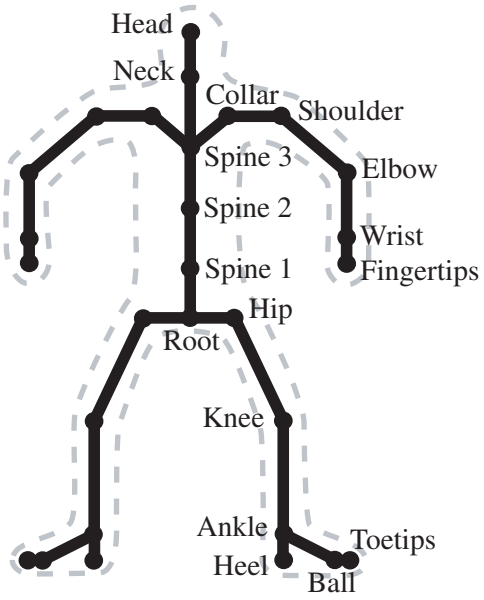

Figure 1.4: Our skeleton model.

coordinate system relative to its parent's coordinate system. The skeleton is typically assumed to have rigid bones, which means that joint offsets are fixed (Chapter 3 relaxes this assumption). All of the experiments performed in this dissertation use the skeleton shown in Figure 1.4. Note that this model can only represent "full-body" motion, since structures such as fingers, toes, and the face are not included. While these details could be added through additional joints, in practice facial and hand capture are performed independently and then integrated with full-body motion as a postprocess.

A *motion* $\mathbf{M}(t)$ is defined as a multidimensional function that specifies the configuration of each of the skeleton's n joints at each point in time:

$$\mathbf{M}(t) = (\mathbf{p}(t), \mathbf{q_1}(t), \ldots, \mathbf{q_n}(t)), \tag{1.1}$$

where $\mathbf{p}$ is the position of the root in the global coordinate system and $\mathbf{q_i}$ is the orientation of the i^{th} joint relative to its parent's coordinate system (for an introduction to coordinate transformations, refer to any standard graphics text, such as Foley et al. [26]). Orientations are represented as unit quaternions [84]; see Grassia [34] for a comparison with other representations, such as Euler angles or orthonormal matrices. Motion data provides a discrete set of skeletal poses, or *frames*,

$\mathbf{M}(t_1), \ldots, \mathbf{M}(t_k)$ that corresponds to a regular sampling of the underlying motion $\mathbf{M}(t)$. We generate root positions in between samples through linear interpolation, and intermediate joint orientations are computed through spherical linear interpolation [84].

While motion data ultimately takes the form of a series of skeletal poses, in practice there is considerable amount of processing needed to obtain this representation from the raw measurements of a motion capture system. In particular, since the human body is not truly a rigid skeleton with simple ball-and-socket joints, the raw measurements must be approximated with a best-fit sequence of skeletal postures. Techniques for performing this fit can be found in the research literature [11, 67, 103] and are standard parts of commercial motion capture processing software. We assume that the conversion of raw measurements to skeletal movement has already taken place.

1.1.2 Constraints

A *constraint* is any property that must be exhibited by a motion. A constraint might state physical limitations, such as a constraint that restricts how far a joint can rotate or prohibits interpenetration of different body parts, or it can relate to properties important to the meaning of a motion, such as a constraint that requires the hands of a clapping character to make contact at certain times. Abstractly, constraints may be thought of as meta-data that attach labels to ranges of frames; these labels might be applied to a single frame, a range of frames, or the entire motion. We assume that motions are already annotated with relevant constraint information. Although automated methods exist for generating these annotations in special cases (e.g., when the character must be in contact with an object in the environment [10, 52]), our experience suggests that manual annotation is preferable if accuracy is desired. We view this as simply another step in the process of converting raw measurements into a useable form, much like fitting skeletal motion to the actual movement of the performer, and we have found that in the context of the full motion capture processing pipeline it is not especially taxing.

1.2 Overview

1.2.1 End Effector Cleanup

Motion graphs and motion blending create new motions through simple procedures that can fail to preserve important features of the original data. One feature that is particularly easy to lose are kinematic constraints on end effectors. These constraints restrict the positions and/or orientations of a hand or foot during specified time intervals, and they typically represent interactions between the character and the environment. For example, one especially common kind of constraint is a *footplant*, which requires part of a foot to remain stationary on the ground. Footplants exist in most circumstances where a character uses its legs to bear weight.

Chapter 3 introduces a new algorithm for adjusting a motion so it satisfies kinematic end effector constraints. While techniques exist for enforcing more general kinds of constraints, they rely on iterative numerical methods that can introduce distracting visual discontinuities into a motion. Our algorithm, in contrast, exclusively uses analytical methods requiring a fixed amount of per-frame computation, and it enforces constraints exactly without introducing discontinuities. This smoothness guarantee is achieved by allowing limbs to change length (which has not been considered in previous work on constraint enforcement) and by explicitly ensuring that changes to individual joint parameters remain continuous when constraints turn on and off. When combined with methods for automatically inferring constraints on motions generated through motion graphs and motion blending (Chapters 4 and 5), our algorithm can automatically correct constraint violations and thereby extend the range of realistic motion that can be produced through these models.

1.2.2 Motion Graphs

Motion graphs (Figure 1.2) have historically been built by hand: an artist determines which segments of captured motion can be connected and uses software tools to adjust them so they can join seamlessly. This can be a tedious and time-consuming process. Chapter 4 shows how a motion graph can be automatically constructed by identifying places in a data set where motions are locally similar and then synthesizing special transition motions at these points. Similarity

is determined through a novel point-based distance metric that factors out irrelevant differences stemming from each motion's choice of global coordinate system. Constraints from the original data are propagated into transitions and enforced using the methods of Chapter 3, allowing simple linear blending methods to be used without sacrificing properties like crisp footplants.

While our approach allows motion graphs to be built with little effort, these motion graphs are generally quite complicated, containing many individual clips with no clear organization. Rather than having a user directly interact with such a graph, Chapter 4 introduces a general search framework for finding a sequence of edges that optimizes a user-supplied objective function and obeys restrictions on what kinds of motion are legal. To demonstrate the potential of this approach, we apply this framework to the problem of creating motion that traverses arbitrary paths.

1.2.3 Motion Blending

Most blending algorithms (Figure 1.2) are not automatic in the sense that they require the user to add special annotations to the input motions. Although automatic algorithms exist, they are quite fragile and can only be used in rather special circumstances. In particular, they have three principal limitations:

1. **Timing differences are not accounted for.** Logically corresponding events in two motions usually occur at different absolute times; for example, the time between successive heelstrikes is shorter in a hurried walk than in a relaxed walk. Existing automatic blending methods ignore timing differences, causing them to combine frames from structurally unrelated parts of the input motions. The effect of this depends on the input motions, but is usually quite bizarre — a blend of the different walk styles, for instance, would not be a new walk but an odd movement where the legs skip across the ground with only cursory contact.

2. **Only very similar root trajectories can be blended.** Existing methods (both manual and automatic) combine root trajectories through simple averaging. If the input root trajectories are not already quite similar, this can cause the blended motion to unnaturally slow down,

produce extreme footsliding artifacts, and create large and discontinuous changes in root orientation.

3. **Kinematic constraints are not applied to blends.** Existing blending algorithms combine motions through relatively simple procedures that fail to preserve kinematic constraints. While these constraints can be enforced as a postprocess (using, e.g., the methods of Chapter 3), the constraints must first be known. Existing automatic algorithms are unable to determine what constraints should be applied to a blend, preventing the application of constraint satisfaction algorithms.

Chapter 5 provides an automatic blending algorithm that avoids these problems. The heart of this algorithm is the construction of a data structure called a registration curve, which encapsulates relationships among the timing, local coordinate frame, and constraint state of arbitrarily many input motions. This information can be used to avoid the problems discussed above without resorting to user intervention. Chapter 5 shows how to automatically build registration curves and use them to create blends, and it demonstrates registration curves in common blending applications such as transitioning, interpolation, and continuous control.

1.2.4 Parameterized Motion

Given a set of example motions representing different variations of the same kind of action, motion blending can interpolate/extrapolate these examples to produce new motions with different properties. For example, given a set of punches that target different locations, interpolation could be used to create punches that target new locations. In principle, a user could then control the action simply by adjusting interpolation weights. However, this is an awkward form of control because interpolation weights typically have no simple connection to high-level motion properties. To correct this, a map can be built between relevant motion features (such as the target location of a punch) and interpolation weights, resulting in a *parameterized motion* that can be controlled directly through these features (Figure 1.5) [76, 96].

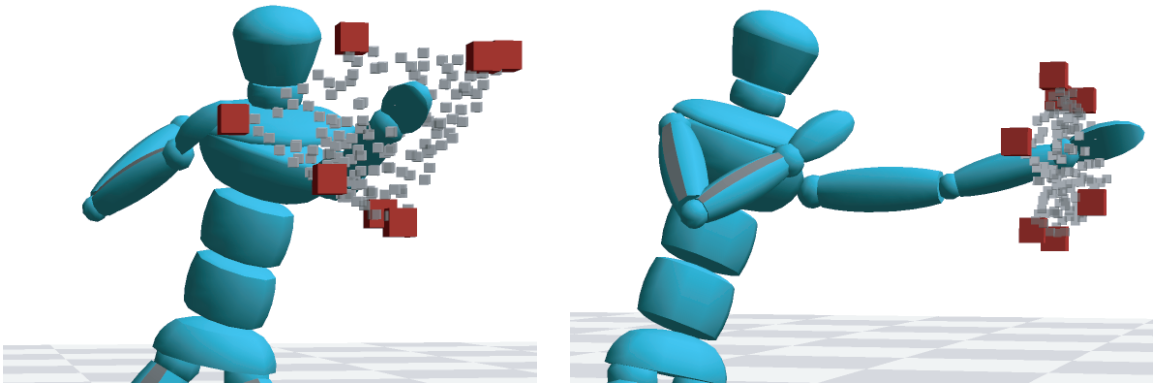

Figure 1.5: A parameterized punch. Red cubes show target locations of captured punches and grey cubes indicate the range of target locations that can be reached by interpolating the examples.

Existing methods for constructing parameterized motions are designed for small data sets consisting exactly of separate example motions that evenly sample a predefined range of variation. In a large data set, however, the necessary examples will in general be buried within longer streams of motion. Currently users must manually scan through the data and extract the segments of interest. This task is complicated by the fact that they must not just find *some* pieces of motion that contain the desired examples, but must rather carefully crop out a set of actions that begin and end in analogous skeletal postures — for example, a parameterized walk must be built from examples that all begin and end at the same point in the locomotion cycle. Also, in a large data set the available range of variation for a given action is generally not known in advance, which is problematic since existing parameterization techniques implicitly assume that the accessible region of parameter space has a simple, known boundary.

Chapter 6 shows how to automate the extraction of examples through a novel database search algorithm that efficiently locates motion segments logically similar to (but perhaps numerically distinct from) a query motion. It also extends existing work on parameterization by providing a more efficient algorithm that ensures accuracy and guarantees that only reasonable blends can ever be created (that is, it explicitly limits the allowable amount of extrapolation from the data). Our methods are demonstrated on a 37,000-frame data set (about ten minutes of motion sampled at 60Hz) containing a variety of different kinds of motion.