

Automated Extraction and Parameterization of Motions in Large Data Sets

Lucas Kovar*
University of Wisconsin-Madison

Michael Gleicher
University of Wisconsin-Madison

Abstract

Large motion data sets often contain many variants of the same kind of motion, but without appropriate tools it is difficult to fully exploit this fact. This paper provides automated methods for identifying logically similar motions in a data set and using them to build a continuous and intuitively parameterized space of motions. To find logically similar motions that are numerically dissimilar, our search method employs a novel distance metric to find “close” motions and then uses them as intermediaries to find more distant motions. Search queries are answered at interactive speeds through a precomputation that compactly represents all possibly similar motion segments. Once a set of related motions has been extracted, we automatically register them and apply blending techniques to create a continuous space of motions. Given a function that defines relevant motion parameters, we present a method for extracting motions from this space that accurately possess new parameters requested by the user. Our algorithm extends previous work by explicitly constraining blend weights to reasonable values and having a run-time cost that is nearly independent of the number of example motions. We present experimental results on a test data set of 37,000 frames, or about ten minutes of motion sampled at 60 Hz.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: motion capture, motion synthesis, motion databases

1 Introduction

Large motion data sets are now commonplace in real-world projects that require expressive character animation. These data sets are valuable not only because they contain many different kinds of actions, but also because any particular action can have many variants. This provides animators with flexibility in selecting motions appropriate for specific circumstances. In particular, collections of related motions can serve as raw material for algorithms that make continuous, parameterized spaces of motion, allowing one to do things like animate a kick simply by stating where the target is. In this manner, large motion capture data sets bring us closer to the goal of being able to create realistic motion simply by specifying what it is supposed to do.

This goal is still far from reality, however. A family of related motions is hard to exploit when scattered inside a data set; the individual motions must first be identified and extracted. To do this, users currently must scan through the data set and manually crop the frames of interest. This can be tedious and time-consuming, even

when the data set is annotated with descriptive labels. For example, a data file labelled “punch” might contain many individual punches and related but distinct actions such as dodging a counter-blow.

While there is value in simply seeing what a data set has to offer, a user often has a particular motion in mind, such as a punch that targets a specific location. In this case blending techniques can be applied to the extracted motions to produce a *parameterized* motion that gives a user direct control over relevant motion properties. However, existing methods are designed for small data sets consisting exactly of example motions that evenly (though not regularly) sample a predetermined range of variation. In a large data set, the user may find many example motions, some of which are redundant and some of which are quite dissimilar, and these examples will in general yield a space of synthesizable motions whose boundaries are hard to predict in advance.

This paper presents automated tools for locating logically similar motion segments in a data set and using them to construct parameterized motions that provide accurate and efficient control. The remainder of this section presents an overview of our methods and a summary of our contributions. Section 2 discusses related work. Next, Section 3 provides details on how we search for similar motion segments and presents some experimental results. Section 4 then explains how we build parameterizations and illustrates our technique on several examples. Finally, Section 5 concludes with a brief discussion of the advantages and limitations of our methods.

1.1 Overview

1.1.1 Searching Motion Data Sets

Given a segment of the motion data set (the *query*), our system automatically locates and extracts motion segments that are “similar”, i.e., that represent variations of the same action or sequence of actions. Our method uses three key ideas, each of which is a contribution to the problem of searching a motion data set.

1. **Multi-step search.** Logically similar motions can have very different skeletal poses (Figure 1), but existing computational methods can only reliably find motions that are *numerically* similar to the query in the sense that corresponding skeletal poses are roughly the same (see Section 2.1). On the other hand, in a large data set it is likely that some logically similar motions will also be numerically similar. We therefore add robustness to the search by concentrating on finding these closer motions and then using them as new queries in order to find more distant motions.
2. **Using time correspondences to determine similarity.** A simple way of measuring numerical similarity is to identify corresponding frames and compare their average distance to a threshold value. We complement this by analyzing the correspondences themselves: similar motions are required to have “clear” time correspondences. This is based on the intuition that if two motions are similar, it should be easy to pick out corresponding events.
3. **Interactivity through precomputation.** To provide interactive speeds, we do not start each search from scratch. Instead,

*e-mail: {kovar,gleicher}@cs.wisc.edu

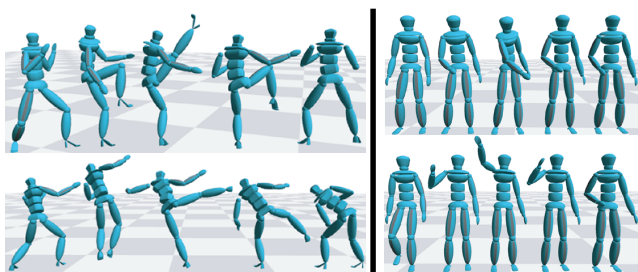


Figure 1: Logically similar motions may be numerically dissimilar. **Left:** A standing front kick vs. a leaping side kick. Note the differences in the arms, torso posture, and kick trajectory. **Right:** While these two reaching motions have somewhat similar skeletal postures, the changes in posture are in completely opposite directions.

we precompute a *match web*, which is a compact and efficiently searchable representation of all possibly similar motion segments.

1.1.2 Creating Parameterized Motions

Once example motions have been located, they can be blended to make new motions. A blend is effectively a weighted average of the examples, and the set of all blends forms a continuous space of related motions. This space can be converted into a parameterized motion through a user-specified *parameterization function* \mathbf{f} that picks out relevant motion features. For example, \mathbf{f} might compute the position of the hand at the apex of a reach or the average speed and curvature of the root path during a walk cycle.

Abstractly, \mathbf{f} maps blend weights to motion parameters. Our goal is to compute \mathbf{f}^{-1} : given a set of target parameters, we want blend weights that produce the appropriate motion. Since \mathbf{f}^{-1} in general has no closed form representation, it is common to approximate it with scattered data interpolation methods that assign each example motion a blend weight based on the distance between its parameters and the target parameters [Rose et al. 1998]. We use a similar approach, but offer several improvements over previous work:

1. **Automation.** The user’s only task is to supply \mathbf{f} ; our system handles the rest. In particular, we extend the method of Kovar and Gleicher [2003] to automatically register the example motions so they can be blended.
2. **Blend Weight Constraints.** Blending can only reliably create new motions near the examples, which means only a finite region of parameter space is accessible. In particular, blend weights should be convex or nearly convex so the blends will represent interpolations or limited extrapolations of the examples. Existing methods place no constraints on blend weights and can break down when specified parameters are far from those of the example motions. We ensure that blend weights are convex or nearly convex, thereby limiting the allowable amount of extrapolation and projecting unattainable parameter requests back onto the accessible portion of parameter space.
3. **Accuracy.** If the examples are not sufficiently close in parameter space, direct application of scattered data interpolation may yield inaccurate results (Figure 2). We correct this by automatically sampling the set of valid blends in order to densely sample the accessible region of parameter space.
4. **Efficiency.** Large data sets may contain many example motions. We automatically identify and remove redundant examples to reduce storage requirements. Also, while previous methods have used scattered data interpolation algorithms that

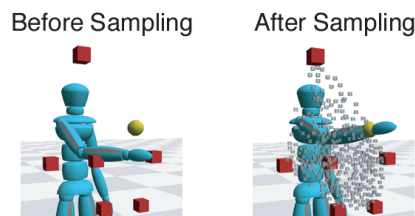


Figure 2: **Left:** Six example reaching motions create a sparse sampling of parameter space that leads to an inaccurate parameterization. The dots indicate parameter samples and the yellow sphere shows the desired location of the wrist. **Right:** We automatically generate a denser sampling that provides greater accuracy.

require $O(n)$ time for n examples, our method’s run time is nearly independent of the number of examples.

2 Related Work

2.1 Searching for Motion

Our search problem is related to time sequence retrieval, 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. [1994]. First, a low-dimensional approximation is extracted from each time series in the database. Example approximations include the first few coefficients of a Fourier [Agrawal et al. 1993] or wavelet [Chan and Fu 1999] transform, the average values in adjacent windows [Keogh et al. 2001], and bounding boxes [Vlachos et al. 2003]. 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 [Guttman 1984].

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 [Böhm et al. 2001]. However, one drawback is that a direct numerical comparison is used to determine similarity. With existing metrics, a large distance may reflect either that motions are unrelated or that they are different variations of the same action, and there is no way to distinguish between these two cases. On the other hand, a small distance is a reasonable indicator of similarity because it implies that individual poses are numerically similar. Our strategy is hence to find close motions and then use them as new queries to find more distant motions, allowing us to use lower and more reliable distance thresholds without sacrificing the ability to find motions that are numerically distinct from the query. 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 [Roweis and Saul 2000; Tenenbaum et al. 2000]. Jenkins and Mataric [2002] 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 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 [Lee et al. 2002; Wang and Bodenheimer 2003] 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 [Castelli and Bergman 2001], video [Veltkamp et al. 2001], and 3D models [Funkhouser et al.

2003]. To search motion capture data sets, Cardle et al. [2003] used a variant of the GEMINI framework discussed earlier in this section. Liu et al. [2003] 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.

In order to automatically construct transitions, several recent research efforts have identified locally similar regions in a motion data set [Arikan and Forsythe 2002; Arikan et al. 2003; Kim et al. 2003; Kovar et al. 2002; Lee et al. 2002; Wang and Bodenheimer 2003]. We, in contrast, are interested in finding entire motions that are similar. For the case of rhythmic motion, Kim et al. [2003] automatically identified similar motions by using beat analysis to segment a motion data set and then clustering motions based upon a similarity metric. Our method applies to more general data sets and is geared toward content-based search rather than clustering.

The problem of searching for motions is related to that of generating descriptive labels. Arikan et al. [2003] used support vector machines to automate the process of annotating a motion data set. While their technique was not designed to provide precise boundaries between different actions, even a coarse set of annotations can make database search more efficient and robust (see Section 3).

2.2 Parameterizing Blends

Wiley and Hahn [1997] and the Verbs and Adverbs system of Rose et al. [1998] pioneered the technique of building parameterized motions from blends of captured examples. Subsequent papers have further developed the underlying blending mechanisms [Kovar and Gleicher 2003] and have provided new methods for parameterizing the space of blends [Rose et al. 2001; Park et al. 2002]. This previous work has assumed that the example motions have been identified and cropped from the original data set, whereas we provide an automated method for extracting them. Additionally, we show how to build accurate parameterizations that offer greater run-time efficiency than previous methods and that guarantee that blend weights have reasonable values. The remainder of this section expands on previous work relating to the parameterization of blends.

Many methods for building parameterized motions 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 not necessary [Unuma et al. 1995; Rose et al. 1998]. Similarly, strict accuracy is unnecessary if blending is not being used to directly control parameters of interest. For example, while Park et al. [2002] used blending 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. [2001] improved the accuracy of scattered data interpolation by adding additional samples to parameter space. Specifically, they identified sets of target parameters for which the approximation was particularly poor and used gradient descent to find blend weights that yielded those parameters. We offer an alternate solution based on more directly sampling the space of blends. The general strategy of sampling the space of blends was previously used by Wiley and Hahn [1997] to obtain regular samplings of parameter space and by Zordan and Hodgins [2002] 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 blend weights to reasonable values and how to sample in a way that scales well to large numbers of examples.

A number of previous efforts have performed scattered data interpolation by computing a best-fit linear map between blend weights and motion parameters and then adding radial basis functions centered on each example [Rose et al. 1998; Park et al. 2002; Rose et al. 2001]. This can produce blend weights containing large negative weights [Allen et al. 2002], and if the user requests parameters far from the examples, blend 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. [2002]. This allows us to explicitly constrain blend weights to reasonable values, project points outside the accessible region of parameter space back onto it, and compute blends in time that is nearly independent of the number of example motions.

One popular application of parameterized motion is to control inverse kinematics (IK) tasks such as reaching [Wiley and Hahn 1997; Rose et al. 2001]. Grochow et al. [2004] recently introduced an alternate approach based on using motion capture data to construct probabilistic models of individual skeletal poses.

3 Searching for Motions

A motion is a continuous function $\mathbf{M}(t)$ that is regularly sampled into frames $\mathbf{M}(t_i)$, where each frame is a skeletal pose defined by its joint orientations and the position of the root joint. Given a query motion \mathbf{M}_q that is a segment of some motion in the data set, our goal is to find other motion segments that are similar to \mathbf{M}_q , i.e., segments that represent variations of the same action. We refer to these as *matches*. One challenge in finding matches is that individual frames are high-dimensional objects with non-Euclidean distance metrics [Kovar et al. 2002; Lee et al. 2002]. As a result, traditional methods for organizing the data into a spatial hierarchy (such as a BSP-tree) can not be directly applied [Böhm et al. 2001]. A second challenge is that logically similar motions may be numerically dissimilar in the sense that corresponding poses may have very different joint orientations and angular velocities (Figure 1). Traditional search algorithms implicitly equate numerical similarity with logical similarity, and as a result they have difficulty distinguishing motions that are unrelated from those that are different versions of the same kind of action (see Section 2.1).

Our search strategy is to find “close” matches that are numerically similar to the query and then use them as new queries to find more distant matches. Our algorithm for determining numerical similarity allows arbitrary metrics for comparing individual frames, and timing differences are factored out to allow matches to be of different duration than the query. A user executes a search by providing a query and a distance threshold that is used to determine whether two motion segments are numerically similar. Numerically similar matches are then identified and automatically submitted as new queries, and the process iterates until no new matches are found. Since each match spawns a new query, it is crucial that individual queries be processed quickly. In particular, we would like searching to be fast enough that the distance threshold can be tuned interactively. To make this feasible, we preprocess the data set into a *match web*, which is a compact and efficiently searchable representation of all motion segments that, given a sufficiently large distance threshold, would be considered numerically similar.

One difficulty is that numerically similar motions may *not* be logically similar. For example, the overall structure of a man walking looks much like that of a woman walking, and reaching for an object appears quite similar to simply touching it. This problem is hard to correct because it involves high-level understanding of what motions mean. For an unlabelled data set, we require users to independently confirm that matches have the correct meaning. However, most large data sets contain some sort of descriptive labels — if nothing else, a filename or location in the directory hierarchy can

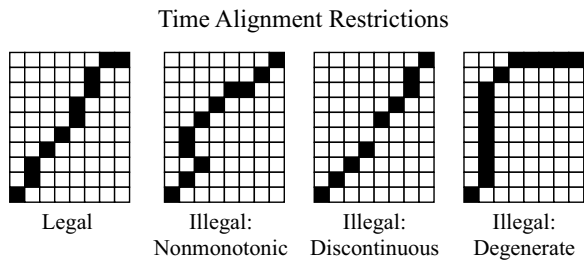


Figure 3: Time alignments must be continuous, monotonic, and non-degenerate.

serve as clues to a motion’s contents. When labels are present, we limit our search to semantically relevant parts of the data set.

In the remainder of this section, we describe our criteria for determining whether two motion segments are numerically similar, explain how to build match webs and use them to quickly answer similarity queries, and present experimental results.

3.1 Criteria for Numerical Similarity

Two criteria are used to determine numerical similarity:

1. Corresponding frames should have similar skeleton poses.
2. Frame correspondences should be easy to identify. That is, related events in the motions should be clearly recognizable.

Both criteria involve frame correspondences, so we first discuss how to obtain these. We require the set of frame correspondences to form a continuous, monotonically increasing, and non-degenerate mapping between frames, which we call a *time alignment* (Figure 3). The non-degeneracy condition limits the time alignment’s slope to be between $\frac{1}{k}$ and k for some $k \geq 1$, which in the discrete case means that at most k frames of one motion can be mapped to a single frame of the other. Intuitively, this restricts motions to being locally sped up or slowed down by at most a factor of k . Given a distance function d for individual frames, dynamic programming can be used to compute an optimal time alignment that minimizes the total distance between matched frames. See Bruderlin and Williams [1995] and Kovar and Gleicher [2003] for details. Several options exist for the frame distance function; we use the one suggested by Kovar et al. [2002].

An analysis of the time alignment determines whether two motion segments satisfy the numerical similarity criteria. We start by computing d for every pair of frames, forming a grid of distances where cell (i, j) specifies $d(\mathbf{M}_1(t_i), \mathbf{M}_2(t_j))$ (Figure 4). The time alignment is a path on this grid from the lower left to the upper right that minimizes the total cost of its cells. To test the first criterion, we find the average value of the cells on this path and compare against a user-specified threshold ϵ . We use the average value rather than the total in order to measure distance independently of path length.

The second criterion can be interpreted in terms of the *local* optimality of the time alignment. If a cell on the time alignment is a horizontal or vertical 1D local minimum, then the frame correspondence is strong in the sense that holding one frame fixed and varying the other only yields more dissimilar skeletal poses. To illustrate, consider the top of Figure 4, which shows the distance grid for two different walk cycles. The magenta path on the left is the calculated time alignment and the highlighted cells on the right show all of the horizontal and vertical 1D minima. Note that nearly every cell on the time alignment is one of these minima. This is because frames at the same point in the locomotion cycle are far more similar than frames that are out of phase. The bottom of Figure 4 repeats this analysis for two kicking motions. While the average distance between corresponding frames is about 5 times higher, frames at related parts of the kick (chambering, extension, and retraction) are

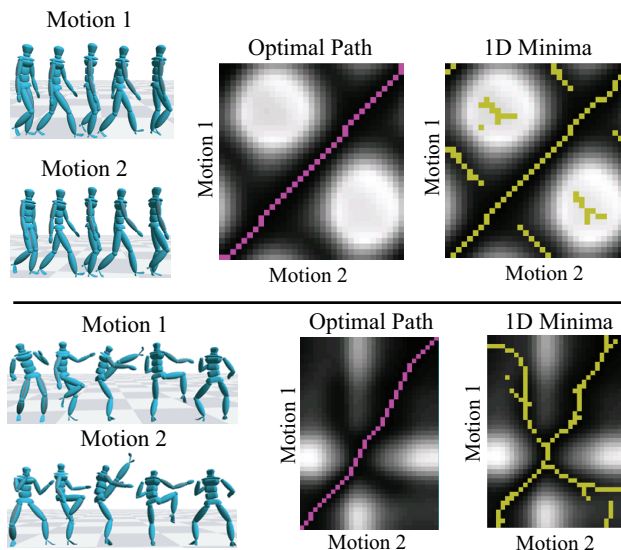


Figure 4: Comparing time alignments (magenta cells on the left) with local minima locations (yellow cells on the right). Darker pixels show smaller frame distances. Local minima locations were *not* used when computing the time alignments. **Top:** Two walk cycles. **Bottom:** Two kicks.

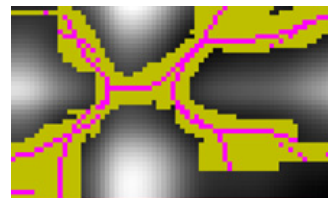


Figure 5: Local minima (magenta) are extended to form the valid region (yellow).

still more similar than pairs of unrelated frames. This is again reflected in the local minima of d : about 60% of the time alignment’s cells are 1D minima, and the rest are close to 1D minima.

Ideally every cell on the time alignment would be a local minimum, since then each correspondence would be “obvious”. In practice, however, this is overly restrictive. The finite sampling rate causes noise in the exact location of minima, and motions that have stretches of similar frames (e.g., pauses) produce basins in the distance grid where the locations of minima are effectively arbitrary. We therefore instead require cells on the time alignment to be *near* 1D minima. To enforce this, we compute all 1D local minima and extend each one along the directions in which it is a minimum (horizontal, vertical, or both) until a cell is encountered whose value is at least α percent larger than the minimum’s value, where α is defined by the user (see Figure 5; we set α to 15%). We call the resulting region on the grid the *valid region*. The time alignment is restricted to the valid region, which involves a simple modification of standard dynamic programming algorithms. If no time alignment can be created under this restriction, then the motions fail the second criterion and are considered dissimilar.

3.2 Match Webs

We now turn to the problem of precomputing all potential matches for each motion segment in the data set. Without loss of generality, we limit the discussion to finding segments in a single motion \mathbf{M}_1 that are numerically similar to another (possibly identical) motion \mathbf{M}_2 ; if a data set has many motions, then they are compared pairwise. Consider the distance grid formed by computing d for

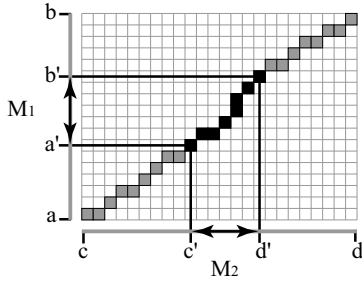


Figure 6: Any subregion of an optimal time alignment for $M_1[a, b]$ and $M_2[c, d]$ is an optimal time alignment for two shorter motion segments $M_1[a', b']$ and $M_2[c', d']$.

every pair of frames and let $M_i[q, r]$ denote the motion segment $M_i(t_q), \dots, M_i(t_r)$. Two motion segments $M_1[a, b]$ and $M_2[c, d]$ are *potentially similar* if there is some distance threshold for which they are numerically similar under the criteria of Section 3.1. For this to hold, there must exist at least one time alignment starting at cell (a, c) and ending at cell (b, d) that is everywhere inside the valid region and obeys the restrictions shown in Figure 3. If not, then $M_1[a, b]$ and $M_2[c, d]$ cannot be numerically similar. Otherwise, dynamic programming provides the optimal time alignment between $M_1[a, b]$ and $M_2[c, d]$.

Two observations can now be made that allow us to compactly represent *all* potentially similar motion segments. First, if cells (a, c) and (b, d) can be connected with a valid time alignment, then it is likely that nearby pairs $(a \pm \delta, c \pm \delta)$, $(b \pm \delta, d \pm \delta)$ can also be connected. In other words, we can perturb the boundaries of $M_1[a, b]$ and $M_2[c, d]$ to find other potentially similar motion segments. It therefore makes sense to identify locally optimal pairs of motion segments where the time alignment has a locally minimal average cell value. Second, any subsection of the optimal time alignment for $M_1[a, b]$ and $M_2[c, d]$ is itself an optimal time alignment for motion segments inside $M_1[a, b]$ and $M_2[c, d]$ (Figure 6). This is a natural consequence of optimality: the path connecting any two cells on a time alignment must be optimal because otherwise we could choose a different path that would lower the overall cost. Hence an optimal time alignment represents an entire family of potentially similar motion segments, not just a single pair.

In light of these observations, we search for long paths on the distance grid that correspond to locally optimal time alignments. We start by looking for chains of 1D minima that satisfy the continuity, monotonicity, and non-degeneracy restrictions. First, we locate all minima that have no neighbors to the left, bottom, or bottom-left, since these cannot be in the interior of a chain. Then for each of these minima we form a chain by iteratively searching the top, right, and top-right cells for other minima, making sure along the way that the non-degeneracy condition is not violated. Since some local minima are spurious — for example, walk cycles that are 180 degrees out of phase have local minima at frames where the legs are closest together — some of these chains will not be meaningful. As a heuristic, we simply remove chains below a threshold length (0.25s in our implementation). Each remaining minima chain is a locally optimal time alignment since moving any cell increases the average distance.

Since the precise location of an individual minimum is somewhat arbitrary (Section 3.1), nearby minima chains that ought to be connected may be separate. To be conservative, we consider connecting any two chains as long as the connecting path is inside the valid region of the grid and has a length (measured via Manhattan distance) less than a threshold L , which was $2s$ in our implementation. For each chain C , we identify other chains C' in the vicinity. We then compute for each cell C_i on C the optimal path to each cell on C' whose Manhattan distance to C_i is less than L . The path with the smallest average distance is retained as the final connection, or

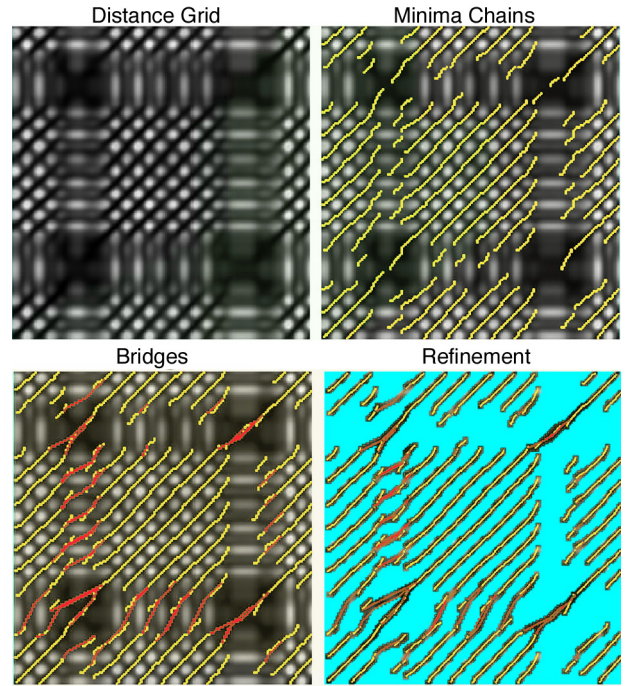


Figure 7: To build a match web, we compute the distance between every pair of frames, find chains of local 1D minima, and add bridges that connect nearby chains. A match web built at low resolution may be refined at a higher resolution.

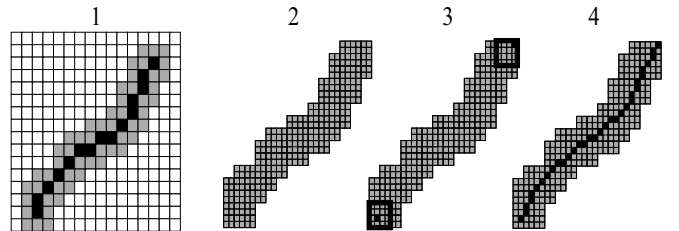


Figure 8: To refine a chain, it is first padded (1) and upsampled (2) to form a search region. New endpoints are then found via local search (3) and connected with an optimal path (4).

bridge, between C and C' . Note that because bridges must be inside the valid region, nearby chains may not be connectable. Also, this restriction makes bridge calculation more efficient because the dynamic programming algorithm processes fewer cells.

The result of this procedure is a network of paths on the distance grid, some of them minima chains and others bridges between chains (Figure 7). This network represents all potentially similar pairs of motion segments, and we refer to it as a match web. Starting from any cell on any path of the match web, we can generate a time alignment by travelling further down the path and possibly branching off onto connecting paths. While this time alignment is not necessarily optimal, it is close to optimal since every cell is either a local minimum or has a distance value close to a nearby minimum. Also, match webs can be stored compactly since we need only retain the grid position and value of each cell on each path.

Match webs can be constructed more efficiently by building them at a low resolution and then refining them at higher resolutions. The lowest resolution match web is built by downsampling M_1 and M_2 and running the algorithm described above. Each minima chain is then refined as shown in Figure 8: first, the chain is padded and upsampled to form a search region, then new endpoints are placed at the smallest-valued cells in the vicinity of the

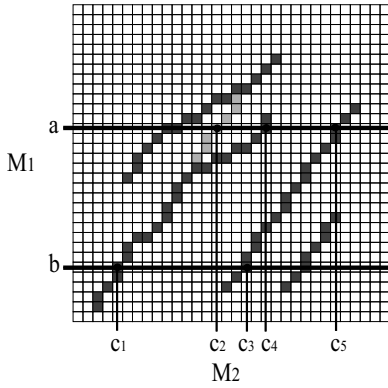


Figure 9: A simple search example for the query $M_1[a, b]$. There are three potential matches: $M_2[c_1, c_2]$, $M_2[c_1, c_4]$, and $M_2[c_3, c_5]$.

old endpoints, and finally these points are joined with an optimal path. Each bridge is handled similarly, except the endpoints are restricted to be on the refined versions of the chains it connects. Figure 7 shows an example of refining a low-resolution match web.

3.3 Searching With Match Webs

Given a match web for M_1 and M_2 , a query motion segment $M_1[a, b]$, and a distance threshold ϵ , we now explain how to locate numerically similar motion segments in M_2 . The search algorithm essentially intersects the match web with the rectangle defined by the region from row a to row b , as shown in Figure 9. Let a *match sequence* be a sequence of cells formed by selecting any cell of any path on the match web and then travelling down that path, possibly branching off onto connecting paths. We are interested in finding all match sequences that span from row a to row b . We start by finding every path that contains a cell in row a . For each such path, the leftmost (i.e., earliest) cell in row a serves as the initial cell of a match sequence. We check whether the path contains cells in row b , and if so, we return a match sequence formed by appending all cells up to the last one in row b . We next consider branching off onto connecting paths to search for additional match sequences. This is done by walking down the path until we either reach its end or a cell in row b , adding cells to the match sequence as we go and recursively processing each connecting path that is encountered.

Each match sequence is a time alignment between the query and a potential match in M_2 defined by the match sequence’s first and last columns. Any match sequence whose average cell value is greater than ϵ is discarded. While each remaining match sequence technically provides a match to the query, some of these matches significantly overlap and hence are redundant. In Figure 9, for example, this is true of $M_2[c_1, c_2]$ and $M_2[c_1, c_4]$. To remove these redundancies, we sort the matches in order of increasing average distance and place them in an array. The first element of this array is then returned as a match, and every other element that overlaps with it by more than a threshold percentage σ is discarded. This procedure iterates until no matches are left.

So far we have found the matches that are closest to the query, which we call first-tier matches. To find more distant matches, the search algorithm is repeated for each first-tier match, yielding second-tier matches. This continues until no new matches are found. In this manner a graph is built where the nodes are motion segments and edges between nodes indicate that the segments are numerically similar. We call this data structure a *match graph* (Figure 10). Each edge is associated with a time alignment and is assigned a cost equal to the average value of the time alignment’s cells. We define the distance between two nodes as the cost of the minimal-cost connecting path on the match graph. The match graph

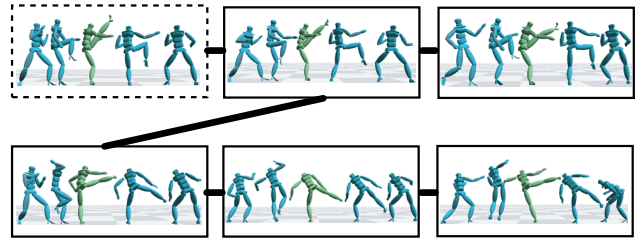


Figure 10: An example match graph. The query is in the dotted box and the other nodes are matches. Edges indicate numerical similarity.

can by itself be of interest since it depicts numerical similarity relationships in the matches. To illustrate, Figure 10 shows a match graph where the query was a front kick; note the progression from front kick to standing side kick to leaping side kick.

When processing queries other than the initial query, we must decide whether each “new” match is a heretofore unseen motion or a duplicate of an existing match. When doing this, we must account for the fact that, in practice, duplicates will overlap a great deal but not span identical intervals of frames. Let M be the current query and M' be a newly identified match. We start by comparing M' against each node and record the one with the greatest degree of overlap, M_{max} . If this maximum overlap is less than a tolerance $\bar{\sigma}$, M' is added to the match graph as a new node along with an edge connecting it to M . If the maximum overlap is greater than σ , then the frame interval of M_{max} is averaged with that of M' and an edge is added between M and M_{max} . Otherwise, M' is discarded. Appropriate values for σ and $\bar{\sigma}$ depend on the query. In general, we have found $\sigma \approx 80\%$ and $\bar{\sigma} \approx 20\%$ to work well, but queries involving multiple periods of a cyclic motion (e.g. walking) require larger values of $\bar{\sigma}$ to allow greater overlap in distinct matches.

3.4 Experimental Results

We tested our match web implementation on a data set containing 37,000 frames, or a little over 10 minutes of motion sampled at 60Hz. This data was divided into thirty files ranging in length from 3s to 75s, and it included both motions where the actor performed a scripted sequence of specific moves and motions consisting of random variations of the same action. The former class of motions included picking up and putting back objects at predefined locations, walking/jogging in a spiral at different speeds, stepping onto/off of platforms of various heights, and sitting down/standing up using chairs of various heights. The latter class of motions consisted of kicks, punches, cartwheels, jumping, and hopping on one foot. All experiments were ran on a machine with 1GB of memory and a 1.3GHz Athlon processor.

Figure 11 shows some query motions and the sets of the matches returned by our system. In each case, the entire search process took less than half a second. Manually cropping matches from the data set with similar precision would be quite tedious, especially in the case of the walking query, where 95 matches were identified.

The remainder of this section provides details on the processing time and storage needed to build match webs and execute searches, presents results on the accuracy of the search, and briefly discusses some advantages and limitations of our approach.

Time and Storage. We initially constructed a match web for the entire data set by building a low resolution version at 10Hz and then refining it in two stages, first to 20Hz and then to 60Hz. The total computation time was 50.2 minutes. Without compression, the size of the match web on disk was 76.2MB. While this is three times the size of the original data, it nonetheless comfortably fits into main memory. Applying a standard compression algorithm (Lempel-Ziv encoding, as implemented in gzip) reduced the size by a factor of

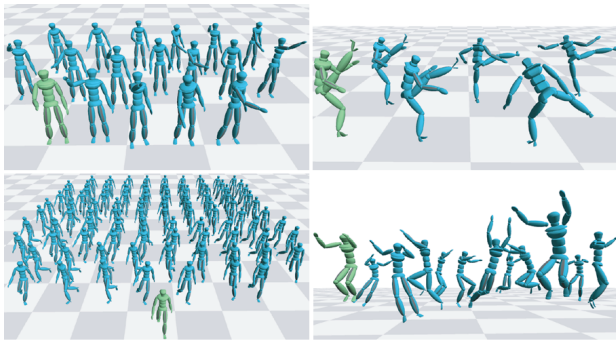


Figure 11: In clockwise order from the upper left, search results for reaching, kicking, walking, and jumping. Query motions are green and matches are blue.

three to 24.6MB. Finding matches for a 1.5s query took on average 0.024s. Since each match is used as a new query, the time needed for a full search depends on how many matches are found — for example, finding 100 matches would take about 2.4s.

Using just the names of the data files, we next divided the data set into six categories: cartwheels, fighting, reaching, locomotion, jumping/hopping, and miscellaneous. Each individual data file, however, still contained multiple actions; for example, one “reaching” file contained six reaching motions, many walk cycles, and some motion of the actor readying himself. Separate match webs were built for each category, which took a total of 3.4 minutes and consumed 45MB of disk space without compression. Searching the reaching and locomotion match webs, which each comprised about a quarter of the data, took on average 0.006s for a 1.5s query, or about a quarter of the time needed in the unlabelled case. These results may be interpreted as follows. Building a match web requires $O(n^2)$ time for a data set of n frames, and so dividing the data set into six pieces should reduce computation time by roughly an order of magnitude (see Section 5 for more discussion on scalability). On the other hand, motions in different categories tend to be dissimilar and hence have sparse match webs, so dividing up the data set produces a more modest savings in storage than in computation time. The reduced search time stems from our search algorithm being linear in the size of the data set, so smaller data sets yield proportionately faster searches.

Accuracy. To test the accuracy of the search results, we first restricted searches to data files in the same semantic category as the query. We entered queries of walking, jogging, jumping, hopping, punching, cartwheeling, sitting down/standing up, stepping onto/off of platforms, kicking, and picking up an object. For each query we attempted to find a distance threshold that would find all logically similar motion segments (identified manually) and nothing else. This was possible for all but the last two queries. For the kick query, we were able to find all kicks involving the same leg except for a spinning back kick. This kick was sufficiently different that it did not have strong time correspondences with any other kick (criterion 2 in Section 3.1). For the query of a character picking up an object from a shelf, the system returned every motion involving reaching to the shelf, but in half the cases the character was putting the object back. A human can distinguish these motions because the reaching arm is initially hanging downwards when picking the object up and bent when putting it back (see Figure 12). However, this difference is sufficiently subtle relative to the rest of the motion that our system could not discern it.

We next ran the same experiments using the original, unlabelled data set. The results were identical, with two exceptions. First, more matches were returned for the walking query, since some motions not labelled as locomotion contained short segments of walking. Second, for the reaching query we found that any distance

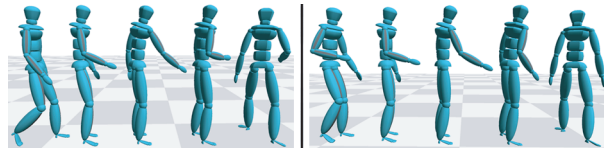


Figure 12: Due to their strong numerical similarity, our system confuses picking up an object (left) with putting it back to the same location (right).

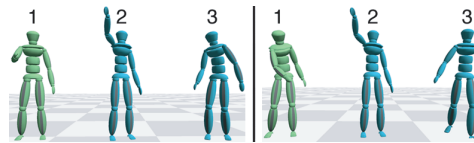


Figure 13: **Left.** Relative to a middle reach (1), a high reach’s (2) numerical distance is comparable to looking over one’s shoulder (3), but it is closer in terms of graph distance. **Right.** For a query of a lower-left reach (1), any threshold that returns an upper-right reach (2) as a first-tier match also returns spurious matches, such as walking (3).

threshold large enough to return all of the data set’s 17 reaching motions also included spurious matches. This is because some reaching motions were sufficiently different than the others that, in terms of our numerical similarity metrics, they were comparable to logically unrelated motions such as looking over one’s shoulder (Figure 13). However, when sorted in order of increasing distance to the query, the first 17 matches were the true reaching motions.

Discussion. Our search algorithm defines matches as motion segments that are either close to the query or connected to it via a sequence of close intermediary matches. This allows one to find distant matches while still using smaller, more reliable distance thresholds that prune unrelated motion segments. For example, given a query of someone reaching to the lower left, any distance threshold large enough for an upper-right reach to be considered close (i.e., a first-tier match) also produced many spurious matches, such as walking motions (Figure 13). On the other hand, using a lower threshold and multi-step search correctly identified the upper-right reach as closer to the query than any other non-reaching motion.

Nonetheless, some matches may be sufficiently far from the others that to find them one *must* use a threshold which will also return spurious matches. Since these spurious matches are used as new queries, they can lead to additional spurious matches. Two possible solutions are to incorporate semantic information, which prunes the space of candidate matches, and to sort the matches based on shortest-path distance to the query, on the assumption that spurious matches have a greater total distance than true matches. In our experiments, both of these were successful.

If one logical match is very far from the others, then it may not be possible to generate clear time correspondences with any other match (that is, the second criterion of Section 3.1 would fail). In this case, that match will not be part of the match web and cannot be found by our system. We believe this is reasonable because our ultimate goal is to blend the matches to create parameterized motions, and a match that is this different from the others is unlikely to yield successful blends.

4 Parameterizing Motion

Once example motions have been collected, they can be blended to create new motions [Wiley and Hahn 1997; Rose et al. 1998]. Several blending algorithms are in the literature; we use the one suggested by Kovar and Gleicher [2003]. While a synthesized motion can be adjusted simply by varying the blend weights, in general blend weights have no simple relationship to motion features. To

provide more intuitive control, we parameterize the space of blends according to a user-supplied parameterization function \mathbf{f} that computes relevant properties of the initial query motion \mathbf{M}_q . For the following discussion, we assume that the inputs to \mathbf{f} are joint positions and orientations. This allows a wide range of properties to serve as the basis for the parameterization, including the location of an end effector at a point in time; the average, minimum, or maximum angular velocity of a joint; or features of aggregate quantities such as the center of mass.

Abstractly, \mathbf{f} maps a set of blend weights \mathbf{w} to a parameter vector \mathbf{p} . Our goal is to invert this function: given a set of parameters, we want blend weights that produce the corresponding motion. Unfortunately, in general \mathbf{f}^{-1} has no closed form representation. Moreover, since the number of examples is almost always greater than the dimensionality of the parameter space, \mathbf{f} is a many-to-one function, and thus computing \mathbf{f}^{-1} is an ill-posed problem. In light of this, we use scattered data interpolation to construct an approximate representation of \mathbf{f}^{-1} from a set of discrete samples $(\mathbf{p}_1, \mathbf{w}_1), \dots, (\mathbf{p}_n, \mathbf{w}_n)$. The i^{th} example motion \mathbf{M}_i is associated with a sample where $\mathbf{p} = \mathbf{f}(\mathbf{M}_i)$ and \mathbf{w} has 1 in the i^{th} position and 0 everywhere else. Given a new set of parameters \mathbf{p}' , nearby parameter samples are identified, and their blend weights are averaged according to the distance to \mathbf{p}' . This procedure uses the sampled blend weights to restrict the set of possible output blend weights, making the computation of \mathbf{f}^{-1} well-posed.

The approximation to \mathbf{f}^{-1} becomes more accurate as the parameter space is sampled more densely, and in the limit the samples provide a lookup table that directly maps parameters to blend weights. Uniform sampling is also desirable because clusters of samples can skew the approximation. For instance, if the closest neighbors are all duplicates of the same sample, the approximation simply returns that sample. However, the sampling of parameter space provided by the example motions is not guaranteed to be dense or uniform, and hence the approximation of \mathbf{f}^{-1} may be inaccurate (Figure 2). To correct this, we generate blends to create additional samples of \mathbf{f}^{-1} , with the goal of ensuring that the parameter space is sampled densely and uniformly. This requires addressing the following issues:

1. **Motion registration.** Motions must be registered in time before they can be blended. We exploit the structure of the match graph to automatically register the example motions.
2. **Sampling strategy.** To scale to large numbers of examples, we sample subsets of blend weights by finding sets of example motions that are nearby in parameter space. These blend weights are constrained to reasonable values that represent either interpolations or limited extrapolations.
3. **Fast interpolation that preserves constraints.** We use a k -nearest-neighbors technique that is efficient for large example sets and respects blend weight constraints.

In the remainder of this section, we provide more details on each of these matters and conclude with some example results.

4.1 Registration

Motions are registered in time through a timewarp curve $\mathbf{s}(u)$, which is a continuous analogue of a time alignment. If there are N_e example motions, then each point on \mathbf{s} is an N_e -dimensional vector specifying a set of corresponding frame times. Each dimension of \mathbf{s} is required to be strictly increasing so that given a frame of motion, the associated point on \mathbf{s} can be uniquely identified. While early work in motion blending constructed timewarp curves manually [Rose et al. 1998], more recent work [Kovar and Gleicher 2003] has automatically determined frame correspondences

by minimizing the distance between matched frames. This is accomplished through the same dynamic programming methods that were discussed in Section 3. While this strategy works well when motions are close, it can fail for more distant motions. For example, if two reaching motions target very different locations (Figure 1), then frames at the apex of each reach are the most *dissimilar* in the two motions. The optimization will hence explicitly avoid matching these frames together.

This problem can be avoided by using “in between” motions to infer the timing relationship between distant motions. We start by using Dijkstra’s algorithm to identify the shortest path from the query \mathbf{M}_q to every other motion in the match graph. Any edge that is not on one of these shortest paths is discarded. For each remaining edge, we use dynamic programming to calculate a new time alignment for the nodes it connects. This is reasonable since, by construction, edges on the match graph only connect close motions with “obvious” time correspondences (Section 3.1). We next fit an endpoint interpolating, strictly increasing spline to the time alignment to generate a proper timewarp curve [Kovar and Gleicher 2003]. Given a frame of \mathbf{M}_q , we can now find the corresponding frame in any other motion by walking down the shortest path, using the timewarp curve at each edge to convert the frame of the current motion to the corresponding frame of the next motion. In this manner for any frame $\mathbf{M}_q(t_q)$ we can generate a frame correspondence $(\mathbf{M}_q(t_q), \mathbf{M}_1(t_1), \dots, \mathbf{M}_{N_e-1}(t_{N_e-1}))$. Finally, to generate a timewarp curve for the entire match graph, we sample \mathbf{M}_q to generate a dense set of these frame correspondences and fit an N_e -dimensional, strictly increasing, endpoint interpolating spline.

If any part of the parameterization function involves the skeletal configuration on a specific frame $\mathbf{M}_q(t_0)$, then this frame index t_0 is converted to the corresponding index u_0 on the timewarp curve. This allows the parameterization function to be computed for any other example motion and for any blend of the examples.

4.2 Sampling

While our goal is to produce a dense sampling of parameter space, we only have direct control over the blend weights. A simple strategy is hence to indirectly sample the parameter space by densely sampling the blend weights. However, this is infeasible for large example sets because the dimensionality of the blend weight space is proportional to the number of examples, and so the number of samples needed to achieve a given sampling density grows exponentially with the number of examples. Moreover, such a sampling would be redundant because \mathbf{f} is a many-to-one function — the same region of parameter space would be covered repeatedly by different sets of blend weights.

These difficulties can be avoided by limiting blends to subsets of examples that are nearby in parameter space. Intuitively, the goal of sampling is to fill in the gaps in parameter space, and the most natural strategy for filling any given gap is to combine the closest example motions. For example, imagine the examples are various reaching motions and we want to sample blends that reach near chest height. This could theoretically be done by combining motions that reach to the ground with ones that involve standing on tiptoes, but it is more sensible to instead combine example motions that reach within a similar region.

We can turn this intuition into an algorithm as follows. First, we compute the parameters of each example motion. To approximate the accessible region of parameter space, we compute a bounding box and expand each dimension by a fixed percentage about the central value (20% in our implementation). We then randomly sample points in this region and for each find the $d + 1$ example motions with the closest parameters, where d is the dimensionality of the parameter space. We use this number of neighbors because it is the minimum necessary to form a volume in parameter space. The weight of every other motion is set to zero, and a random set

of weights is generated for the neighbors under the restriction that these weights be nearly convex, or *valid*. Specifically, we require

$$-\delta \leq w_i \leq 1 + \delta, \quad \sum_i w_i = 1, \quad (1)$$

where δ controls the allowable degree of extrapolation. In particular, when $\delta = 0$ only interpolation is allowed. A random set of valid blend weights can be calculated as follows. Let S be the sum of all blend weights that have been assigned values; initially $S = 0$. While unassigned weights exists, randomly select one of these weights w . If w is the last unassigned weight, set it to $1 - S$. Otherwise randomly assign it a value from the interval $[\max(-\delta, -\delta - S), \min(1 + \delta, 1 + \delta - S)]$. This ensures that both w and $S + w$ have a value in the range $[-\delta, 1 + \delta]$. The latter condition is important because it guarantees that the final weight is valid.

To prevent parameter samples from being too close, for each new sample the closest existing sample is found and the former is discarded if the distance is below a threshold. In our experiments, which focused on joint positions, this threshold was half an inch.

4.3 Interpolation

Given a new set of parameters $\tilde{\mathbf{p}}$, we use k -nearest-neighbors interpolation to find blend weights $\tilde{\mathbf{w}}$ that produce those parameters. Let the k nearest neighbors be $\mathbf{p}_1, \dots, \mathbf{p}_k$, in order of increasing distance, and let \mathbf{w}_i be the blend weights associated with \mathbf{p}_i . $\tilde{\mathbf{w}}$ is approximated as

$$\tilde{\mathbf{w}} = \sum_{i=1}^k \alpha_i \mathbf{w}_i. \quad (2)$$

Following Allen et al [2002], each α_i is initially assigned the value

$$\alpha_i = \frac{1}{D(\tilde{\mathbf{p}}, \mathbf{p}_i)} - \frac{1}{D(\tilde{\mathbf{p}}, \mathbf{p}_k)}, \quad (3)$$

where D computes the distance between two parameters (Euclidean distance in our implementation). These weights are then normalized so they sum to 1. Since the α_i are nonnegative and sum to 1, $\tilde{\mathbf{w}}$ is inside the convex hull of the \mathbf{w}_i , and it is straightforward to show that $\tilde{\mathbf{w}}$ therefore satisfies the conditions in (1). This ensures that any parameters specified by the user will produce a motion within the space of valid blends. In particular, parameters that are not attainable are projected onto the accessible region of parameter space.

As a result of our sampling procedure, at most $d + 1$ elements of each \mathbf{w}_i are nonzero. This implies that $\tilde{\mathbf{w}}$ will in the worst case have $k(d + 1)$ nonzero weights, and in practice there are fewer because nearby parameter samples tend to have the same set of nonzero weights. Since motions with zero weight can be ignored in the blending calculation, the asymptotic run time of our algorithm is independent of the number of examples. This analysis neglects the cost of finding the k nearest neighbors, but we have found that even a brute force nearest neighbor calculation is negligible relative to the cost of computing a blend.

4.4 Results and Applications

We have implemented the above algorithms and used them to create a variety of parameterized motions; see Table 1 for a summary. In each case we generated a thousand parameter samples using the method described in Section 4.2. The time needed to generate these samples varied from 1.7s to 6.7s, and after eliminating redundant samples the storage cost was 8.2% of the example motion data in the worst case and 3.2% on average. In all of our experiments new motions could be synthesized in real time, and these motions matched the user’s target parameters to within visual tolerance as long as they were within the accessible region of parameter space. We set $k = 12$ when performing nearest neighbor interpolation.

| Motion | # Examples | Parameterization |
|-----------|------------|-------------------------------|
| reach | 6 | apex of reach |
| walk | 96 | final root position on ground |
| kick | 4 | target location |
| sit | 2 | lowest height of hips |
| step up | 4 | heel height when on platform |
| punch | 7 | target location |
| hop | 4 | final root position on ground |
| cartwheel | 11 | final root position on ground |

Table 1: Parameterized motions built in our experiments.

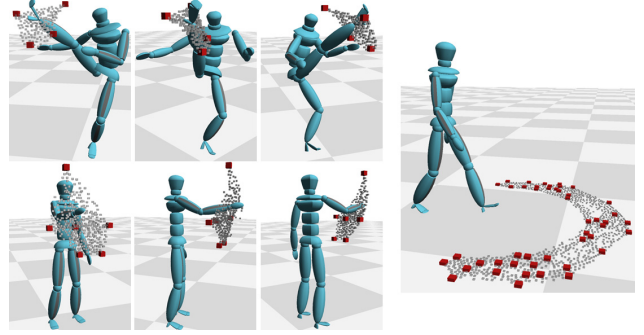


Figure 14: Visualization of the accessible locations for the ankle of a kick (upper left), wrist of a reach (lower left), and final position of a walk cycle (right). Large red cubes show parameters of example motions; small grey cubes are sampled parameters.

In addition to providing more accurate parameterizations, the sampled parameters can be used to visualize the range of synthesizable motions. Figure 14 shows some cases where this can be accomplished simply by drawing markers at the location of each parameter sample. Another application of our methods is automatic removal of redundant example motions, which can reduce the parameterized motion’s memory footprint. For each example motion, we compute its parameters and see if they can be reproduced within a user-specified tolerance by interpolating nearby examples. If so, it is discarded. For our parameterized walk, we removed all example motions where the final root location could be reproduced to within a quarter inch, reducing the number of motions from 96 to 46. The parameterized walk also shows the scalability of our scattered data interpolation method: including all 46 example motions in a blend takes an order of magnitude longer than using our algorithm.

The automation provided by our system makes it feasible to experiment with unusual parameterized motions. Starting with 34s of cartwheel data, we built a parameterized motion where the user could control the final position of a sequence of cartwheels. The total amount of time needed to build the match web, identify a particular cartwheel, execute a search for other cartwheels, and generate parameter samples was less than thirty seconds.

5 Discussion

This paper has presented automated methods for extracting logically related motions from a data set and converting them into an intuitively parameterized space of motions. One contribution of this work is a novel search method that uses numerically similar matches as intermediaries to find more distant matches, together with a precomputed representation of all possibly similar motion segments that makes this approach efficient. A second contribution is an automatic procedure for parameterizing a space of blends according to user-specified motion features. This algorithm samples blends to build an accurate approximation of the map from motion parameters to blend weights, and it uses a scalable scattered data

interpolation method that preserves constraints on blend weights.

We conclude with a brief discussion of the scalability and generality of our methods.

Scalability. For our test data set, which we believe is sizeable relative to what is used in current research, the time and space costs for building and storing a match web were quite manageable. However, a match web for a data set of n frames takes $O(n^2)$ time to construct and $O(n^2)$ space to store. The time costs can be mitigated by using a multi-resolution construction method (as discussed in Section 3.2) and by computing match webs for different pairs of motions in parallel, and storage can be reduced through compression algorithms. Nonetheless, for massive databases it will not be feasible to build match webs for every pair of motions. A simple solution is to partition the database into independent modules based on semantic content and compute match webs separately for these modules. Since a very large database will contain many unrelated motions, such a division would be natural and is likely to already be reflected in the organization of the data files. The development of alternatives to match webs that are asymptotically more efficient yet have similar performance characteristics is left for future work.

While our parameterization algorithms apply in theory to parameter spaces of arbitrary dimensionality, in practice they are limited by the quantity of available data. Roughly speaking, we need at least enough examples to cover every combination of minimum/maximum values for individual parameters (the “corners” of the space), and more examples can provide higher quality results. The number of necessary example motions is hence exponential in the number of parameters. Developing methods to ease the data requirements while preserving motion quality is left for future work.

Generality. While we have focused on parameterization functions involving joint positions and orientations, our methods allow parameterizations based on abstract properties like mood. For qualitative features like these where accuracy is less meaningful, we can simply skip the sampling step of Section 4.2 and apply scattered data interpolation directly to the example motions.

Motion sets found by our search engine are not guaranteed to be blendable. For example, one of our query motions consisted of a character stepping toward a shelf, picking up an object, and walking away. While our system correctly identified the eight other pick-up actions in the database, in three of these the initial step was with the wrong foot, and so these motions had to be discarded. Similarly, we tried to construct a parameterized leaping motion but found the blended body trajectory to be physically implausible. More generally, the only reliable way of determining whether motions can be successfully blended is to create and look at specific blends. In light of this, one of the primary advantages of our system is that it greatly speeds and simplifies the process of experimentation.

Acknowledgements: This work was made possible through motion data donations from House of Moves and Demian Gordon and financial support from NSF grants CCR-9984506 and CCR-0204372. Lucas Kovar is supported by a Cisco Systems Distinguished Graduate Fellowship.

References

- AGRAWAL, R., FALOUTSO, C., AND SWAMI, A. 1993. Efficient similarity search in sequence databases. In *Proceedings of the 4th International Conference on Foundations of Data Organizations and Algorithms (FODO)*, Springer Verlag, 69–84.
- ALLEN, B., CURLISS, B., AND POPOVIĆ, Z. 2002. Articulated body deformation from range scan data. *ACM Transactions on Graphics* 21, 3, 612–619.
- ARIKAN, O., AND FORSYTHE, D. A. 2002. Interactive motion generation from examples. *ACM Transactions on Graphics* 21, 3, 483–490.
- ARIKAN, O., FORSYTH, D. A., AND O'BRIEN, J. 2003. Motion synthesis from annotations. *ACM Transactions on Graphics* 22, 3, 402–408.
- BÖHM, C., BERCHTOLD, S., AND KEIM, D. A. 2001. Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. *ACM Computing Surveys* 33, 3, 322–373.
- BRUDERLIN, A., AND WILLIAMS, L. 1995. Motion signal processing. In *Proceedings of ACM SIGGRAPH 1995*, Annual Conference Series, 97–104.
- CARDLE, M., VLACHOS, M., BROOKS, S., KEOGH, E., AND GUNOPULOS, D. 2003. Fast motion capture matching with replicated motion editing. In *Proceedings of SIGGRAPH 2003 Technical Sketches & Applications*.
- CASTELLI, V., AND BERGMAN, L. 2001. *Image Databases: Search and Retrieval of Digital Imagery*. John Wiley & Sons.
- CHAN, K., AND FU, W. 1999. Efficient time series matching by wavelets. In *Proceedings of the 15th IEEE International Conference on Data Engineering*, 126–133.
- FALOUTSOS, C., RANGANATHAN, M., AND MANOLOPOULOS, Y. 1994. Fast subsequence matching in time-series databases. In *Proceedings of 1994 ACM SIGMOD International Conference on Management of Data*, 419–429.
- FUNKHOUSER, T., MIN, P., KAZHDAN, M., CHEN, J., HALDERMAN, A., AND DOBKIN, D. 2003. A search engine for 3d models. *ACM Transactions on Graphics* 22, 1, 83–105.
- GROCHOW, K., MARTIN, S., HERTZMANN, A., AND POPOVIĆ, Z. 2004. Style-based inverse kinematics. *ACM Transactions on Graphics* 23, 3.
- GUTTMAN, A. 1984. R-trees: a dynamic index structure for spatial searching. In *Proceedings of 1984 ACM SIGMOD International Conference on Management of Data*, 47–57.
- JENKINS, O. C., AND MATARIĆ, M. 2002. Deriving action and behavior primitives from human motion data. In *Proceedings of 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-2002)*, 2551–2556.
- KEOGH, E., CHAKRABARTI, K., PAZZANI, M., AND MEHROTRA. 2001. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proceedings of 2001 ACM SIGMOD International Conference on Management of Data*, 151–162.
- KIM, T., PARK, S., AND SHIN, S. 2003. Rhythmic-motion synthesis base on motion-beat analysis. *ACM Transactions on Graphics* 22, 3, 392–401.
- KOVAR, L., AND GLEICHER, M. 2003. Flexible automatic motion blending with registration curves. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2003*.
- KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. *ACM Transactions on Graphics* 21, 3, 473–482.
- LEE, J., CHAI, J., REITSMA, P., HODGINS, J., AND POLLARD, N. 2002. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics* 21, 3, 491–500.
- LIU, F., ZHUAN, Y., WU, F., AND PAN, Y. 2003. 3d motion retrieval with motion index tree. *Computer Vision and Image Understanding* 92, 2-3, 265–284.
- PARK, S. I., SHIN, H. J., AND SHIN, S. Y. 2002. On-line locomotion generation based on motion blending. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2002*.
- ROSE, C., COHEN, M., AND BODENHEIMER, B. 1998. Verbs and adverbs: multidimensional motion interpolation. *IEEE Computer Graphics and Application* 18, 5, 32–40.
- ROSE, C., SLOAN, P., AND COHEN, M. 2001. Artist-directed inverse-kinematics using radial basis function interpolation. *Computer Graphics Forum* 20, 3.
- ROWEIS, S. T., AND SAUL, L. K. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* 290, 5500, 2323–2326.
- TENENBAUM, J. B., DE SILVA, V., AND LANGFORD, J. C. 2000. A global geometric framework for nonlinear dimensionality reduction. *Science* 290, 5500, 2319–2323.
- UNUMA, M., ANJYO, K., AND TEKEUCHI, T. 1995. Fourier principles for emotion-based human figure animation. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, 91–96.
- VELTKAMP, R. C., BURKHARDT, H., AND KRIEGEL, H. P. 2001. *State-of-the-Art in Content-Based Image and Video Retrieval*. Kluwer Academic Publishers.
- VLACHOS, M., HADJIELEFTHERIOU, M., GUNOPULOS, D., AND KEOGH, E. 2003. Indexing multi-dimensional time-series with support for multiple distance measures. In *9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 216–225.
- WANG, J., AND BODENHEIMER, B. 2003. An evaluation of a cost metric for selecting transitions between motion segments. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2003*.
- WILEY, D., AND HAHN, J. 1997. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Application* 17, 6, 39–45.
- ZORDAN, V., AND HODGINS, J. 2002. Motion capture-driven simulations that hit and react. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2002*.