Parametric Motion Graphs

Rachel Heck and Michael Gleicher*
University of Wisconsin-Madison

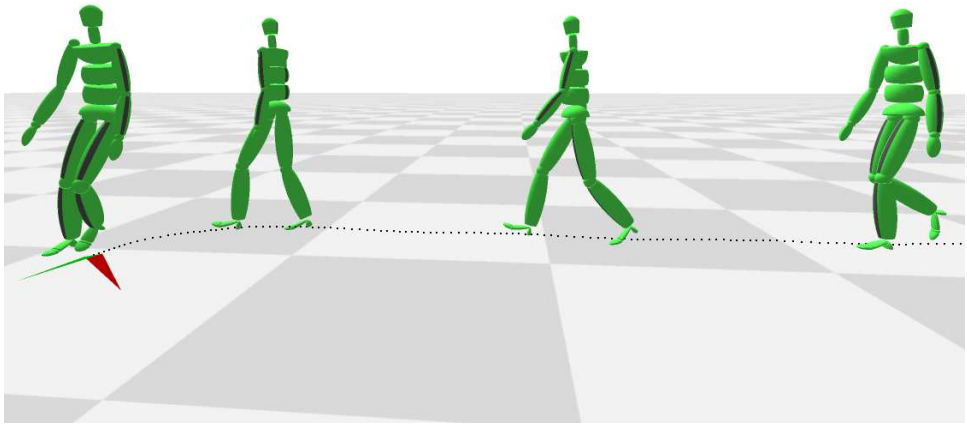

Figure 1: An interactively controllable walking character using parametric motion graphs to smoothly move through an environment. The character is turning around to walk in the user-requested travel direction, depicted by the red arrow on the ground.

Abstract

In this paper, we present an example-based motion synthesis technique that generates continuous streams of high-fidelity, controllable motion for interactive applications, such as video games. Our method uses a new data structure called a *parametric motion graph* to describe valid ways of generating linear blend transitions between motion clips dynamically generated through *parametric synthesis* in realtime. Our system specifically uses blending-based parametric synthesis to accurately generate any motion clip from an entire *space* of motions by blending together examples from that space. The key to our technique is using sampling methods to identify and represent good transitions between these spaces of motion parameterized by a continuously valued parameter. This approach allows parametric motion graphs to be constructed with little user effort. Because parametric motion graphs organize all motions of a particular type, such as reaching to different locations on a shelf, using a single, parameterized graph node, they are highly structured, facilitating fast decision-making for interactive character control. We have successfully created interactive characters that perform sequences of requested actions, such as cartwheeling or punching.

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

Keywords: motion capture, motion synthesis, motion graphs

1 Introduction

In many interactive applications, such as video games and simulations, humanoid characters play an essential role. One important aspect of these characters is the way they move. These movements must not only be of sufficient fidelity but must also respond to user control and dynamically changing environments. Ideally, any motion synthesis method used in an interactive application should efficiently produce continuous streams of high-fidelity motions; be responsive to changing inputs; generate motions that accurately meet supplied constraints, such as the location where a character should punch; and allow easy authoring of new movements.

Computer animation researchers and practitioners have provided a number of methods for generating character motions. However, existing approaches make limiting tradeoffs between motion quality, accuracy, responsiveness, and ease of authoring. Methods used in practice for creating the motions in video games require extensive work to author the structures used for motion control, and often the results are still limited in their movement quality and/or control accuracy. Alternatively, methods developed by animation researchers provide automated authoring of high-fidelity motions, but these methods fail to simultaneously provide the accurate control, flexibility in movement types, and responsiveness demanded by interactive applications. Our goal is to provide a motion synthesis technique that produces accurate, controllable, high-fidelity motion streams and allows automated authoring of interactive characters.

In this paper, we introduce the *parametric motion graph*, an example-based motion synthesis data structure. Like other example-based data structures, parametric motion graphs provide easy authoring of high-quality motions but also supply the responsiveness, precise control, and flexibility demanded by interactive applications. A parametric motion graph describes possible ways to generate seamless streams of motion by concatenating short motion clips generated through blending-based parametric synthesis. Blending-based parametric synthesis allows accurate generation of any motion from an entire space of motions, by blending together examples from that space. For example, parametric synthesis can generate motions of a person picking up an item from any location on a shelf by blending together a small set of example motions. While neither seamless motion concatenation nor parametric synthesis is a new idea, by combining both techniques, parametric motion graphs can provide accurate control through parametric synthesis and can generate long sequences of high-fidelity motion without visible seams using linear blend transitions.

In contrast to many other automated methods for representing transitions between motions, parametric motion graphs are highly

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

structured, facilitating efficient interactive character control. The nodes of a parametric motion graph represent entire parametric motion spaces that produce short motions for given values of their continuously valued parameters. The directed edges of the graph encode valid transitions between source and destination parameterized motion spaces. This structure efficiently organizes the large number of example motions that can be blended together to produce the final motion streams. Because of this structure, we have been able to easily author interactively controllable characters that can walk, run, cartwheel, punch, change facing direction, and/or duck in response to user-issued requests.

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

The rest of this paper is organized as follows. Section 1.1 provides an overview of our technique. Then, Section 2 discusses other work related to interactive motion generation. In Sections 3 and 4, we detail our method for building and extracting information from a parametric motion graph. Then, Section 5 presents the results of our work, describing some of the parametric motion graphs we built and the applications we made to show their utility. Finally, Section 6 concludes with a general discussion of the presented technique, including a number of the technique's limitations.

1.1 An Overview of Our Technique

To provide parametric motion graphs as a method for interactive character control, we describe how to:

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

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

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

To illustrate the utility of parametric motion graphs, we give a concrete example. We can create a character that can be directed through an environment with continuous steering controls. Using an existing blending-based parametric synthesis method, we first build a parametric motion space of a person walking at different curvatures for two steps. Next, we quickly build a parametric motion graph from this motion space using the algorithm presented in this paper. The resulting graph contains a single node, representing the parameterized walking motion space, and a single edge that starts and ends at this node. This edge describes how to transition from the end of a generated walking clip to any generated walking clip in a subspace of the motion space. This simple structure organizes many motions in a way that allows efficient character control at runtime. By translating a user's desired travel direction into desired curvature requests, we can synthesize a continuous stream of walking motion that reacts to a user's commands. This stream of motion will be smooth, run at interactive rates, and will only contain high-fidelity transitions between clips of motion. See Figure 1 to see a screen capture of our interactive walking character. Unlike other techniques that can create interactive walking characters, our technique requires little authoring effort, is capable of accurate motion generation, and works with a wide range of different motions. For instance, once we have an interactive walking character, it is easy to create a character that locomotes by running or cartwheeling simply by building a parameterized motion space of running or cartwheeling motions, also parameterized on curvature.

2 Related Work

Researchers have studied ways to generate human motion in an automated way; two of these approaches serve as the foundation of our work. The first, parametric synthesis, is the set of techniques that map motion parameters to motion, allowing the generation of any motion from an entire space of motions by supplying the relevant parameters. Previous work on parametric synthesis can be divided into two groups: *procedural* and *blending-based*. Procedural parametric synthesis generates very specific parametric motion spaces using highly-specialized algorithms [Perlin 1995; Perlin and Goldberg 1996; Hodgins et al. 1995]. Blending-based parametric synthesis builds parameterized spaces of motion in a general way using motion interpolation [Bruderlin and Williams 1995] on a set of examples from that space [Wiley and Hahn 1997; Rose et al. 1998; Kovar and Gleicher 2004; Mukai and Kuriyama 2005]. For example, blending-based parametric synthesis can generate motions of a person punching toward any location within an enclosed area by analyzing and blending example punching motions from that space. A parametric motion graph uses blending-based parametric synthesis to generate clips of motion that accurately meet user-specified constraints, allowing us to represent an infinite number of motions in a simple compact structure. What this previous work on parametric synthesis does not provide is a way to transition between these different parameterized spaces of motions.

The second motion generation approach that our work builds upon is synthesis-by-concatenation. This approach generates long motion streams by piecing together many short motion clips. Early work focused on realistic ways to transition between two motion clips [Rose et al. 1996; Lamouret and van de Panne 1996]. More recent work allows possible transitions in a motion collection to be represented using graph structures [Arikan and Forsythe 2002; Kovar et al. 2002; Lee et al. 2002; Arikan et al. 2003; Kim et al. 2003]. Like with Video Textures [Schödl et al. 2000], these techniques focus on using automated comparison methods to locate frames of motion that look similar enough to be used as a transition point, allowing easy authoring of the transition graphs. Like this previous work, we are interested in finding and representing possible transitions between motions. But because parametric motion graphs use parametric synthesis to generate accurate motion clips, they must represent possible transitions between *spaces* of motion.

While the motion graphs described above are capable of producing natural motion, they are not structured, thus requiring a costly global search in order to locate motion sequences that meet specified constraints. This dependence on global search makes it difficult to use these structures for interactive character control. Other

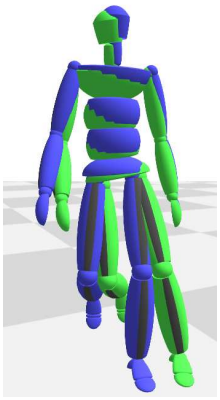

Figure 2: At the transition point between two motions of a character turning towards the right, our character remains leaning into the turn (as shown in green) while the character using a fat graph must return to the common transition pose with no lean (as shown in blue), causing the character to "bob" as it goes around the turn.

researchers have augmented these graphs by precomputing graph properties that aid in control. These techniques include the *mobility maps* of Srinivasan et al. [2005] and the method of Lee and Lee [2004]. Unlike parametric motion graphs, these methods are only able to represent transitions between a discrete number of motions and become unwieldy as the number of motions becomes large. Furthermore, these techniques do not directly address the problem that the underlying graph is unstructured; they instead deal with the unstructured graph in a more efficient way.

The gaming industry often uses hand-generated, structured graphs called move trees to represent possible transitions between motion clips [Mizuguchi et al. 2001]. Because move trees are constructed for easy interactive character control, they have a deliberate, hand-designed structure that aids in choosing motions based on user requests. Because parametric motion graphs use parameterized motion spaces to represent entire motion families, they also offer a way to deliberately manage the complexity of the transitions between many motions. This structure facilitates the use of parametric motion graphs for controlling characters in realtime.

Other researchers have built structured representations for online locomotion generation. Sun and Metaxas [2001] use a procedural parametric walking motion to generate streams of motion that adjust to uneven terrain and user-defined curvature. Using a similar method, Park et al. [2002] generate locomotion, such as walking and jogging, whose curvature is controllable. Kwon et al. [2005] group motion segments based on footstep patterns and generate transitions between these groups using a hierarchical motion graph, where the coarsest level describes general transition patterns and the more detailed levels capture the cyclic nature of locomotion. These techniques for generating controllable locomotion show the utility of using structure to produce controllable motion in realtime. Our technique uses structure to control more general motions with a wide variety of different motion properties.

One structured-graph technique that shares many of our goals is Snap-Together Motion [Gleicher et al. 2003]. The key of this technique is to identify poses that appear many times in the original motion examples. These poses become "hub" nodes in a graph, and edges between these nodes correspond to motion clips that can transition between these poses. A recent extension to this work groups similar clips that connect the same "hub" nodes into parametric edges, forming a new structure called a fat graph [Shin and Oh 2006]. Fat graphs and parametric motion graphs are very similar in that they combine parametric synthesis and synthesis-by-concatenation to provide interactive control. But parametric motion graphs have a number of other benefits. Because all motions representing the same logical action, such as walking or dodging, are always grouped together, parametric motion graphs provide even more structure and a graph author can easily see the logical connections between motion types. Parametric motion graphs also represent continuously changing transition points and ranges within a single type of motion. A fat graph must use more than one "hub" node in order to capture a portion of the complexity of these shifting transition possibilities. Fat graphs are also limited in the quality of their results by the use of "hub" nodes; motions are constantly forced to return to a common pose at each transition point, as illustrated in Figure 2. On the other hand, parametric motion graphs handle natural variations in the transition poses of similar motions.

3 Building a Parametric Motion Graph

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

As described in Section 1, each node of a parameterized motion graph represents a parametric motion space implemented using blending-based parametric synthesis. For all of our examples, we perform blending-based parametric synthesis using the techniques of Kovar and Gleicher [2004]. Kovar and Gleicher describe how to automatically find and extract logically similar motions, or motions where the character is performing the same basic action, from a motion database; they then provide a method for building and representing parameterized spaces from these motions using blending techniques. We chose to use this method because it produces high-quality results, allows for quick experimentation with many different types of motion, provides a simple and efficient method for producing motion clips at runtime, and results in parameterized motion spaces that are *smooth*. A parameterized motion space is considered smooth if small changes in the input parameters produce small changes in the generated motion.

While we can build the nodes of a parametric motion graph using this existing technique, our key challenge is finding a way to identify and represent possible transitions between these parameterized nodes. The smoothness property of the motion spaces represented by our graph nodes allows us to tackle this challenge using sampling. The rest of this section describes in detail how to identify and represent edges between source and target graph nodes, $\mathbf{N}_s$ and $\mathbf{N}_t$ respectively. Throughout this description, we define a motion as the continuous function $\mathbf{M}(t)$, which provides values for each of the degrees of freedom of a hierarchical skeleton model at time t. In practice, $\mathbf{M}(t)$ is represented as regularly sampled frames, $\mathbf{M}(t_1),\ldots,\mathbf{M}(t_n)$, and values for times not sampled are produced using linear interpolation on the degrees of freedom. The parameterized motion space represented by node $\mathbf{N}_i$ is denoted by $\mathscr{P}^i(l)$, where l is a vector of relevant motion parameters, such as the target of a punch; a parametric motion space produces a short motion, $\mathbf{M}_i$, for any given value, l_i, of its continuously valued parameters.

3.1 Identifying Transitions Between Motion Spaces

To start, consider the case where the nodes $\mathbf{N}_s$ and $\mathbf{N}_t$ represent small motion spaces whose valid parameter ranges only include a single point. This case reduces to the traditional synthesis-by-concatenation problem; can we locate a frame of motion near the end of the motion generated by $\mathbf{N}_s$, $\mathbf{M}_1$, and a frame of motion near the beginning of the motion generated by $\mathbf{N}_t$, $\mathbf{M}_2$ that are similar enough to allow a linear blend transition from one to the other over a short window centered at these frames? To compute the similarity between two frames of motion, $\mathbf{D}(\mathbf{M}_1(t_i),\mathbf{M}_2(t_j))$, we use a

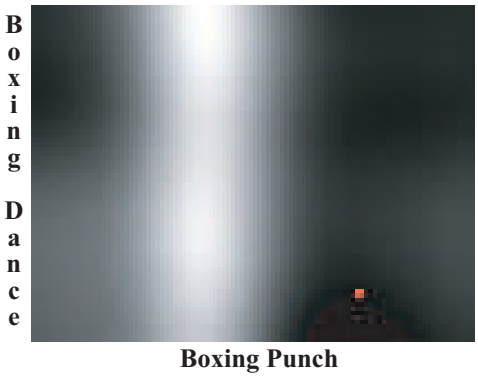

Figure 3: A distance grid. Darker regions denote greater similarity between frames of a boxing dance motion and frames of a punching motion. The light red dot marks the optimal transition point.

distance metric originally introduced by Kovar et al. [2002]. First, for both frames, we form a point cloud based on the locations of each skeletal joint over a small window of time surrounding the frame. We then compute the optimal sum-of-squared-distances between corresponding points in the two point clouds, given that each cloud may be translated along the floor plane and rotated about the vertical axis. Depending on the length of the window over which the point cloud is built, this distance metric can take into account relative joint positions, joint velocities, and joint accelerations when measuring similarity. Refer to [Kovar et al. 2002] for a closed form solution to the optimization.

Given the point cloud similarity metric, we can conclude that a good transition exists from $\mathbf{M}_1$ to $\mathbf{M}_2$ if and only if there exists a frame, t_1, near the end of $\mathbf{M}_1$ and a frame, t_2, near the beginning of $\mathbf{M}_2$ such that $\mathbf{D}(\mathbf{M}_1(t_1),\mathbf{M}_2(t_2)) \leq T_{GOOD}$, where T_{GOOD} is a tunable threshold. We first calculate the distance between every pair of frames in the possible transition regions, forming a grid. The pair of frames corresponding to the grid cell with the minimum distance value is called the optimal transition point. If the distance value of the optimal transition point is below T_{GOOD}, then it is possible to transition between $\mathbf{M}_1$ and $\mathbf{M}_2$ at that point, (t_o^1, t_o^2), by aligning the motions using the optimal translation and rotation computed for the computation $\mathbf{D}(\mathbf{M}_1(t_o^1),\mathbf{M}_2(t_o^2))$. Figure 3 shows an example of this distance grid computation between two motions.

Now consider the general case where $\mathbf{N}_s$ and $\mathbf{N}_t$ represent larger spaces. For any sufficiently large space, it is unlikely that the motions represented by the space look similar enough to be treated like a single motion. For example, in the walking example discussed in Section 1, the walking character can only transition to other walking motions where the character walks at a similar curvature to its current one. However, since each parameterized motion space represents an infinite number of motions, it is infeasible to compare all possible pairs of motions represented by each of the parameterized nodes. One possible approach is to reduce each parameterized motion space to a discrete number of motions chosen from the full space. To find and represent good transitions between all pairs of motions from a source set of size *m* and a target set of size *n*, we would need to repeat the technique described above *mn* times. Unfortunately, by transforming a continuous motion space into a discrete set of motions, we lose much of the accuracy that parametric synthesis provides us; accuracy can be increased by adding more motions to these sets but this results in a combinatorial explosion in the number of required comparisons and the amount of space needed to store the possible transitions.

Yet, we observe that in a smooth parameterized motion space, motions generated for any local neighborhood of parameter space look similar. For example, consider a parameterized motion space representing motions of a person punching, parameterized on the location of the punch. Two motions in this space where the punches land $1mm$ apart look similar. In this case we can compute the possible transitions from one of these motions and use the result for both. This observation leads us to approach the problem of identifying and representing transitions between parameterized motion spaces using sampling, extending the method for locating possible linear blend transitions between individual motions.

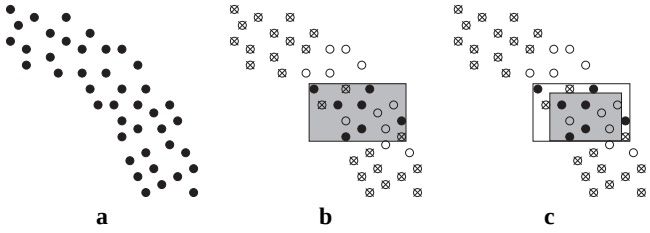

Figure 4: Process for determining the valid transition region in target parameter space for a particular motion. (a) A set of randomly chosen samples from the target space. (b) Darkened circles produce good transitions, crossed out circles produce bad transitions, and empty circles produce neutral transitions. The shaded box encloses all good samples but also includes some bad samples. (c) The adjusted shaded box excludes all bad samples. In practice, usually little to no adjustment is made to the bounding box.

3.2 Building a Parametric Motion Graph Edge

An edge between source and target nodes, $\mathbf{N}_s$ and $\mathbf{N}_t$ respectively, maps any point, l_i^s, in $\mathscr{P}^s$ to the subspace of $\mathscr{P}^t$ that can be transitioned to from $\mathbf{M}_i^s = \mathscr{P}^s(l_i^s)$. It also supplies the time at which that transition should occur. Assuming it is possible to transition from every point in $\mathbf{N}_s$ to some subspace in $\mathbf{N}_t$, we can build an edge between these nodes using sampling. We start by generating two lists of random parameter samples, $\mathbf{L}^s = \{l_1^s, \ldots, l_{n_s}^s\}$ and $\mathbf{L}^t = \{l_1^t, \ldots, l_{n_t}^t\}$ (see Figure 4a). In order to accurately capture the variations in the target space, n_t should be large. The exact number depends heavily on the size of the parameter space, but we found 1000 samples to be more than enough for all of the cases we tried, even for parameterized motion spaces that have three parameters. In contrast, n_s should be small, while still covering the source space, as this number affects the amount of storage needed for an edge. For our examples, n_s was around 50.

Now consider a sample from $\mathbf{L}^s$, l_1^s. This sample corresponds to the motion $\mathbf{M}_1^s = \mathscr{P}^s(l_1^s)$. We can determine if $\mathbf{M}_1^s$ can transition to each motion represented by the parameter samples in $\mathbf{L}^t$ by computing the optimal transition point with each motion $\{\mathbf{M}_1^t, \ldots, \mathbf{M}_{n_t}^t\}$. Samples from $\mathbf{L}^t$ that produce good transitions are added to the list of parameter samples $\mathbf{L}_{GOOD}^t$. Using the observation that motions close in parameter space look similar, we can assume that any parameter vector for $\mathscr{P}^t$ whose nearest parameter samples from $\mathbf{L}^t$ appear in $\mathbf{L}_{GOOD}^t$ can also be transitioned to from $\mathbf{M}_1^s$. Thus, the list $\mathbf{L}_{GOOD}^t$ defines the subspace of $\mathscr{P}^t$ to which $\mathbf{M}_1^s$ can transition.

Unfortunately, we cannot represent the subspace of $\mathbf{N}^t$ that can be transitioned to from $\mathbf{M}_1^s$ by listing the points in $\mathbf{L}_{GOOD}^t$ because, as described in Section 1.1, we plan to determine what transitions are possible at runtime using a simple and efficient interpolation scheme (as shown in Figure 5); interpolating between potentially different numbers of uncorrelated points in a meaningful way is difficult, if not impossible. So, instead, we represent each subspace as a simple shape that can always be interpolated (i.e. bounding boxes, spheres, triangles). We have found axis-aligned bounding boxes work well for our data; we use axis-aligned bounding boxes to represent all of the transition parameter subspaces.

Using simple, easily interpolated shapes to represent transition regions introduces a considerable problem. Any simple shape that contains all points in $\mathbf{L}^t_{GOOD}$ could also contain other points from $\mathbf{L}^t$ that were not deemed good transition candidates (see Figure 4b). To guarantee that bad transitions are not included in the transition subspace of $\mathbf{N}^t$, we take a conservative, double threshold approach. First, while constructing the list $\mathbf{L}^t_{GOOD}$, we also form a list, $\mathbf{L}^t_{BAD}$, containing all samples from $\mathbf{L}^t$ that generate motions whose optimal transition point distance is greater than T_{BAD}, where $T_{BAD} \geq T_{GOOD}$. Next, we compute the bounding box of all parameter samples in $\mathbf{L}^t_{GOOD}$. Finally, we consider each sample in $\mathbf{L}^t_{BAD}$; if the sample falls within the subspace defined by the bounding box, we make the minimal adjustment to the dimensions of the bounding box such that the sample falls at least ε away, where $\varepsilon > 0$. In this way, we construct a bounding box that contains many, if not all, of the samples from $\mathbf{L}^t_{GOOD}$ without including any of the samples from L^t_{BAD}. Neutral samples from $\mathbf{L}^t$ whose optimal transition point distance falls between T_{GOOD} and T_{BAD} are considered good enough if they fall within the transition subspace of $\mathbf{N}^t$ but will not be explicitly included in the space (see Figure 4c). In practice, the system makes very few bounding box adjustments to remove bad samples and in most cases makes none at all.

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

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

- The Value of the Parameter Vector l^s_i
- The Computed Transition Bounding Box for l^s_i
- The Average, Normalized Transition Point for l^s_i

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

Up until this point, we have assumed that we can transition from every point in $\mathbf{N}_s$ to some subspace of $\mathbf{N}_t$. We define that a transition exists between nodes $\mathbf{N}_s$ and $\mathbf{N}_t$ if and only if for any motion contained in $\mathbf{N}_s$ there exists *some* motion in $\mathbf{N}_t$ that it can transition to. Thus, if we find a sample in $\mathbf{L}^s$ whose adjusted bounding box is empty, we cannot create an edge between $\mathbf{N}_s$ and $\mathbf{N}_t$

4 Parametric Motion Graph Lookup

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

For each outgoing edge of $\mathbf{N}_s$, we begin by finding the k-nearest neighbors to $\tilde{l}^s$ from the transition sample list in terms of Euclidean distance, where k is normally one more than the number of dimensions of $\mathscr{P}^s$. Let us call these neighbors $l^s_1, \ldots, l^s_k$, ordered from closest to farthest from $\tilde{1}^s$. Following the work of Allen et al. [2002], each l^s_i is associated with a weight, w_i:

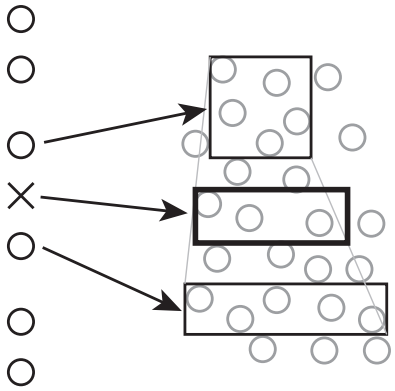

Figure 5: Mapping a parameter vector, depicted by the X, from the 1-D parameter space on the left, to a valid transition region in the 2-D parameter space on the right. X's bounding box is the weighted average of the bounding boxes for its 2-nearest neighbors.

$$w_i = \frac{w'_i}{\sum_{j=1}^{k} w'_j} \tag{1}$$

$$w'_i = \frac{1}{\varepsilon(\tilde{l}^s, l^s_i)} - \frac{1}{\varepsilon(\tilde{l}^s, l^s_k)} \tag{2}$$

where ε gives the Euclidean distance between parameter samples.

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

$$\mathbf{B}(\mathbf{N}_s, \mathbf{N}_t) = \sum_{i=1}^{k} w_i * \beta(l^s_i) \tag{3}$$

where $\beta(l^s_i)$ gives the value of the bounding box for the sample l^s_i, represented by the location of the box's center and its width in each dimension, as stored in the edge (see Figure 5). Similarly, we can compute the normalized transition point as a weighted sum of the average, normalized transition points for each l^s_i stored in the edge.

5 Results

The examples in this paper were computed on a laptop computer with a $1.75GHz$ Pentium M Processor, $1GB$ of RAM, and an ATI Mobility Radeon X300 graphics card. All of the generated motions were sampled at $30Hz$. Each of the parametric motion graphs we generated can synthesize and render streams of motion at more than 180 frames per second. In this section, we provide details for some of the example parametric motion graphs we designed for interactive character control. Following the description of these graphs, we present the results of a number of experiments for testing the usefulness of these graph structures in interactive applications.

5.1 Graphs

To build each graph described below, an author starts by choosing the parameterized motion spaces needed for the graph from our available motion space database. These parameterized motion spaces then appear as disconnected nodes in the graph. Next, the author simply chooses two nodes to generate an edge between and specifies values for $\mathbf{T}_{GOOD}$, $\mathbf{T}_{BAD}$, n^s, and n^t. While it is possible to set the values of $\mathbf{T}_{GOOD}$ and $\mathbf{T}_{BAD}$ without user input, the ability to adjust these values allows an author to determine where to set the tradeoff between motion quality and flexibility. For our example

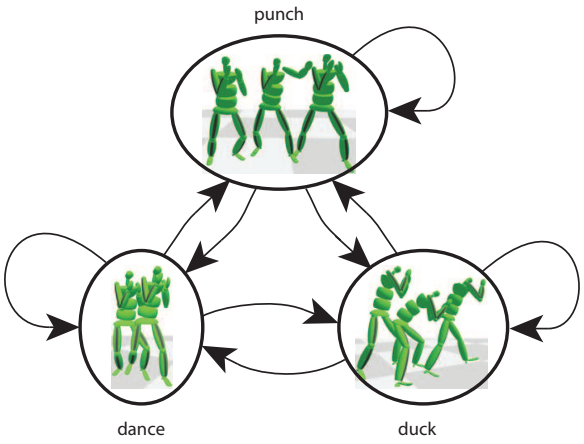

Figure 6: A boxing graph.

graphs, the amount of time it took to generate a single edge varied from $2-147$ seconds, depending on the complexity of the source and target parameterized motion spaces. In practice, we found it took two or three iterations in order to tune the parameters $\mathbf{T}_{GOOD}$ and $\mathbf{T}_{BAD}$ for each edge. Empirically, we find that setting $\mathbf{T}_{GOOD}$ to .5 and $\mathbf{T}_{BAD}$ to .7 serves as a good starting point. Even for our largest graph, it was possible to store the graph's structure and edge information in a plain text file requiring less than $50KB$ of space.

5.1.1 Single Node Locomotion Graphs

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

Since our technique requires little authoring effort, it is possible to experiment with non-obvious motions. We also built a parametric motion graph that encodes locomotion control through cartwheeling. Like the graphs for walking and running, our cartwheel locomotion graph contains only a single node. This node represents a parameterized motion space of a character doing a cartwheel, rotating towards the right by varying amounts on one foot, and then doing a cartwheel in another direction. Again, the parameterized motion space maps the angular change in travel direction of the character from the beginning of the motion to the end (between -13 degrees and 157 degrees) to synthesized motions. Each of these single node locomotion graphs take less than 5 minutes to build from beginning to end using our unoptimized system.

5.1.2 General Graphs

In addition to single-node locomotion graphs, we have also built several larger graphs. The simplest is a two-node graph that combines the walking and running nodes described earlier. This graph can control the travel direction of a character that can both run and walk. We have also built a seven-node, fourteen-edge graph containing motions for walking and running at different curvatures, sitting down and standing up from chairs of heights between 1ft and 1.9ft tall, stepping up onto and stepping off of platforms of heights between .8ft and 1.8ft tall, and leaping over distances between 2 and 3ft. It takes about 11 minutes to build this graph. The final graph organizes a total of 256 example motions so that they can be blended to produce continuous streams of controllable animation.

In order to show that our technique works when controlling a number of different non-locomotion actions, we built a parametric motion graph that encodes the motions of a boxer punching, ducking, and "dancing" from one foot to the other. The boxing graph consists of exactly three nodes. The first node represents all motions of a boxing character punching to some location in a 6ft wide, 2ft tall, and 5ft deep space. The parameterized motion space maps desired punch locations in relation to the starting configuration of the root to synthesized punching motions. The second node of the boxing graph represents motions of a boxing character ducking below different heights (between 3.4ft and 5.6ft from the ground) and is parameterized on how low the character ducks. The third and final node encodes motions of a character "dancing" from one foot to another while maintaining a boxing ready stance. When "dancing", the character rotates by different amounts (between -27 and 46 degrees). Thus, the "dancing" motion space maps the change in facing direction from the beginning of the motion to the end of the motion to synthesized "dancing" motions. In total, the parameterized motion spaces used for these graph nodes blend between 275 different motion captured examples. A discrete motion transition graph that represents transitions between this number of motions would be very large and unwieldy. In contrast, our final graph (Figure 6) contains only nine edges, one connecting every pair of nodes. It takes approximately 7 minutes and 40 seconds to build the graph.

5.2 Applications

We implemented a number of different applications to test the usefulness of our technique. In this section, we describe these applications in detail and provide our results.

5.2.1 Random Graph Walks

Our first application shows that parametric motion graphs can generate seamless, high-fidelity motion streams in realtime. For each of our graphs, we can produce a random stream of motion by taking random walks on the graph. We start by choosing a random node and parameter vector from the graph. When the parameterized motion space associated with the node is supplied with the chosen parameter vector, we can render a motion that matches this parameter request in realtime. While playing the motion, when we reach the possible transition region, we randomly choose an edge from those leaving our current node. The node that this edge points to is the new target node. Using the method described in Section 4, we compute the optimal transition point and the parameter subspace of the target node that we can transition to from our current parameter vector. We then randomly choose a new target parameter vector enclosed in this subspace. Finally, when we reach the blending window centered at the optimal transition point, we compute the point cloud alignment between our current motion and our newly chosen motion, align the motions, and then blend between them. This process is repeated indefinitely. By randomly generating continuous streams of motion, we can confirm that our technique produces smooth motions and avoids poor transitions. Please see our accompanying video for the results of this application.

5.2.2 Target Directed Control

Our second application tests whether our walking character can accurately reach a target location using a greedy graph search similar to the one used for locomotion control in [Srinivasan et al. 2005]

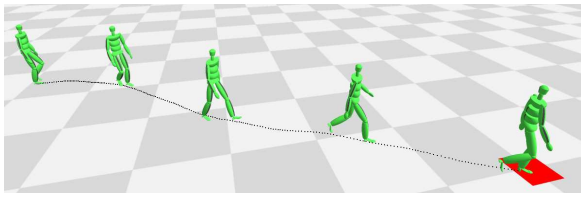

Figure 7: Using parametric motion graphs, this character walks to a specified location, depicted by the red square on the ground.

and crowd control in [Sung et al. 2005]. For this application, we generate a motion stream in the same way we do for random graph walks, except that when it is time to choose a new parameter vector from the target bounding box, we choose the parameter vector that best adjusts the character's travel direction towards a target. Figure 7 shows that our character is able to accurately reach a target location without wandering by using this simple control algorithm.

We also allow a user to request that the character reach the target location oriented in a particular direction. For this case, we choose the parameter vector that both adjusts the character's travel direction towards the target and orients the character towards the desired facing direction. We place more weight on the orientation component of this optimization function as the character gets closer to the target. In several cases, our character can perform the requested action very well. But we find that in others, the character approaches the target and then turns in circles trying to orient themselves. This result is anticipated as we know that the character's minimum turning radius is quite large. Inspired by the work of Reitsma and Pollard [2004], we used a discrete, brute force method to embed our parametric motion graph in the environment in hopes of better understanding this problem. This embedding made it clear that our character could easily meet location constraints within a reasonable radius but that for most locations, there were only a few orientations that the character could be in when they arrived.

5.2.3 Interactive Character Control

Our last and most important application allows users to interactively control a character. To do this, we attach a function to each node that translates user requests to parameters. For example, for walking and cartwheeling, we wanted a user to control the travel direction of the character by specifying the desired travel direction using a joystick. So, we attached a function to each of these nodes that could compute the angular change between the character's current direction of travel and desired direction of travel.

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

This process has the effect of creating interactive characters that perform requested actions as accurately as possible without introducing poor transitions between motion clips. By limiting the transitions to good ones, our characters occasionally miss targets; in these cases, the character still "reacts" to the target by choosing a good transition that gets closest to meeting the request. Using our technique, we produced walking, running, and cartwheeling characters whose travel direction can be controlled; a character who can either run or walk in a desired travel direction; a boxing character that is able to change facing direction while "dancing", punch towards specified 3D locations, and duck below a specified height; and a character that can walk or run in a desired direction, step onto and off of platforms, sit down and stand up from chairs, and leap

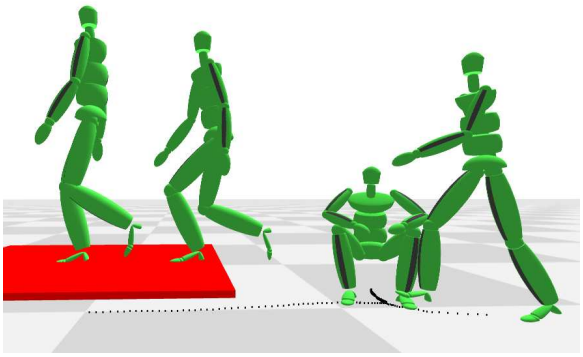

Figure 8: An interactively controllable character using parametric motion graphs. The character has just stepped up onto a platform after sitting down in a chair.

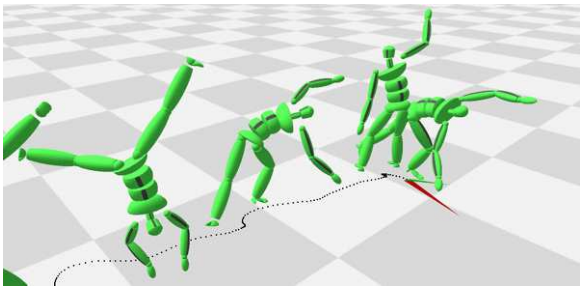

Figure 9: An interactively controllable cartwheeling character using parametric motion graphs to smoothly move through an environment. The character has changed cartwheeling direction in order to travel in the user-requested direction depicted by the red arrow.

over distances. Figures 1, 8, 9, and 10 show some of our results. The accompanying video provides examples of all of these characters being controlled in realtime.

6 Discussion

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

While we use the method of Kovar and Gleicher [2004] to produce parameterized motion spaces, our methods do not require that motions be generated with any particular parametric motion synthesis method. However, parametric motion graphs do require smooth parameterized motion spaces; our sampling and interpolation methods depend on nearby motions in parameter space looking similar (see Section 3). While we have not provided an example, our method should work just as well using a procedural parametric synthesis method, as long as it produces smooth motion spaces.

One larger limitation is that we cannot represent transitions between two nodes if there is any motion in the source node that cannot transition to the target node. For example, consider two nodes that represent a person walking at different curvatures where the first allows a much wider range of curvatures than the other. Because the extreme motions of the first node do not look like any of the motions in the second node, we will be unable to create an edge between the nodes. One possible solution to is to dynamically add additional nodes to the graph when large enough continuous pieces of a source node can transition to the target node. This new node

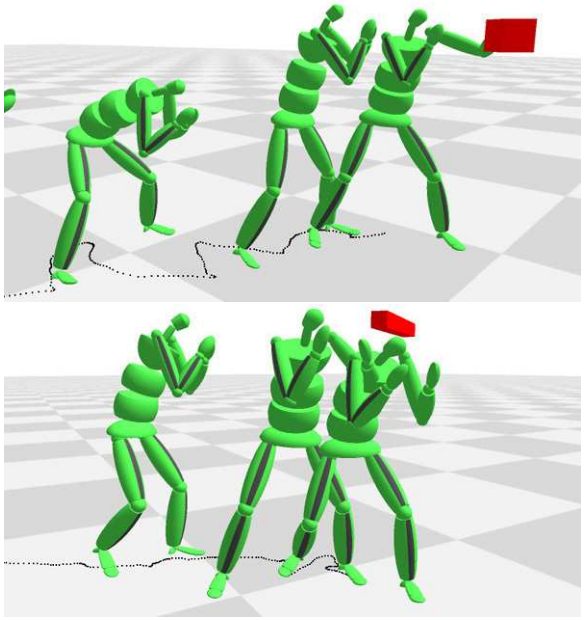

Figure 10: An interactively controllable boxing character that uses parametric motion graphs. The character is punching towards a user-requested target in the top image. In the bottom image, the character is ducking below a user specified height.

would represent the same parameterized motion space as the first except that its range would be limited to the range of parameters that have valid transitions to the target node.

Our method is also limited to transitioning between motions only at one point near the end of a clip. Similarly, we do not adjust the parameter vector while generating a motion. These limitations mean that for motion spaces that represent long motions, it may take time for the character to react to user requests. This problem can be lessened by choosing parameterized motion spaces that represent short motion clips. Other methods for improving the responsiveness of longer clips is future work. We also plan to explore better local search methods than the greedy one in Section 5.2.2.

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

7 Acknowledgements

We thank Lucas Kovar and Mohamed Eldawy for their help during development, Jehee Lee for the use of his boxing data, and the NSF for their support through grants CCR-0204372 and IIS-0416284.

References

ALLEN, B., CURLESS, B., AND POPOVIC, Z. 2002. Articulated body deformation from range scan data. *ACM Transactions on Graphics*.

ARIKAN, O., AND FORSYTHE, D. A. 2002. Interactive motion generation from examples. *ACM Transactions on Graphics*.

ARIKAN, O., FORSYTH, D. A., AND O'BRIEN, J. 2003. Motion synthesis from annotations. *ACM Transactions on Graphics*.

BRUDERLIN, A., AND WILLIAMS, L. 1995. Motion signal processing. In *ACM SIGGRAPH*.

GLEICHER, M., SHIN, H. J., KOVAR, L., AND JEPSEN, A. 2003. Snap-together motion: Assembling run-time animation. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics*.

HODGINS, J., W.WOOTEN, BROGAN, D., AND O'BRIEN, J. 1995. Animating human athletics. In *ACM SIGGRAPH*.

KIM, T., PARK, S., AND SHIN, S. 2003. Rhythmic-motion synthesis based on motion-beat analysis. *ACM Transactions on Graphics*.

KOVAR, L., AND GLEICHER, M. 2004. Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics*.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. *ACM Transactions on Graphics*.

KWON, T., AND SHIN, S. Y. 2005. Motion modeling for on-line locomotion synthesis. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

LAMOURET, A., AND VAN DE PANNE, M. 1996. Motion synthesis by example. In *Eurographics workshop on Computer animation and simulation*.

LEE, J., AND LEE, K. H. 2004. Precomputing avatar behavior from human motion data. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

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*.

MIZUGUCHI, M., BUCHANAN, J., AND CALVERT, T. 2001. Data driven motion transitions. In *Eurographics Short Presentations*.

MUKAI, T., AND KURIYAMA, S. 2005. Geostatistical motion interpolation. In *ACM SIGGRAPH*.

PARK, S. I., SHIN, H. J., AND SHIN, S. Y. 2002. On-line locomotion generation based on motion blending. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

PERLIN, K., AND GOLDBERG, A. 1996. Improv: A system for scripting interactive actors in virtual worlds. In *ACM SIGGRAPH*.

PERLIN, K. 1995. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*.

REITSMA, P. S. A., AND POLLARD, N. S. 2004. Evaluating motion graphs for character navigation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

ROSE, C., GUENTER, B., BODENHEIMER, B., AND COHEN, M. 1996. Efficient generation of motion transitions using spacetime constraints. In *ACM SIGGRAPH*.

ROSE, C., COHEN, M., AND BODENHEIMER, B. 1998. Verbs and adverbs: multidimensional motion interpolation. *IEEE CG&A*.

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video textures. In *ACM SIGGRAPH*.

SHIN, H. J., AND OH, H. S. 2006. Fat graphs: Constructing an interactive character with continuous controls. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

SRINIVASAN, M., METOYER, R. A., AND MORTENSEN, E. N. 2005. Controllable character animation using mobility maps. *Graphics Interface*.

SUN, H., AND METAXAS, D. 2001. Automating gait animation. In *ACM SIGGRAPH*.

SUNG, M., KOVAR, L., AND GLEICHER, M. 2005. Fast and accurate goal-directed motion synthesis for crowds. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

WILEY, D., AND HAHN, J. 1997. Interpolation synthesis of articulated figure motion. *IEEE CG&A*.