

# Interactive Motion Generation from Examples

Okan Arikan

D. A. Forsyth

University of California, Berkeley

## Abstract

There are many applications that demand large quantities of natural looking motion. It is difficult to synthesize motion that looks natural, particularly when it is people who must move. In this paper, we present a framework that generates human motions by cutting and pasting motion capture data. Selecting a collection of clips that yields an acceptable motion is a combinatorial problem that we manage as a randomized search of a hierarchy of graphs. This approach can generate motion sequences that satisfy a variety of constraints automatically. The motions are smooth and human-looking. They are generated in real time so that we can author complex motions interactively. The algorithm generates multiple motions that satisfy a given set of constraints, allowing a variety of choices for the animator. It can easily synthesize multiple motions that interact with each other using constraints. This framework allows the extensive re-use of motion capture data for new purposes.

**CR Categories:** I.2.7 [Artificial Intelligence]: Problem Solving, Control Methods and Search—Graph and tree search strategies I.3.7 [COMPUTER GRAPHICS]: Three-Dimensional Graphics and Realism—Animation

**Keywords:** Motion Capture, Motion Synthesis, Human motion, Graph Search, Clustering, Animation with Constraints

## 1 Introduction

Motion is one of the most important ingredients of CG movies and computer games. Obtaining realistic motion usually involves key framing, physically based modelling or motion capture. Creating natural looking motions with key framing requires lots of effort and expertise. Although physically based modelling can be applied to simple systems successfully, generating realistic motion on a computer is difficult, particularly for human motion. A standard solution is motion capture: motion data for an approximate skeletal hierarchy of the subject is recorded and then used to drive a reconstruction on the computer. This allows other CG characters to be animated with the same motions, leading to realistic, “human looking” motions for use in movies or games. The biggest drawbacks of motion capture are:

1. Most motion capture systems are very expensive to use, because the process is time consuming for actors and technicians and motion data tends not to be re-used.

2. It is very hard to obtain motions that do exactly what the animator wants. Satisfying complex timed constraints is difficult and may involve many motion capture iterations. Examples include being at a particular position at a particular time accurately or synchronizing movement to a background action that had been shot before.

In order to make motion capture widely available, the motion data needs to be made re-usable. This may mean using previous motion capture data to generate new motions so that certain requirements are met, transferring motions from one skeletal configuration to another so that we can animate multiple figures with the same motion without it looking “funny”, or changing the style of the motion so that the directors can have higher level control over the motion. There are three natural stages of motion synthesis:

1. **Obtaining motion demands** involves specifying constraints on the motion, such as the length of the motion, where the body or individual joints should be or what the body needs to be doing at particular times. These constraints can come from an interactive editing system used by animators, or from a computer game engine itself.
2. **Generating motion** involves obtaining a rough motion that satisfies the demands. In this paper, we describe a technique that cuts and pastes bits and pieces of example motions together to create such a motion.
3. **Post processing** involves fixing small scale offensive artifacts. An example would involve fixing the feet so that they do not penetrate or slide on the ground, lengthening or shortening strides and fixing constraint violations.

In this paper, we present a framework that allows synthesis of new motion data meeting a wide variety of constraints. The synthesized motion is created from example motions at interactive speeds.

## 2 Related Work

In the movie industry, motion demands are usually generated by animators. However, automatic generation of motion demands is required for autonomous intelligent robots and characters [Funge et al. 1999]. An overview of the automatic motion planning can be found in [Latombe 1999; O’Rourke 1998].

Generating motion largely follows two threads: using examples and using controllers. Example based motion synthesis draws on an analogy with texture synthesis where a new texture (or motion) that looks like an example texture (or motion example) needs to be synthesized [Efros and Leung 1999; Heeger and Bergen 1995]. Pullen and Bregler used this approach to create cyclic motions by sampling motion signals in a “signal pyramid” [2000]. They also used a similar approach to fetch missing degrees of freedom in a motion from a motion capture database [Pullen and Bregler 2002]. The sampling can also be done in the motion domain to pick clips of motions to establish certain simple constraints [Lamouret and van de Panne 1996; Schodl et al. 2000]. A roadmap of all the motion examples can be constructed and searched to obtain a desired motion

[Choi et al. 2000; Lee et al. 2002; Kovar et al. 2002]. The clips in this roadmap can also be parameterized for randomly sampling different motion sequences [Li et al. 2002]. The motion signals can also be clustered. The resulting Markov chain can be searched using dynamic programming to find a motion that connects two keyframes [Molina-Tanco and Hilton 2000] or used in a variable length Markov model to infer behaviors [Galata et al. 2001] or directly sampled from to create new motions [Bowden 2000]. This is similar to our work. However, our clustering method does not operate on body configurations and our probabilistic search strategy is more effective than dynamic programming as it will be explained below. Types of probabilistic search algorithms have also been used in physically based animation synthesis [Chenney and Forsyth 2000] and rendering [Veach and Guibas 1997]. Controller based approaches use physical models of systems and controllers that produce outputs usually in the form of forces and torques as a function of the state of the body. These controllers can be designed specifically to accomplish particular tasks [Brogan et al. 1998; Hodgins et al. 1995] or they can be learned automatically using statistical tools [Grzeszczuk and Terzopoulos 1995; Grzeszczuk et al. 1998; Mataric 2000].

The motion data can also be post processed to fix problems such as feet sliding on the ground or some constraints not being satisfied [Gleicher 1998; Lee and Shin 1999; Popovic 1999; Rose et al. 1996]. This usually involves optimization of a suitable displacement function on the motion signal. Different body sizes move according to different time scales, meaning that motion cannot simply be transferred from one body size to another; modifying motions appropriately is an interesting research problem [Hodgins and Pollard 1997].

### 3 Synthesis as Graph Search

We assume there is a set of  $N$  motion sequences forming our dataset, each belonging to the same skeletal configuration. Every motion is discretely represented as a sequence of frames each of which has the same  $M$  degrees of freedom. This is required to be able to compare two motions and to be able to put clips from different motion sequences together. We write the  $i$ 'th frame of  $s$ 'th motion as  $s_i$ .

#### 3.1 Motion Graph

The collection of motion sequences could be represented as a directed graph. Each frame would be a node. There would be an edge from every frame to every frame that could follow it in an acceptable splice. In this graph, there would be (at least) an edge from the  $k$ 'th frame to the  $k + 1$ 'th frame in each sequence. This graph is not a particularly helpful representation because it is extremely large — we can easily have tens of thousands of nodes and hundreds of thousands of edges — and it obscures the structure of the sequences.

Instead, we collapse all the nodes (frames) belonging to the same motion sequence together. This yields a graph  $G$  where the nodes of  $G$  are individual motion sequences and there is an edge from  $s$  to  $t$  for every pair of frames where we can cut from  $s$  to  $t$ . Since edges connect frames, they are labelled with the frames in the incident nodes (motion sequences) that they originate from and they point to. We also assume that the edges in  $G$  are attached a cost value which tells us the cost of connecting the incident frames. If cutting from one sequence to another along an edge introduces a discontinuous motion, then the cost attached to the edge is high. Appendix A introduces the cost function that we used. The collapsed graph still has the same number of edges.

For an edge  $e$  from  $s_i$  to  $t_j$ , let  $fromMotion(e) = s$ ,

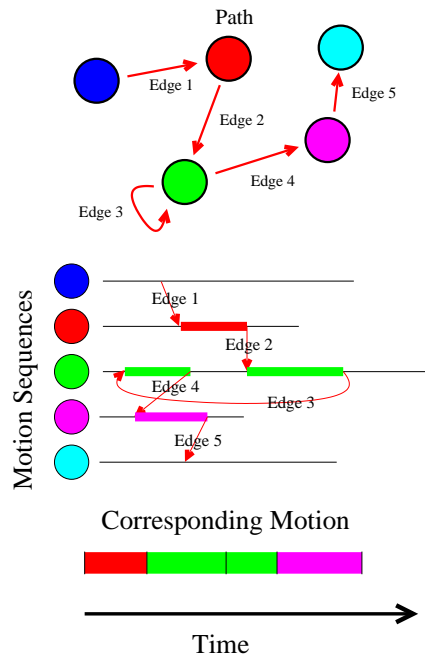


Figure 1: We wish to synthesize human motions by splicing together pieces of existing motion capture data. This can be done by representing the collection of motion sequences by a directed graph (**top**). Each sequence becomes a node; there is an edge between nodes for every frame in one sequence that can be spliced to a frame in another sequence or itself. A valid path in this graph represents a collection of splices between sequences, as the **middle** shows. We now synthesize constrained motion sequences by searching appropriate paths in this graph using a randomized search method.

$toMotion(e) = t$ ,  $fromFrame(e) = i$ ,  $toFrame(e) = j$  and  $cost(e)$  be the cost associated with the edge (defined in Appendix A). In this setting, any sequence of edges  $e_1 \dots e_n$  where  $toMotion(e_i) = fromMotion(e_{i+1})$  and  $toFrame(e_i) < fromFrame(e_{i+1})$ ,  $\forall i$ ,  $1 \leq i < n$  is a valid path and defines a legal sequence of splices. (figure 1).

#### 3.2 Constraints

We wish to construct paths in the motion graph that satisfy constraints. Many constraints cannot be satisfied exactly. For example, given two positions, there may not be any sequence of frames in the collection that will get us from the first position to the second position exactly. We define **hard constraints** to be those that can (and must) be satisfied exactly. Typically, a hard constraint involves using a particular frame in a particular time slot. For example, instead of considering all valid paths, we can restrict ourselves to valid paths that pass through particular nodes at particular times. This way, we can constrain the moving figure to be at a specific pose at a specific time. This enables us to search for motions such as jumping, falling, or pushing a button at a particular time.

A **soft constraint** cannot generally be met exactly. Instead we score sequences using an objective function that reflects how well the constraint has been met and attempt to find extremal sequences. One example is the squared distance between the position of the constraint and the actual position of the body at the time of the constraint. Example soft constraints include:

1. The total number of frames should be a particular number.

2. The motion should not penetrate any objects in the environment.
3. The body should be at a particular position and orientation at a particular time.
4. A particular joint should be at a particular position (and maybe having a specific velocity) at a specific time.
5. The motion should have a specified style (such as happy or energetic) at a particular time.

Finding paths in the motion graph that satisfy the hard constraints and optimize soft constraints involves a graph search. Unfortunately, for even a small collection of motions, the graph  $G$  has a large number of edges and straightforward search of this graph is computationally prohibitive. The main reason is the need to enumerate many paths. There are, in general, many perfectly satisfactory motions that satisfy the constraints equally well. For example, if we require only that the person be at one end of a room at frame 0 and near the other end at frame 5000, unless the room is very large, there are many motions that satisfy these constraints.

## 4 Randomized Search

The motion graph is too hard to search with dynamic programming as there are many valid paths that satisfy the constraints equally well. There may be substantial differences between equally valid paths — in the example above, whether you dawdle at one side of the room or the other is of no significance. This suggests summarizing the graph to a higher level and coarser presentation that is easier to search. Branch and bound algorithms are of no help here, because very little pruning is possible.

In order to search the graph  $G$  in practical times, we need to do the search at a variety of levels where we do the large scale motion construction first and then “tweak” the details so that the motion is continuous and satisfies the constraints as well as possible. Coarser levels should have less complexity while allowing us to explore substantially different portions of the path space. In such a representation, every level is a summary of the one finer level. Let  $G' \leftarrow G'' \leftarrow G''' \leftarrow \dots \leftarrow G^n \leftarrow G$  be such a hierarchical representation where  $G'$  is the coarsest level and  $G$  is the finest. We will first find a path in  $G'$  and then push it down the hierarchy to a path in  $G$  for synthesis.

### 4.1 Summarizing the Graph

All the edges between two nodes  $s$  and  $t$  can be represented in a matrix  $P_{st}$ . The  $(i, j)$ 'th entry of  $P_{st}$  contains the weight of the edge connecting  $s_i$  to  $t_j$  and infinity if there is no such edge. In the appendix A, we give one natural cost function  $C(s_i, t_j)$  for edge weights. We now have:

$$(P_{st})_{ij} = \begin{cases} C(s_i, t_j) & \text{if there is an edge from } s_i \text{ to } t_j \\ \infty & \text{otherwise.} \end{cases}$$

The cost function explained in section A causes the  $P$  matrices to have non-infinite entries to form nearly elliptical groups (figure 2). This is due to the fact that if two frames are similar, most probably their preceding and succeeding frames also look similar.

In order to summarize the graph, we cluster the edges of  $G$ . We now have  $G'$ , whose nodes are the same as the nodes of  $G$ , and whose edges represent clusters of edges of  $G$  in terms of their *fromFrame* and *toFrame* labels. We require that, if there is a cut between two sequences represented by an edge between two nodes in  $G$ , there be at least one edge between the corresponding nodes in

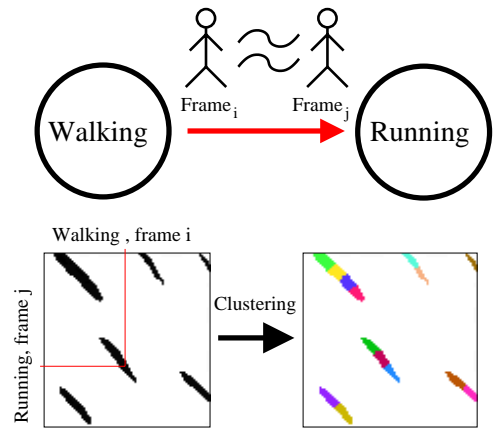


Figure 2: Every edge between two nodes representing different motion clips can be represented as a matrix where the entries correspond to edges. Typically, if there is one edge between two nodes in our graph, there will be several, because if it is legal to cut from one frame in the first sequence to another in the second, it will usually also be legal to cut between neighbors of these frames. This means that, for each pair of nodes in the graph, there is a matrix representing the weights of edges between the nodes. The  $i, j$ 'th entry in this matrix represents the weight for a cut from the  $i$ 'th frame in the first sequence to the  $j$ 'th frame in the second sequence. The weight matrix for the whole graph is composed as a collection of blocks of this form. Summarizing the graph involves compressing these blocks using clustering.

$G'$ . If this were not the case, our summary would rule out potential paths. In order to insure that this condition holds and because the graph is very large, we cluster edges connecting every pair of nodes in  $G$  separately. We cluster unconnected edge groups of  $G$  from the  $P$  matrices (defined between every pair of nodes) using k-means [Bishop 1995]. The number of clusters is chosen as  $\frac{\text{majoraxislength}}{\text{minoraxislength}}$  for each group where the axis lengths refer to the ellipse that fits to the cluster (obtained through Principal Component Analysis).

The nodes of  $G'$  are the same as the nodes of  $G$ . The edges connecting nodes in  $G'$  are cluster centers for clusters of edges connecting corresponding nodes in  $G$ . The centers are computed by taking the average of the edges in terms of *fromFrame*, *toFrame* and *cost* values. At this point, every edge in  $G'$  represents many edges in  $G$ . We would like to have a tree of graph representations whose root is  $G'$ , and whose leaves are  $G$ . We use k-means clustering to split each cluster of edges in half at each intermediate level and obtain a hierarchical representation  $G' \leftarrow G'' \leftarrow G''' \leftarrow \dots \leftarrow G^n \leftarrow G$  for the original graph  $G$ . This is an instance of Tree-Structured Vector Quantization [Gersho and Gray 1992].

Thus, in our summarized graph  $G'$ , each edge is the root of a binary tree and represents all the edges in close neighborhood in terms of the edge labels. Note that the leaf edges are the edges in the original graph and intermediate edges are the averages of all the leaf edges beneath them. A path in  $G$  represents a sequence of clips; so does a path in  $G'$ , but now the positions of the clip boundaries are quantized, so there are fewer paths.

### 4.2 Searching the Summaries

While searching this graph, we would like to be able to generate different alternative motions that achieve the same set of constraints. During the search, we need to find paths close to optimal solutions but do not require exact extrema, because they are too hard to find. This motivates a random search. We used the following search strategy:

1. Start with a set of  $n$  valid random “seed” paths in the graph  $G'$
2. Score each path and score all possible mutations
3. Where possible mutations are:
  - (a) Delete some portion of the path and replace it with 0 or 1 hops.
  - (b) Delete some edges of the path and replace them with their children
4. Accept the mutations that are better than the original paths
5. Include a few new valid random “seed” paths
6. Repeat until no better path can be generated through mutations

Intuitively the first mutation strategy replaces a clip with a (hopefully) better one and the second mutation strategy adjusts the detailed position of cut boundaries. Since we start new random “seed” paths at every iteration, the algorithm does not get stuck at a local optimum forever. Section 4.2.2 explains these mutations in more detail.

Hard constraints are easily dealt with; we restrict our search to paths that meet these constraints. Typically hard constraints specify the frame (in a particular node) to be used at a particular time. We do this by ensuring that “seed” paths meet these constraints, and mutations do not violate them. This involves starting to sample the random paths from the hard constraint nodes and greedily adding sequences that get us to the next hard constraint if any. Since the path is sampled at the coarse level, a graph search can also be performed between the constraint nodes. At every iteration we check if the proposed mutation deletes a motion piece that has a hard constraint in it. Such mutations are rejected immediately. Note that here we assume the underlying motion graph is connected. Section 4.2.1 explains the constraints that we used in more detail.

Notice that this algorithm is similar to MCMC search (a good broad reference to application of MCMC is [Gilks et al. 1996]). However, it is difficult to compute proposal probabilities for the mutations we use, which are strikingly successful in practice.

This is an online algorithm which can be stopped at anytime. This is due to the fact that edges in intermediate graphs  $G' \dots G^n$  also represent connections and are valid edges. Thus we do not have to reach the leaf graph  $G$  to be able to create a path (motion sequence). We can stop the search iteration, take the best path found so far, and create a motion sequence. If the sequence is not good enough, we can resume the search from where we left off to get better paths through mutations and inclusion of random paths. This allows an intuitive computation cost vs. quality tradeoff.

#### 4.2.1 Evaluating a Path

Since during the search all the paths live in a subspace implied by the hard constraints, these constraints are always satisfied. Given a sequence of edges  $e_1 \dots e_n$ , we score the path using the imposed soft constraints. For each constraint, we compute a cost where the cost is indicative of the satisfaction of the constraint. Based on the scores for each of the constraints, we weight and sum them to create a final score for the path (The  $S$  function in equation 1). We also add the sum of the costs of the edges along the path to make sure we push the search towards paths that are continuous. The weights can be manipulated to increase/decrease the influence of a particular soft constraint. We now have an expression of the form:

$$S(e_1 \dots e_n) = w_c * \sum_{i=1}^n cost(e_i) + w_f * F + w_b * B + w_j * J \quad (1)$$

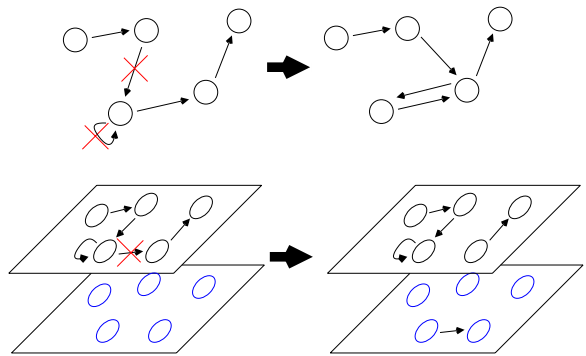


Figure 3: The two mutations are: deleting some portion of the path (top-left, crossed out in red) and replacing that part with another set of edges (top-right), and deleting some edges in the path (bottom-left) and replacing deleted edges with their children in our hierarchy (bottom-right)

Where  $w_c, w_f, w_b$  and  $w_j$  are weights for the quality (continuity) of the motion, how well the length of the motion is satisfied, how well the body constraints are satisfied and how well the joints constraints are defined. We selected these weights such that an error of 10 frames increases the total score the same amount as an error of 30 centimeters in position and 10 degrees in orientation. The scores  $F, B$  and  $J$  are defined as:

1.  $F$ : For the number of frame constraints, we compute the squared difference between the actual number of frames in the path and the required number of frames.
2.  $B$ : For body constraints, we compute the distance between the position and orientation of the constraint versus the actual position and orientation of the torso at the time of the constraint and sum the squared distances. The position and orientation of the body at the constraint times are found by putting the motion pieces implied by the subsequent edges together (figure 1). This involves taking all the frames of motion  $toMotion(e_i)$  between frames  $fromFrame(e_{i+1})$  and  $toFrame(e_i)$  and putting the sequence of frames starting from where the last subsequence ends or from the first body constraint if there is no previous subsequence. Note that we require that we have at least two body constraints enforcing the position/orientation of the body at the beginning of the synthesized motion (so that we know where to start putting the frames down) and at the end of the synthesized motion. The first body constraint is always satisfied, because we always start putting the motions together from the first body constraint.
3.  $J$ : For joint constraints, we compute the squared distance between the position of the constraint and the position of the constrained joint at the time of the constraint and sum the squared distance between the two. To determine the configuration of the body at the time at which the constraint applies, we must assemble the motion sequence up to the time of the constraint; in fact, most of the required information such as the required transformation between start and end of each cut is already available in the dataset.

#### 4.2.2 Mutating a Path

We implemented two types of mutations which can be performed quickly on an active path.



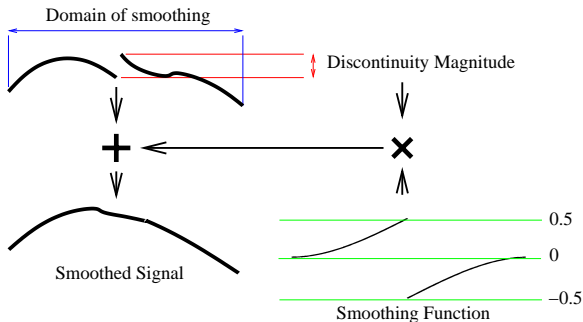


Figure 4: In the synthesized motion, discontinuities in orientation are inevitable. We deal with these discontinuities using a form of localized smoothing. At the top left, a discontinuous orientation signal, with its discontinuity shown at the top right. We now construct an interpolant to this discontinuity, shown on the bottom right and add it back to the original signal to get the continuous version shown on the bottom left. Typically, discontinuities in orientation are sufficiently small that no more complex strategy is necessary.

1. **Replace a sequence** by selecting two edges  $e_i$  and  $e_{i+j}$  where  $0 \leq j \leq n - i$ , deleting all the edges between them in the path and connecting the unconnected pieces of the path using one or two edges in the top level graph  $G'$  (if possible). Since in the summarized graph, there are relatively fewer edges, we can quickly find edges that connect the two unconnected nodes by checking all the edges that go out from  $toMotion(e_i)$ , and enumerating all the edges that reach to  $fromMotion(e_{i+j})$  and generate a valid path. Note that we enumerate only 0 or 1 hop edges (1 edge or 2 edge connections respectively).
2. **Demoting two edges** to their children and replacing them with one of their children if they can generate a valid path. Doing this mutation on two edges simultaneously allows us to compensate for the errors that would happen if only one of them was demoted.

We check every possible mutation, evaluate them and take the best few. Since the summary has significantly fewer edges than the original graph, this step is not very expensive. If a motion sequence cannot generate a mutation whose score is lower than itself, we decide that the current path is a local minimum in the valid path space and record it as a potential motion. This way, we can obtain multiple motions that satisfy the same set of constraints.

### 4.2.3 Creating and Smoothing the Final Path

We create the final motion by taking the frames between  $toFrame(e_i)$  and  $fromFrame(e_{i+1})$  from each motion  $toMotion(e_i)$  where  $1 \leq i < n$  (figure 1). This is done by rotating and translating every motion sequence so that each piece starts from where the previous one ended. In general, at the frames corresponding to the edges in the path, we will have  $C^0$  discontinuities, because of the finite number of motions sampling an infinite space. In practice these discontinuities are small and we can distribute them within a smoothing window around the discontinuity. We do this by multiplying the magnitude of the discontinuity by a smoothing function and adding the result back to the signal (figure 4). We choose the smoothing domain to be  $\pm 30$  frames (or one second of animation) around the discontinuity and

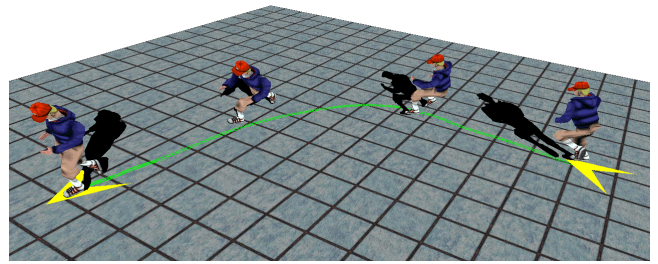


Figure 5: Body constraints allow us to put “checkpoints” on the motion: in the figure, the arrow on the right denotes the required starting position and orientation and the arrow on the left is the required ending position and orientation. All constraints are also time stamped forcing the body to be at the constraint at the time stamp. For these two body constraints, we can generate many motions that satisfy the constraints in real-time.

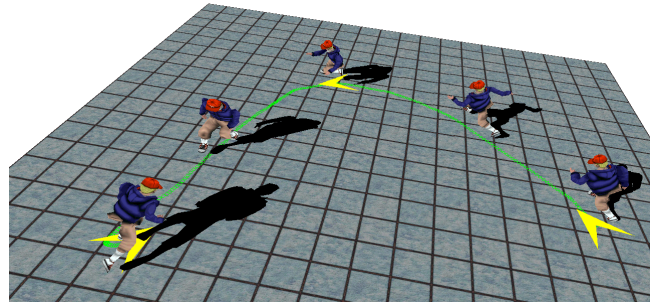


Figure 6: We can use multiple “checkpoints” in a motion. In this figure, the motion is required to pass through the arrow (body constraint) in the middle on the way from the right arrow to the left.

$$y(f) = \begin{cases} 0 & f < d - s \\ \frac{1}{2} * \left(\frac{f-d+s}{s}\right)^2 & d - s \leq f < d \\ -\frac{1}{2} * \left(\frac{f-d+s}{s}\right)^2 + 2 * \left(\frac{f-d+s}{s}\right) - 2 & d \leq f \leq d + s \\ 0 & f > d + s \end{cases}$$

as the smoothing function that gives the amount of displacement for every frame  $f$ , where  $d$  is the frame of the discontinuity and  $s$  if the smoothing window size (in our case 30). To make sure that we interpolate the body constraints (i.e. having a particular position/orientation at a particular frame), we take the difference between the desired constraint state, subtract the state at the time of the constraint and distribute this difference uniformly over the portion of the motion before the time of the constraint. Note that these “smoothing” steps can cause artifacts like feet penetrating or sliding on the ground. However, usually the errors made in terms of constraints and the discontinuities are so small that they are unnoticeable.

### 4.3 Authoring Human Motions

Using iterative improvements of random paths, we are able to synthesize human looking motions interactively. This allows interactive manipulation of the constraints. This is important, because motion synthesis is inherently ambiguous as there may be multiple motions that satisfy the same set of constraints. The algorithm can find these “local minimum” motions that adhere to the same constraints. The animator can choose between them or all the different motions

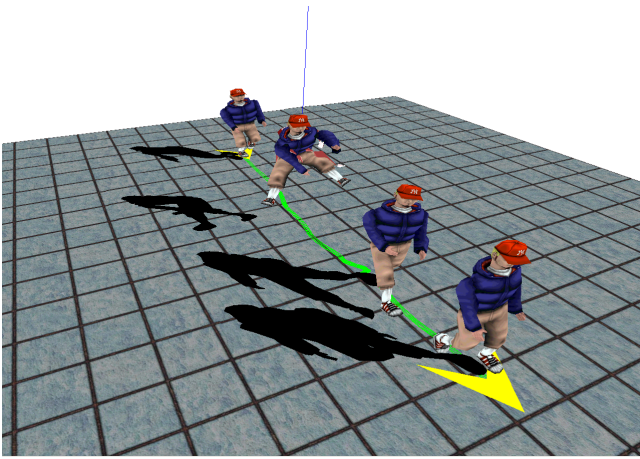


Figure 7: In addition to body constraints, joint constraints can be used to further assign “checkpoints” to individual joints. In this figure, the head of the figure is also constrained to be high (indicated by the blue line), leading to a jumping motion.

can be used to create a variety in the environment. Since the algorithm is interactive, the animator can also see the ambiguity and guide the search by putting extra constraints (figure 6).

Currently, we can constrain the length of the motion, the body’s position and orientation at a particular frame (figure 5,6), a joint (e.g. head, hand) to a particular state at a particular frame (figure 7), or constrain the entire body’s pose at a particular frame (figure 8). Notice that we can synthesize multiple interacting motions independently using hard constraints (figure 9); we simply select the poses, position and orientation at which the figures interact and this framework fills in the missing motion, in a sense, interpolating the constraints. These are only a few of the constraints that can be implemented. As long as the user specifies a cost function that evaluates a motion and attaches a score that is indicative of the animator’s satisfaction with the path, many more constraints can be implemented. For example, if the motions in our database are marked with their individual stylistic attributes, we can also constrain the style of the desired motion by penalizing motions that do not have the particular style. In a computer game environment, we can constrain the synthesized motion to avoid obstacles in the environment. In such a case, body position/orientation constraints can also come from an underlying path planner. Thus, given high level goals (such as going from point A to point B, say) human looking motions can be generated automatically.

## 5 Results

We have presented a framework that allows interactive synthesis of natural looking motions that adhere to user specified constraints. We assess our results using four criteria. Firstly, the motion looks human. Secondly, the motions generated by the method do not have unnatural artifacts such as slipping feet on the ground or jerky movement. Third, the user specified constraints are satisfied, i.e. the motion passes through the required spot at the required time, or the character falls to a particular position (figure 8). Finally, motions are generated interactively — typically depending on the quality of the path desired, an acceptable 300 frame motion is found in between 3 and 10 seconds on an average PC (Pentium III at 800 Mhz). This speed allows interactive motion authoring. For example, we generated the real-time screen captures in the attached video using a dataset of 60-80 unorganized, short (below 300 frames each) motion capture fragments. The average precomputation time

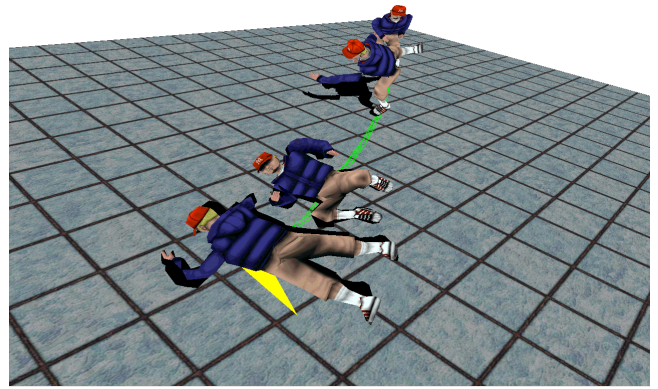


Figure 8: Using hard constraints, we can force the figure to perform specific activities. Here, we constrain the end of the motion to be lying flat on the ground at a particular position/orientation and time. Our framework generates the required tipping and tumbling motion in real-time.

required for this many motions (computing the motion graph) is 5 hours on the same computer. On average, the results shown in the video contain 3-30 motion pieces cut from the original motions.

This framework is completely automatic. Once the input motions are selected, the computation of the hierarchic motion graph does not require any user intervention and the resulting representation is searched in real-time.

For many kinds of constraints the motion synthesis problem is underconstrained; there are many possible combinations of motion pieces that achieve the same set of constraints. Randomized search is well suited to find many different motions that satisfy the constraints. On the other hand, some constraints, may not be met by any motion. In this case, randomized search will try to minimize our objective motion and find the “closest” motion. For example, if the user asks for 100 meters in 5 seconds, the algorithm will tend to put fast running motions together but not necessarily satisfying the constraints. Similarly, if the set of motions to begin with do not form a connected graph, the algorithm will perform searches confined to the unconnected graphs. If there are hard constraints in different unconnected components, we will not even be able to find starting seed paths. From this perspective, the selection of the database to work with is important. In our system, we used 60-100 football motions that have a strong bias towards motions that run forward. However, as the attached video suggest, the randomized search has no problem finding rare motions that turn back to satisfy the constraints. The motion databases that we used were unorganized except that we excluded football warming up and tackling motions unless they were desired (figure 9).

The randomized search scales linearly as a function of the database size with a very small constant. We have tried datasets of 50-100 motions without a noticeable change in the running time of the algorithm. The linearity in the running time comes from the linear increase in the number of alternative mutations at every step. Note that as the database size gets larger, the constant  $\tau$  (Appendix A) that is used to create the edges can get lower since more motions mean that we expect to find better connections between motions, decreasing the number of edges. This will lead to a sublinear increase in the running time.

The framework can work on any motion dataset: it can be created by traditional key framing, physically based modelling or motion capture. For example, we can take the motion data for “Woody” — who may well have been key-framed, from “Toy Story” and create new “Woody” motions automatically. The framework is also appli-

cable to non-human motion synthesis. For example, this framework can be used to generate control signals for robots to achieve a particular task by generating the motion graph for previously known motion-control signal pairs. During the synthesis we can not only synthesize the final robot motion but also the associated control signals that achieve specific goals. Since the generated motions are obtained by putting pieces of motions in the dataset, the resulting motions will also carry the underlying style of the data. This way, we can take the motion data for one character, and produce more motions with the intrinsic style of the character.

## 6 Future Work

During the construction of the final motion, better ways of smoothing between adjacent motions could be used to improve realism [Popovic 1999]. Using better post processing, motions could also be synthesized on non-uniform surfaces which the current framework cannot handle. Additional post processing may involve physically based modelling to make sure the synthesized motions are also physically correct.

Automatic integration of higher level stylistic constraints could be incorporated into the framework, avoiding the arduous job of labelling every motion with the intrinsic style by hand. By analyzing patterns in the motion dataset, we might also infer these styles or obtain higher level descriptions [Brand and Hertzmann 2001]. The synthesized motions are strictly bound to the motions that were available in the original dataset. However, it is conceivable that the motions that are very close to the dataset could also be incorporated in the synthesizable motions using learned stylistic variations.

The integrity of the original dataset directly effects the quality of the synthesized motion. For example, if the incoming motion dataset does not contain any “turning left” motions, we will not be able to synthesize motions that involve “turning left”. An automatic way of summarizing the portions of the “possible human motions” space that have not been explored well enough by the dataset could improve the data gathering and eventually the synthesized motions. This could also serve as a palette for artists: some portions of the precomputed motion graph can be paged in and out of memory depending on the required motion. For example, the animator could interactively select the motions that need to be used during the synthesis, and only the portion of the motion graph involving the desired motions could be loaded. This would give animators a tool whereby they can select the set of motions to work with in advance and the new motions will be created only from the artist selected set. Furthermore this encourages comprehensive re-use of motion data.

## 7 Acknowledgements

This research was supported by Office of Naval Research grant no. N00014-01-1-0890, as part of the MURI program. We would like to thank Electronic Arts for supplying us with the motion data.

## A Appendix: Similarity Metric

We define the torso coordinate frame to be the one where the body stands centered at origin on the xz plane and looks towards the positive z axis. Any point  $p$  in the torso coordinate frame can be transformed to the global coordinate frame by  $T(s_i) + R(s_i) \cdot p$ , where  $T(s_i)$  is the  $3 \times 1$  translation of the torso and  $R(s_i)$  is the  $3 \times 1$  rotation of the torso and  $R(s_i)$  represents the rotation matrix associated with the rotation.

We wish to have a weight on edges of the motion graph (section 3.1) that encodes the extent to which two frames can follow each

other. If the weight of an edge is too high, it is dropped from the graph. To compute the weight of an edge, we use the difference between joint positions and velocities and the difference between the torso velocities and accelerations in the torso coordinate frame.

Let  $P(s_i)$  be a  $3 \times n$  matrix of positions of  $n$  joints for  $s_i$  in torso coordinate frame. Equation 2 gives us the difference in joint position and body velocity.

$$D_{s_i,t_j} = [(P(s_i) - P(t_j)) (|T(s_i)| - |T(t_j)|)' (|R(s_i)| - |R(t_j)|)'] \quad (2)$$

We then define the normalizing matrices  $O$  and  $L$  in equation 3 and 4.

$$O = \max_{s_i} (|D_{s_i,s_i}^T D_{s_i,s_{i+1}}|) \quad (3)$$

$$L = \max_{s_i} (|D_{s_i,s_i}' D_{s_i,s_{i+1}}'|) \quad (4)$$

Then the cost function in equation 5 is used to relate  $s_i$  to  $t_j$ .

$$C(s_i,t_j) = \text{trace}(D_{s_i,t_j} M O^{-1} D_{s_i,t_j}^T + D_{s_i,t_j}' T L^{-1} D_{s_i,t_j}'^T) \quad (5)$$

Where diagonal  $(n+2) \times (n+2)$  matrices  $M$  and  $T$  are used to weight different joints differently. For example, position differences in feet are much more noticeable than position differences of hands because the ground provides a comparison frame. We have found  $M$  and  $T$  matrices empirically by trying different choices. Unfortunately, defining a universal cost metric is a hard problem. The metric defined above produces visually acceptable results.

Using this cost metric, we create edges from  $s_i$  to  $t_j$  where  $C(s_i,t_j) < \tau$ . For an edge  $e$  from  $s_i$  to  $t_j$ , we set  $\text{cost}(e) = C(s_i,t_j)$ .  $\tau$  is a user specified quality parameter that influences the number of edges in  $G$ . We have fixed this value so that cuts created between motions along the edges do not have visible artifacts. Note that an error that is visible on a short person may not be visible on an extremely large person. Thus, in theory, the weights must be adjusted from person to person. However, in practice, possible size variation of adult people is small enough that we used the same weights for different people without creating a visible effect.

## References

- BISHOP, C. M. 1995. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford.
- BOWDEN, R., 2000. Learning statistical models of human motion.
- BRAND, M., AND HERTZMANN, A. 2001. Style machines. In *Proceedings of SIGGRAPH 2000*, 15–22.
- BROGAN, D. C., METOYER, R. A., AND HODGINS, J. K. 1998. Dynamically simulated characters in virtual environments. *IEEE Computer Graphics & Applications* 18, 5, 58–69.
- CHENNEY, S., AND FORSYTH, D. A. 2000. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of SIGGRAPH 2000*, 219–228.
- CHOI, M. G., LEE, J., AND SHIN, S. Y. 2000. A probabilistic approach to planning biped locomotion with prescribed motions. Tech. rep., Computer Science Department, KAIST.
- DEMPSTER, W., AND GAUGHRAN, G. 1965. Properties of body segments based on size and weight. In *American Journal of Anatomy*, vol. 120, 33–54.
- EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. In *ICCV (2)*, 1033–1038.
- FORTNEY, V. 1983. The kinematics and kinetics of the running pattern of two-, four- and six-year-old children. In *Research Quarterly for Exercise and Sport*, vol. 54(2), 126–135.



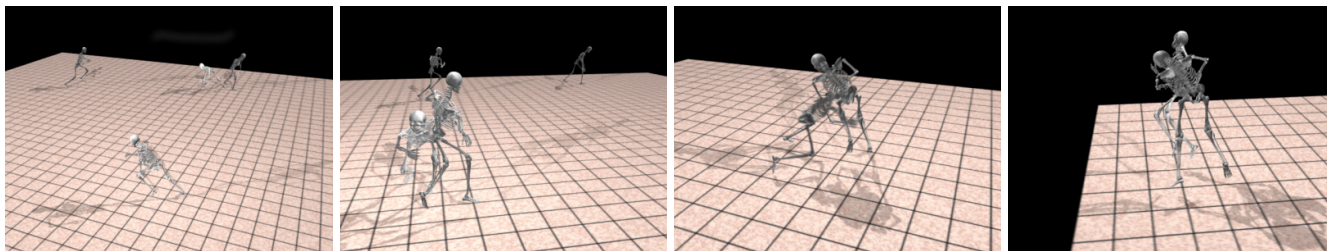


Figure 9: Using combinations of constraints, we can generate multiple motions interacting with each other. Here, the skeletons are constrained to be tackling each other (using hard constraints) at specified times and specified positions/orientations. We can then generate the motion that connects the sequence of tackles.

- FUNGE, J., TU, X., AND TERZOPOULOS, D. 1999. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *Proceedings of SIGGRAPH 1999*, 29–38.
- GALATA, A., JOHNSON, N., AND HOGG, D. 2001. Learning variable length markov models of behaviour. In *Computer Vision and Image Understanding (CVIU) Journal*, vol. 81, 398–413.
- GERSHO, A., AND GRAY, R. 1992. *Vector Quantization and signal compression*. Kluwer Academic Publishers.
- GILKS, W., RICHARDSON, S., AND SPIEGELHALTER, D. 1996. *Markov Chain Monte Carlo in Practice*. Chapman and Hall.
- GLEICHER, M. 1998. Retargetting motion to new characters. In *Proceedings of SIGGRAPH 1998*, vol. 32, 33–42.
- GRZESZCZUK, R., AND TERZOPOULOS, D. 1995. Automated learning of muscle-actuated locomotion through control abstraction. In *Proceedings of SIGGRAPH 1995*, 63–70.
- GRZESZCZUK, R., TERZOPOULOS, D., AND HINTON, G. 1998. Neuroanimator: Fast neural network emulation and control of physics based models. In *Proceedings of SIGGRAPH 1998*, 9–20.
- HEEGER, D. J., AND BERGEN, J. R. 1995. Pyramid-Based texture analysis/synthesis. In *Proceedings of SIGGRAPH 1995*, 229–238.
- HODGINS, J. K., AND POLLARD, N. S. 1997. Adapting simulated behaviors for new characters. In *Proceedings of SIGGRAPH 1997*, vol. 31, 153–162.
- HODGINS, J., WOOTEN, W., BROGAN, D., AND O'BRIEN, J., 1995. *Animated human athletics*.
- KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *Proceedings of SIGGRAPH 2002*.
- LAMOURET, A., AND VAN DE PANNE, M. 1996. Motion synthesis by example. In *Eurographics Computer Animation and Simulation '96*, 199–212.
- LATOMBE, J. P. 1999. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. In *International Journal of Robotics Research*, vol. 18, 1119–1128.
- LEE, J., AND SHIN, S. Y. 1999. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of SIGGRAPH 1999*, 39–48.
- LEE, J., CHAI, J., REITSMA, P., HODGINS, J., AND POLLARD, N. 2002. Interactive control of avatars animated with human motion data. In *Proceedings of SIGGRAPH 2002*.
- LI, Y., WANG, T., AND SHUM, H. Y. 2002. Motion texture: A two-level statistical model for character motion synthesis. In *Proceedings of SIGGRAPH 2002*.
- MATARIC, M. J. 2000. Getting humanoids to move and imitate. In *IEEE Intelligent Systems*, IEEE, 18–24.
- MCMAHON, T. 1984. *Muscles, Reflexes and Locomotion*. PhD thesis, Princeton University Press.
- MOLINA-TANCO, L., AND HILTON, A. 2000. Realistic synthesis of novel human movements from a database of motion capture examples. In *Workshop on Human Motion (HUMO'00)*, 137–142.
- NELSON, R., BROOKS, C., AND N.PIKE. 1977. Biomechanical comparison of male and female distance runners. In *Annals of the NY Academy of Sciences*, vol. 301, 793–807.
- O'ROURKE, J. 1998. *Computational Geometry in C*. Cambridge University Press.
- POPOVIC, Z. 1999. *Motion Transformation by Physically Based Spacetime Optimization*. PhD thesis, Carnegie Mellon University Department of Computer Science.
- PULLEN, K., AND BREGLER, C. 2000. Animating by multi-level sampling. In *Computer Animation 2000*, 36–42. ISBN 0-7695-0683-6.
- PULLEN, K., AND BREGLER, C. 2002. Motion capture assisted animation: Texturing and synthesis. In *Proceedings of SIGGRAPH 2002*.
- ROSE, C., GUENTER, B., BODENHEIMER, B., AND COHEN, M. F. 1996. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of SIGGRAPH 1996*, vol. 30, 147–154.
- SCHODL, A., SZELISKI, R., SALESIN, D., AND ESSA, I. 2000. Video textures. In *Proceedings of SIGGRAPH 2000*, 489–498.
- VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *Proceedings of SIGGRAPH 1997*, vol. 31, 65–76.
- WITKIN, A., AND POPOVIC, Z. 1995. Motion warping. In *Proceedings of SIGGRAPH 1995*, 105–108.