From the ACM SIGGRAPH 2003 conference proceedings. Reprinted with Permission.

Motion Synthesis from Annotations

Okan Arikan David A. Forsyth James F. O'Brien

Abstract

This paper describes a framework that allows a user to synthesize human motion while retaining control of its qualitative properties. The user paints a timeline with annotations — like `walk`, `run` or `jump` — from a vocabulary which is freely chosen by the user. The system then assembles frames from a motion database so that the final motion performs the specified actions at specified times. The motion can also be forced to pass through particular configurations at particular times, and to go to a particular position and orientation. Annotations can be painted positively (for example, *must* `run`), negatively (for example, *may not* run `backwards`) or as a *don't-care*. The system uses a novel search method, based around dynamic programming at several scales, to obtain a solution efficiently so that authoring is interactive. Our results demonstrate that the method can generate smooth, natural-looking motion.

The annotation vocabulary can be chosen to fit the application, and allows specification of composite motions (`run` and `jump` simultaneously, for example). The process requires a collection of motion data that has been annotated with the chosen vocabulary. This paper also describes an effective tool, based around repeated use of support vector machines, that allows a user to annotate a large collection of motions quickly and easily so that they may be used with the synthesis algorithm.

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

Keywords: Motion Synthesis, Human motion, Optimization, Clustering, Animation with Constraints

1 Introduction

The objective of our paper is to provide animators with intuitive controls for synthesizing appealing motions. An ideal model for this system is how a director guides actors and actresses. A similar control can also be used on game characters which can maintain their goals and possible modifiers on how they will attain these goals.

In this paper we present an algorithm that synthesizes motions by allowing the user to specify what actions should occur during the motion as well as specifying modifiers on the actions. These actions and modifiers are represented as annotations that the user paints on a timeline. For example, a motion can be described as "running, picking up and walking while carrying". The annotations that we can use to describe such a motion are `running`, `picking up`, `walking` and `carrying`. The user may also include negative annotations so that the algorithm is prohibited from generating undesired types of actions. Additionally, the user may specify constraints that require the motion to pass through a particular pose or to move to a particular position or orientation at a given time.

Figure 1: In this automatically synthesized motion, the figure is constrained to be tripping, then running, then jumping while still running.

The motion is constructed by cutting pieces of motions from a motion database and assembling them together. The database needs to be annotated before the synthesis. This annotation process is quite flexible: no restrictions on the type of annotation labels are imposed. While annotating appears to be a difficult task, we have produced an annotation process that is quite efficient and easy to use. In our framework the user is required to annotate only a small portion of the database. Our system uses Support Vector Machine (SVM) classifiers to generalize the user annotations to the entire database. The SVM can be interactively guided by the user to correct possible misclassifications. A novice user of our system can annotate the 7 minutes of motion data we have in under an hour using our procedure.

The synthesis algorithm is based on successive dynamic programming optimizations from a representation of the database at a coarse scale to a finer one. The optimization finds blocks of motions that can fit together in a motion sequence and at the same time satisfy the annotations and other low level constraints. The synthesis process is interactive. The user can obtain immediate results and can change the desired motion properties on the fly.

2 Related Work

Recent years have seen a number of algorithms for motion synthesis from motion data. These algorithms create novel motions by cutting pieces from a motion database and reassembling them to form a new motion. These algorithms [Arikan and Forsyth 2002; Lee et al. 2002; Li et al. 2002; Pullen and Bregler 2002; Molina-Tanco

Email{okan,daf,job}@cs.berkeley.edu

and Hilton 2000] can synthesize motions that: follow a path, go to a particular position/orientation, perform a particular activity at a specified time or simulate certain characteristics in the video sequence of a real person. However, relatively limited direction of the motion is possible. For example, there are many ways to follow a specified path and a user may wish to specify how the character will move along the path (e.g., run, walk) or what other actions to take (e.g., wave) while following the path. Motion graphs [Kovar et al. 2002a] attack this problem by confining their search to subgraphs induced by the desired action. However, this method is ill suited if the desired actions have short temporal span, such as "jumping" or "catching" or if the actions are to be composed: "jump and catch while running".

Local search methods for motion synthesis have problems synthesizing motions which require global planning. For example, to be able to jump at a particular point in time, one may need to prepare well in advance. Motion graphs do not allow such long term motion planning. Although the method proposed in [Arikan and Forsyth 2002] looks for a solution to a global optimization problem, it does so by making local changes to the solution. These local changes will break the synchronization that aligns the generated motion with the user's annotations. As a result, a single local mutation will almost invariably generate a poorer solution and will be rejected by the algorithm.

[Blumberg and Galyean 1995] produces controllers that convert behavioral goals or explicit directions into motion. Designing controllers that lead to a natural looking motion is quite difficult, particularly for high level actions such as walking, jumping and running. Possible strategies include imitation [Mataric 2000] or optimization [Grzeszczuk and Terzopoulos 1995]. Scripting is another alternative for control [Perlin and Goldberg 1996], but often has the disadvantage of providing too detailed control over the motion.

[Rose et al. 1998] construct a verb graph by hand where each verb is represented as a linear combination of adverb motions. However, applying this method to large databases with large number of verbs and adverbs poses problems. Interpolating different executions of the same action can also lead to undesirable motions, especially if the interpolated motions are not similar.

Physically based methods can be used to rectify problems that are created by either transitioning [Rose et al. 1996] or interpolating between motions. Physically based methods can also synthesize motions from scratch [Witkin and Kass 1988; Liu and Popovic 2002; Hodgins et al. 1995; Faloutsos et al. 2001], but quickly become too complex for interesting human motions. We believe data driven algorithms, including ours, will benefit from using physically based models, for example, to rectify discontinuities in the synthesized motions.

Starting from an already captured or synthesized motion, different algorithms can be used to fix feet - ground interactions [Kovar et al. 2002b], retarget motion to a character with different body proportions [Gleicher 1998], introduce stylistic attributes such as "happy" or "sad" [Brand and Hertzmann 2000], or warp the motion to satisfy positional constraints [Witkin and Popovic 1995; Gleicher 2001]. An interesting way of capturing motion from cartoons has also been presented by [Bregler et al. 2002].

3 Synthesis

Our objective is to control a human figure. We would like to do this by painting actions (such as stand, walk, run etc.) and modifiers (such as reach, catch, carry etc.) on the timeline. The synthesis algorithm should then create a motion that performs these actions at the right times while having natural looking transitions between actions so that the final motion looks human.

Given an annotated timeline specifying what kinds of motion should occur at particular times, our method automatically assembles subsequences from a motion database into a smooth motion that matches the annotations. We will call the annotations that must be satisfied **annotation constraints**. An example set of annotations would be to run slowly for the first 200 frames then switch to walking for another 100 frames while waving the entire time. The main focus of this paper is to synthesize such motions efficiently. The user also expects the motion to be of a particular length (**length constraint**) and to be continuous (**continuity constraints**). A motion is continuous if it looks natural at all cuts between motions. We may still require the **frame constraints** and the **position constraints** of earlier work [Arikan and Forsyth 2002; Kovar et al. 2002a; Lee et al. 2002]. Recall that a frame constraint will ensure that at a particular time the motion will pass through a particular frame selected from the database. Position constraints ensure that the motion ends at a particular position and orientation. The synthesis process should choose frames from a motion database such that the chosen frames, when put together, are continuous and match the desired set of annotations as well as any possible frame and position constraints.

It is natural to allow annotations to be composed, meaning that there could be a very large set of possible annotations. In practice, however, the database may not contain a continuous set of frames that satisfies every possible combination of these annotations. Furthermore, some annotations may be fundamentally incompatible with each other. For example, one cannot expect to find a motion that `stands` while `running` even though, individually, these two are perfectly reasonable annotations. Another consideration is that annotations can not exactly match the desired motions: we can not find a continuous motion that `runs` for the first 100 frames and then suddenly `walks` for the next 100. Thus in section 3.2, the problem will be formulated as a combinatorial optimization which tries to choose frames so that the motion is continuous and closely matches to the annotations, and a Dynamic Programming (DP, e.g., [Bertsekas 2000]) based solution will be introduced. However, since the solution method will assume every motion in the motion database has already been annotated, section 3.1 will first provide a description of our annotation process.

3.1 Annotating Motions

Annotations are used to describe motions. In order to accommodate a variety of motions, the annotations should be flexible. In other words, the user should be able to have an arbitrary vocabulary that is suited to the motions being annotated. The vocabulary chosen for the annotations defines the level of control of the synthesis: the user can annotate `left foot on the ground` and `right foot on the ground`, but this does not provide an intuitive control for the synthesis unless the user wants a particular foot to be on the ground at a particular time. Although the framework we present in this paper handles such detailed annotations equally well, we choose to focus on annotations that describe the qualitative properties of motion.

The database of motions that we used for our examples consisted of 7 minutes of American football motions. The vocabulary that we chose to annotate this database consisted of: `Run`, `Walk`, `Wave`, `Jump`, `Turn Left`, `Turn Right`, `Catch`, `Reach`, `Carry`, `Backwards`, `Crouch`, `Stand`, and `Pick up`. Some of these annotations can co-occur: `turn left` while `walking`, or `catch` while `jumping` and `running`. Any combination of annotations is allowed, though some combinations may not be used in practice. For example, we cannot conceive of a motion that should be annotated with both `stand` and `run`.

Our annotation vocabulary reflects our database. A different choice of vocabulary would be appropriate for different collections.

Furthermore, the annotations are not required to be canonical. Our algorithm should be equally happy synthesizing dance sequences (with annotation terms like `plié`) and character sketches (with annotation terms like `happy` or `tired`). We have verified that a consistent set of annotations to describe a motion set can be picked by asking people outside our research group to annotate the same database. This implies the annotator does not have to be the same as the animator.

Once a vocabulary is chosen, all the motions in the database must be annotated. For example, if a motion `runs` for the first 100 frames of a motion and then switches to `walking` for the next 100, the first hundred frames must be annotated with `running` and the last 100 must be annotated with `walking`. This process is inherently ambiguous temporally: we can not find an exact frame where we switch from running to walking. But since the synthesis algorithm is formulated as an optimization, a rough set of annotations proved adequate. Even so, annotating a large set of motions by hand would be a difficult and tedious process. Ideally, we would like to annotate only a few examples by hand and then let the system generalize our annotations to the rest of the database. In order to make the annotation process faster, we built a Support Vector Machine (SVM) classifier.

Each annotation, when considered separately, partitions frames that have been annotated into two groups: frames performing the action (group 1) and frames that do not (group -1). Given a new motion, its frames are classified into either group 1 or group -1 using an SVM. To make this classification, every frame needs to be embedded in a suitable coordinate system. The coordinate vector we choose for each frame is the joint positions for one second of motion centered at the frame being classified. Since the motion is sampled in time, each joint has a discrete 3D trajectory in space for the second of motion centered at the frame. The embedding of a frame is simply a vector of joint positions. In order to make sure these 3D positions are consistent and comparable, they are represented in the torso coordinate system of the frame being classified. In this coordinate system, the frame being classified is at the origin and the up, left and forward directions are aligned with 3D coordinate axes.

We can separate new frames into two groups quite easily, using a radial basis function $\langle f_1, f_2 \rangle = \exp - \frac{|f_1 - f_2|}{\gamma}$ as the kernel and by choosing a γ (which controls the curvature of the decision boundary) that achieves the best looking classification results. For each annotation, we train a separate classifier using the motions that have already been annotated. When a new motion is to be annotated, we classify its frames using the SVM for each annotation label. We then display the annotation results to the user who can make corrections if necessary. The user verified data is then added to the SVM training set and the classifier's decision boundary is re-optimized. It is our experience that after annotating 3-4 variants of an annotation, the user rarely needs to correct the auto-annotation results. This way the user simply verifies the automatic annotation results and makes corrections only if necessary. This on-line learning method makes is possible to annotate large databases of motions quite rapidly.

In our implementation, we used a public domain SVM library (`libsvm` [Chang and Lin 2000]). The out of margin cost for the SVM is kept high to force a good fit within the capabilities of the basis function approximation.

3.2 Optimization

Using the results of the previous section, we assume all the motions in the motion database have been previously annotated with a fixed set of annotations. Let $A(f)$ represent the annotation vector for frame f in the motion database. If there are m different kinds of annotations, $A(f)[1 \cdots m] \in \{1, -1\}$ where the k'th element of $A(f)$ is 1 if this frame has the k'th annotation (and -1 if it doesn't). For example, if the first annotation is `running` and the second annotations is `jumping`, frames belonging to a running motion will have the first item of their annotation vectors set to 1. For frames where the figure is jumping in addition to running (i.e., a running jump), the second item will also be 1.

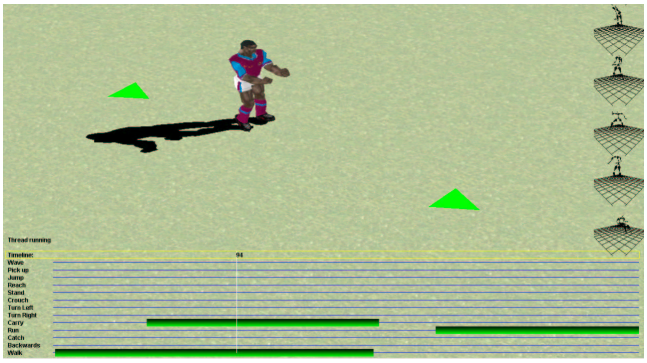

Figure 2: The user interface allows the user to see each available annotation label (bottom of the screen), and paint positive annotations (green bars) and negative annotations (blue bars). The frames that are not painted are interpreted as *don't care*. The user can manipulate geometric constraints directly using the green triangles and place frame constraints on the timeline by choosing motion to be performed (right of the screen).

The output motion is a sequence of frames chosen from the motion database such that when we put the chosen frames together to form the motion, the subsequent frames are continuous and satisfy the annotation constraints. If the frames in the database are represented as $f_1 \cdots f_T$, we need to choose $f_{\sigma_1} \cdots f_{\sigma_n}$ where $\sigma_i \in [1 \cdots T]$ is the frame to use as the i'th frame of the synthesized motion. Here T represents the total number of frames of motion in the database and n is the number of frames of motion to synthesize. The desired motion is represented as a solution to the following objective function:

$$\min_{\sigma_1 \cdots \sigma_n} \left[\alpha \sum_{i=1}^{n} D(i, A(f_{\sigma_i})) + (1-\alpha) \sum_{i=1}^{n-1} C(f_{\sigma_i}, f_{\sigma_{i+1}}) \right] \qquad (1)$$

In this equation, functions D and C evaluate how well the frame's annotations match the desired set of annotations at a particular frame and how well the subsequent frames match each other respectively. The α parameter can be used to favor motions that are more continuous or motions that match the annotation better.

$D(i, A(f))$ compares the annotations of frame f in the motion database versus the desired set of annotations that we would like to have at frame i of the motion to synthesize. We represent the desired set of annotations for the frame in the vector $Q(i)$ which is defined the same as $A(f)$. Since both quantities are vectors of the same length, one possibility is the squared difference. However, this assumes that the user has marked the entire timeline with the annotations that he/she wants. Most of the time though, we want a particular subset of annotations to be performed at certain times and we want a certain subset of annotations not to be performed while not caring about the rest of the annotations. This suggests an alternative form. For each annotation, we can say "yes, have the annotation" (1), "no, do not have the annotation" (-1) or "we do not care" (0). If that annotation should be on and the frame has that annotation, we reward it by making its score better, whereas if the annotation is to be off and if the frame has the annotation,

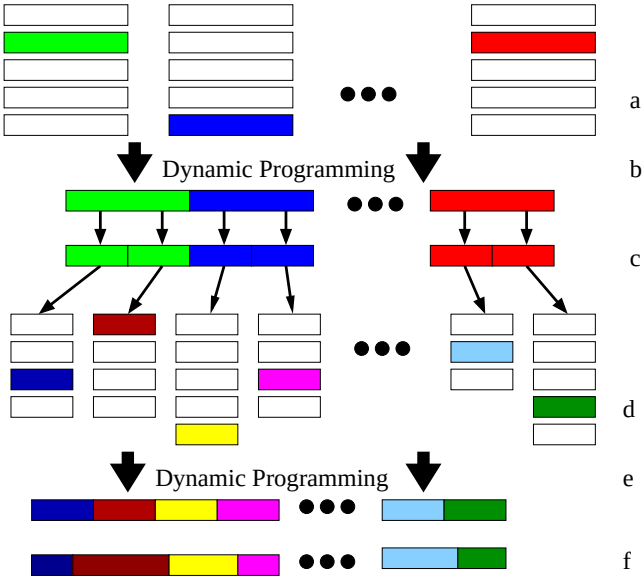

Figure 3: The motion is constructed out of blocks of subsequent frames from a motion database (see section 3.3). (a) 32 frame blocks of motions are selected at random with stratification to be the active set for each time slot that needs to be synthesized. (b) Dynamic programming is used to choose an optimal sequence of 32 frame blocks. (c) The resulting motion is broken into 16 frame blocks. (d) Other 16 frame blocks that are similar to the previous result are used as the active set. (e) Dynamic programming is used to find a finer solution. The search can be repeated this way until we reach down to individual frame level. In practice, we stop at 8 frame blocks. (f) The final solution is passed through a local optimizer that moves the block boundaries to achieve most continuous motion

we make its score worse. Let $Q(i) \in \{-1,0,1\}$ designate the user's annotation preferences and define:

$$D(i,A(f)) = -\sum_{j=1}^{m} Q(i)[j] \times A(f)[j] \quad (2)$$

To be able to compute this function, we allow the user to mark the timeline with annotations to be performed or annotations not to be performed. Unmarked frames are interpreted as "don't care" (see figure 2). The smaller values of D indicate a better match to the desired set of annotations, which is consistent with equation 1.

$C(f_i, f_j)$ computes the goodness in terms of continuity of putting frame f_j after frame f_i. This is essentially the distance between frames f_{i+1} and f_j as in [Arikan and Forsyth 2002; Kovar et al. 2002a; Lee et al. 2002]. Since keeping the distance between every pair of frames in the database is not practical due to $O(T^2)$ storage cost, we compute feature vectors for each frame and take the squared distance between the feature vectors for frames f_{i+1} and f_j. The feature vector for a frame is the joint position, velocities and accelerations for every joint expressed in the torso coordinate frame. Since the dimensionality of these vectors are likely to be large (a 30 joint skeleton would contain 90×3 numbers), we project the dimensionality down to 20 using principal component analysis without much loss in the signal. If the feature vector for frame f is $F(f)$, then $C(f_i, f_j) = ||F(f_{i+1}) - F(f_j)||$.

An important observation is that the objective function in equation 1 is composed of "local" terms[1] that check the goodness of a frame as a function of the immediate neighbors only. Thus, the problem can be solved using DP by using the following cost-to-go function:

$$J(i, f_j) = \min_{\sigma} \left[D(i,A(f_j)) + C(f_\sigma, f_j) + J(i-1, f_\sigma) \right] \quad (3)$$

$$J(1, f_j) = D(1,A(f_j)) \quad (4)$$

The computational cost of DP is $O(n \times T^2)$ (see figure 3). This means DP can not be applied directly even for small databases. The following section describes a hierarchical search algorithm that provides a close approximation while being practical.

3.3 Practical Algorithm

In this section, we describe an algorithm that synthesizes motions by first creating a motion out of big blocks of frames. This initial solution will capture the essential structure of the motion. We then refine this motion by performing a search that operates on smaller blocks. The idea of this refinement is to improve the look of the motion while benefiting from the restriction in the search space provided by the structure of the coarse motion.

The crucial observation is that most of the time, annotations have long temporal support: one usually does not `run` for just one frame. Thus, it is natural to think in terms of blocks of frames. Thus, we perform DP on sequences of 32 frame blocks (about half a second long). This size is about the granularity of our annotations. The total number of 32 frame blocks is still $O(T)$. If we look at all 32 frame blocks, many of them will be very similar to each other. For example, if there are 10 running motions in the database, we will have many copies of the same 32 frame blocks. In order to deal with this issue, we cluster all 32 frame blocks and work on representative blocks from each cluster.

In order to synthesize n frames of motion, we have $\lceil \frac{n}{32} \rceil$ time slots. For each slot, a 32 frame motion sequence needs to be found. In order to find these sequences, each slot should have an active set of possible 32 frame sequences that can go in that block. Since the computational cost of DP is linear in the number of slots and quadratic in the number of sequences in each active set, we would like to have a small number of relatively different sequences so that the solution we get is close to the true global optimum if we had used all 32 frame sequences in our active sets. To get such representative sequences, we take all 32 frame sequences in the motion database and cluster them into 100 clusters.

The number of clusters that we need is found experimentally. 100 clusters is small enough to make DP interactive and large enough to have sufficient variety. The active set for each time slot consists of 100 random sequences, drawn one from each cluster (figure 3-a). This creates a stratified sample of motion blocks which we can search. In order to perform clustering, 32 frame blocks must be embedded in a coordinate system. We compute the coordinate of a 32 frame block by augmenting the feature vectors of the 32 frames into one big vector. We then perform k-means clustering on these augmented vectors to find 100 clusters of 32 frame blocks.

The result of the search using 32 frame sequences is a rough solution because the search did not see the entire motion database and we could only cut between sequences along integer multiples of 32 (figure 3-b). The result of this step is refined by doing another DP on 16 frame sequences. At this step, we would like to find a better motion that has the general structure of the 32 frame solution. Since the 32 frame block DP operated on quite different types of blocks, it can enumerate different motions and capture the general structure of the desired motion quite well. Thus, the 16 frame block solution should improve it in terms of continuity and annotation matching

[1]The D and C functions are inherently local as they measure goodness of individual frames.

without changing the structure of the motion. The active sets of 16 frame sequences are chosen to be those that are "near" the previous solution at the time of the slot (figure 3-c,d). In order to find what is close to a given block of 16 frames, we use clustering again. This time we cluster all sequences of 16 frame blocks. The active set for a block is then all the blocks that are in the same cluster as the 16 frame block of the parent solution. Since the number of 16 frame blocks that we find will be the active set for the next level of search, we would like to have about 100 blocks per cluster which is enforced by clustering into $\frac{T}{100}$ clusters.

One can go down to the individual frame level by doing successive DP, using the result of the previous one to choose the active sets for the next search (figure 3-e). However, in practice, we do only 3 searches and stop at 8 frame sequences. After we go down to the 8 frame level, we create and save the motion. Then we go back and restart the search from the top using another random selection of 32 frame blocks from each cluster. The search started with a random selection of relatively different types of blocks tends to explore the space of motions quite well whereas the lower level searches fine tune the solution found in the previous level. This way, at each iteration, we generate a new motion which should be better than the previous one.

Frame constraints can easily be incorporated into this search procedure. Every time slot during the dynamic programming has an active set of motion blocks where any one of these blocks can be used at that slot. However, we can force a specific frame in the database to occur at a particular time by making the active set for the slot containing that frame consist of only one block: the block that has the desired frame or motion. Since this reduces the number of blocks in the active set for the constrained block to one, frame constraints make the search easier by constraining the search space.

3.4 Position Constraints

During the search, we would like to assert geometric constraints such as "be here" at the end of the motion [Arikan and Forsyth 2002]. The search process described above is iterative: it generates a new motion at each iteration. Even though the motions that it generates may not satisfy position constraints, combinations of these motions may. For example, by taking the first 100 frames of one motion and the last 100 frames of another motion, we may be able to create motions that go to a particular position and orientation. Enumerating all possible cuts between motions that have been generated so far is very costly. The motions are generated in 8 frame blocks, so we can enumerate cuts at the 8 frame block granularity (i.e., we enumerate cuts between integer multiples of 8 frame blocks). Furthermore, instead of enumerating all cuts between all pairs of motions that have been generated, we only enumerate cuts between the motion that has been generated at this iteration and the best 20 motions that have been generated in the previous iterations. This gives us enough variety in terms of the end position/orientation while making the search tractable.

For a given pair of motions and a given cut location, such as after block 4, we can compute the end position/orientation of the motion that had its first 4 blocks from the first one and the remaining blocks from the second one in $O(1)$ time. This involves caching, for every block of every motion, the position and orientation of the body at the ending frame of the block relative to the beginning of the motion.

Whenever a new motion is generated at each iteration, it has an associated score that comes from DP which optimizes for matching the annotations and the continuity of the motion. If there are positional constraints on the motion, we add to this score how close the motion gets to the desired position. For each of the current 20 best motions, we look for a single cut between the motion generated at the last iteration and that motion. If the score of the best motion

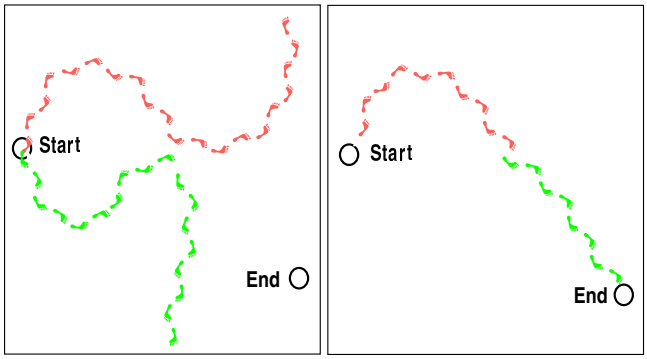

Figure 4: Left- Two motions in the pool of synthesized motions for positional constraints, neither going to where we want it to go. Right- The final motion that is created by putting the end of one motion after the beginning of the other motion achieves the positional constraints.

obtained like this is better than the score of the best motion we had so far, we replace the current solution with that one and put the new motion back into the pool of best 20 motions synthesized.

The motions that are generated at each iteration are quite good in terms of their continuity and they are often quite different motions. Making any cut to any other motion creates a visible discontinuity. Thus we keep a pool of the best 20 motions for each level of the search: for 32, 16 and 8 frame blocks separately. Whenever, a level generates a solution, it is inserted into the level's pool as described above. A solution can remain untouched if it already gets close to the target state. However, if it does not get to the target state, it is likely to be replaced by another that will get closer to the target. Making these position constraint enforcements after every level coerces motions towards the target state. Since the lower levels refine the continuity of the motion, the result is better.

If there is no position constraint on the motion, we do not keep a pool of motions at every search level. Instead, a motion generated in a level is used to find the active sets of motion blocks for the next level and is then discarded. Whenever we generate a new solution at the 8 frame level that is better than the best one we have, we create the motion (see section 4) and start displaying it to the user.

If it is not possible to meet the position constraints while being continuous and satisfying annotations, the search algorithm will be biased towards one of these terms as a function of their weights in the final score computation. By displaying the current best motion to the user, he/she can see if this is happening and change the constraints interactively (see figure 2).

4 Creating the Motion

The final motion is constructed out of 8 frame sequences each of which may come from a different motion. Thus, it may be discontinuous at the block boundaries. Each block has a motion number which identifies the motion that the block comes from, an entry frame number and an exit frame number (which is entry+8 at the beginning). The discontinuity problem is alleviated by passing the motion through a local minimization step that tweaks the entry/exit frames of these blocks to attain the maximum continuity. This can be done by looking at pairs of subsequent blocks. At the boundary point, we switch from frame f_i of motion m_1 to frame f_j of motion m_2. The objective at this step is to decrease the distance $E = (F(f_{i+1}) - F(f_j))^2$ where $F(f)$ is the feature vector for frame f. The feature space provides a comparison medium for two frames: if features for two frames are close in the feature space, they are similar. We minimize the function E with respect to f_i and

f_j using gradient descent (see figure 3-f). We do this optimization for every pair of blocks in the synthesized motion. Even though this step can change the number of frames used in the motion or even worse, move the frame we exit a block before the entry frame, such problems rarely occur. This is because the result of the DP is usually so good that the gradient descent makes 1-2 frame shifts at the block boundaries and the length of the motion stays constant.

After this step, we are left with a sequence of motions and pointers to the entry and exit frames. We take the corresponding frames from each motion and put them together to form a single motion sequence. We also perform a smoothing as in [Arikan and Forsyth 2002] to get rid of offensive small discontinuities.

5 Discussion

Combinatorial explosion makes dynamic programming on the individual frame level entirely intractable. However, we believe the approximation provided by starting at 32 frame blocks and refining the solution is reasonable. Starting the dynamic programming at 16 or 8 frame blocks does not improve the solution substantially in practice, but makes the search process significantly slower. Using a longer initial block creates bigger quantization error for matching the annotations and leads to slower convergence.

The search time is $O(k \times (n^2 + m^2))$ where k is the number of blocks to search (proportional to the desired motion length), n is the number of top level clusters and m is the number of motions per cluster ($m \times n \cong T$). The term n^2 above comes from the initial top level search, and the m^2 term comes from the refinement steps. Thus for $n \approx m$, the search time is linear in the number of frames to synthesize and the number of frames in the database.

The major limitation of the algorithm is its inability to synthesize frames that do not occur in the database. For example, if the database does not contain an instance of `running` and `jumping`, the algorithm will not be able to synthesize a `running jump`. This means the annotation vocabulary must match the database. If chosen annotation labels do not happen in the database, the algorithm will not be able to synthesize matching motions. In such a case, the algorithm may synthesize discontinuous motions if the weight of the motion continuity is small compared to the annotation matching weight (controlled by α). The motion smoothing mechanism can then move the entry and exit frames of the joining blocks substantially to make them join up better. This in turn can change the length of the synthesized motion substantially. However, with a suitable selection of the vocabulary and example motions for each annotation in the dataset, this problem does not happen.

Since we do not have an explicitly computed motion graph where possible connections between motions are enumerated, the search algorithm can put any sequence of frames from one motion after any sequence of frames from another motion. If the end of one sequence does not look like the beginning of the subsequent sequence, the continuity score will be bad, forcing the search to find another arrangement of frames. However, if the user asks for a `walking` and then `running` motion and if the motion dataset does not contain any transition motions from walking to running, the search will fail to find such an alternative arrangement that is continuous. This means that if the user asks for motions that do not happen naturally or do not occur in the database, the search will either omit the annotations and will stay continuous or will satisfy the annotations by a discontinuous motion. The search can be guided to doing either one by changing the influence (α) of the continuity score and the annotation score.

The interactive search makes it possible to get a sense of what kinds of motions can be synthesized from the database. The user can see different kinds of motions that are being obtained after each iteration for a given set of annotations. If the annotations are incompatible and the search is unable to find a desirable motion, the user gets direct feedback and can paint the timeline differently to help the search by clarifying the desired annotations.

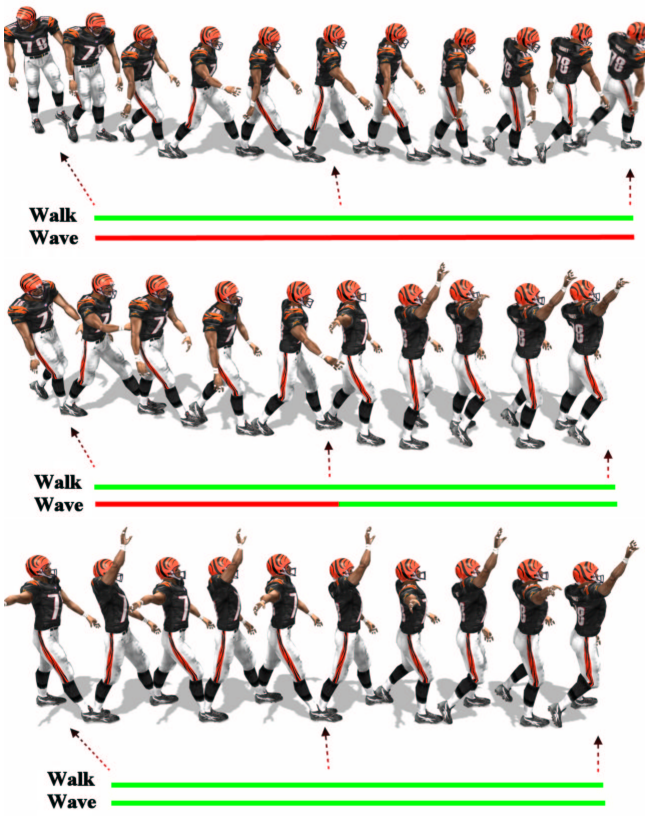

Figure 5: The framework can synthesize motions that match a given set of annotations marked on the timeline. For example, the top figure shows a synthesized motion for walking but not waving. The middle picture is synthesized for walking but waving only for the second half of the motion. The bottom motion is synthesized for walking and waving.

6 Results

We presented an interactive motion synthesis algorithm where we can control qualitative properties of the synthesized motion as well as details. The synthesis process is easy to interact with and generates desired motions quickly. The user can specify what kinds of motions are to be performed at what times and what kind of motions are not to be performed (see figure 5). While the main contribution of the paper is synthesizing motions that match given annotations, the user can also enforce low level constraints. The user can force the synthesized figure to have a particular pose or motion (in the database) at a particular frame (see figures 6 and 1). The algorithm can also synthesize motions that go to a specific position and orientation (see figure 7). As the accompanying video demonstrates, motions synthesized with our system meet the annotations and look human. This means that our continuity score is effective and the search is successful.

The user interface for synthesizing motion is quite easy to use and the synthesis process is interactive. The user can get immediate feedback on the search process and change constraints on the fly. The iterative nature of the search also means as the user waits longer, better motions are generated.

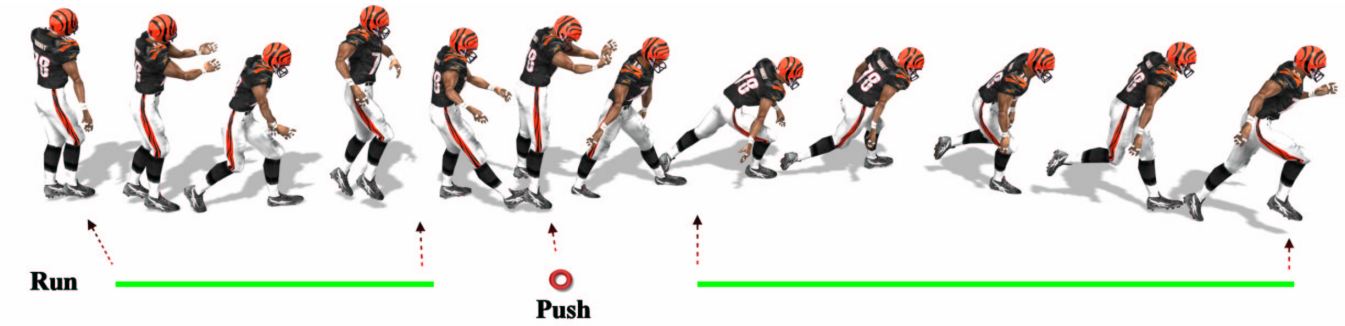

Figure 6: In addition to matching the annotations, a specific frame or motion can be forced to be used at a specific time. Here, the person is forced to pass through a pushing frame in the middle of the motion while running before and after the pushing.

Figure 7: The search can also take positional constraints into account while synthesizing motions for given annotations. Here, the figure is constrained to be running forward and then running backwards. We enforce position constraints indicated as green arrows. For clarity, the running forwards section of the motion is shown on top while running backwards is shown on the bottom.

7 Acknowledgments

We thank the other members of the Berkeley Graphics Group for their helpful criticism and comments. This research was supported by Office of Naval Research grant N00014-01-1-0890 as part of the MURI program, NFS grant CCR-0204377, and State of California MICRO grant 02-055. We would like to thank Electronic Arts for supplying us with the motion data, as well as Sony Computer Entertainment America, Intel Corporation, and Pixar Animation Studios for providing additional funding.

References

ARIKAN, O., AND FORSYTH, D. 2002. Interactive motion generation from examples. In *Proceedings of SIGGRAPH 2002*, 483–490.

BERTSEKAS, D. P. 2000. *Dynamic Programming and Optimal Control*. Athena Scientific.

BLUMBERG, B., AND GALYEAN, T. 1995. Multi-level direction of autonomous creatures for real-time virtual environments. In *Proceedings of SIGGRAPH 1995*, 47–54.

BRAND, M., AND HERTZMANN, A. 2000. Style machines. In *Proceedings of SIGGRAPH 2000*, 183–192.

BREGLER, C., LOEB, L., CHUANG, E., AND DESHPANDE, H. 2002. Turning to masters: Motion capturing cartoons. In *Proceedings of SIGGRAPH 2002*, 399–407.

CHANG, C. C., AND LIN, C. J. 2000. Libsvm: Introduction and benchmarks. Tech. rep., Department of Computer Science and Information Engineering, National Taiwan University.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composable controllers for physics-based character animation. In *Proceedings of SIGGRAPH 2001*, 251–260.

GLEICHER, M. 1998. Retargetting motion to new characters. In *Proceedings of SIGGRAPH 1998*, 33–42.

GLEICHER, M. 2001. Motion path editing. In *Proceedings of 2001 ACM Symposium on Interactive 3D Graphics*, 195–202.

GRZESZCZUK, R., AND TERZOPOULOS, D. 1995. Automated learning of muscle-actuated locomotion through control abstraction. In *Proceedings of SIGGRAPH 1995*, 63–70.

HODGINS, J., WOOTEN, W., BROGAN, D., AND O'BRIEN, J. 1995. Animated human athletics. In *Proceedings of SIGGRAPH 1995*, 71–78.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *Proceedings of SIGGRAPH 2002*, 473–482.

KOVAR, L., GLEICHER, M., AND SCHREINER, J. 2002. Footstake cleanup for motion capture editing. In *ACM SIGGRAPH Symposium on Computer Animation 2002*, 97–104.

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*, 491–500.

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*, 465–472.

LIU, C. K., AND POPOVIC, Z. 2002. Synthesis of complex dynamic character motion from simple animations. In *Proceedings of SIGGRAPH 2002*, 408–416.

MATARIC, M. J. 2000. Getting humanoids to move and imitate. In *IEEE Intelligent Systems*, IEEE, 18–24.

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.

PERLIN, K., AND GOLDBERG, A. 1996. Improv: A system for scripting interactive actors in virtual worlds. In *Proceedings of SIGGRAPH 1996*, 205–216.

PULLEN, K., AND BREGLER, C. 2002. Motion capture assisted animation: Texturing and synthesis. In *Proceedings of SIGGRAPH 2002*, 501–508.

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.

ROSE, C., COHEN, M. F., AND BODENHEIMER, B. 1998. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications 18*, 5 (/), 32–41.

WITKIN, A., AND KASS, M. 1988. Spacetime constraints. In *Proceedings of SIGGRAPH 1988*, 159–168.

WITKIN, A., AND POPOVIC, Z. 1995. Motion warping. In *Proceedings of SIGGRAPH 1995*, 105–108.

From the ACM SIGGRAPH 2002 conference proceedings. Reprinted with Permission.

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 $fromMotion(e) = s$, $toMotion(e) = t$, $fromFrame(e) = i$, $toFrame(e) = j$ and $cost(e)$ be the cost associated with the edge (defined in Appendix A). In this setting, any sequence of edges $e_1 \cdots e_n$ 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).

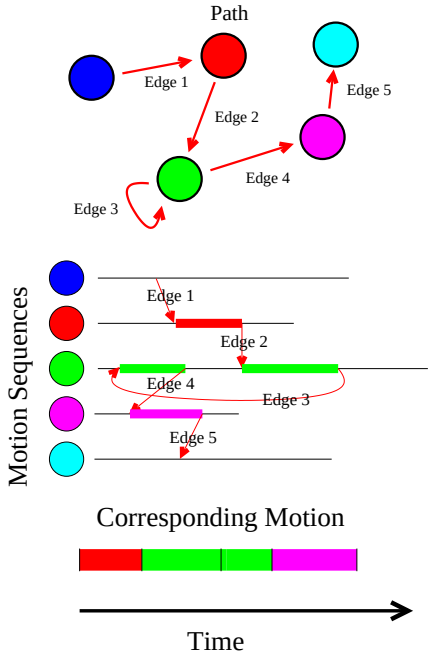

Figure 1: We wish to synthesize human motions by splicing together pieces of existing motion capture data. This can be done by representing the collection of motion sequences by a directed graph (**top**). Each sequence becomes a node; there is an edge between nodes for every frame in one sequence that 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.

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 $G' \leftarrow G'' \leftarrow G''' \leftarrow \cdots \leftarrow G^n \leftarrow G$ 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 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

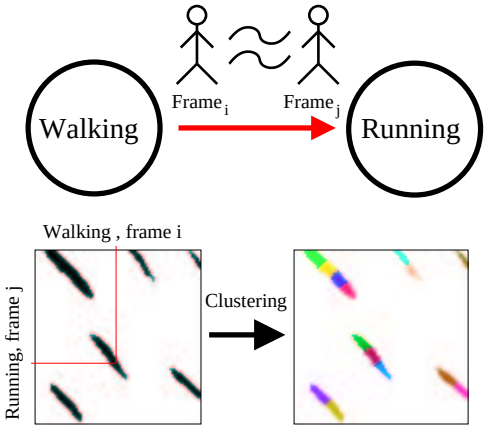

Figure 2: Every edge between two nodes representing different motion clips can be represented as a matrix where the entries correspond to edges. Typically, if there is one edge between two nodes in our graph, there will be several, because if it is legal to cut from one frame in the first sequence to 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.

of G from the P matrices (defined between every pair of nodes) using k-means [Bishop 1995]. The number of clusters is chosen as $\frac{majoraxislength}{minoraxislength}$ 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 $G' \leftarrow G'' \leftarrow G''' \leftarrow \cdots \leftarrow G^n \leftarrow G$ 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 $G' \cdots G^n$ 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 $e_1 \cdots e_n$, 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:

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

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.

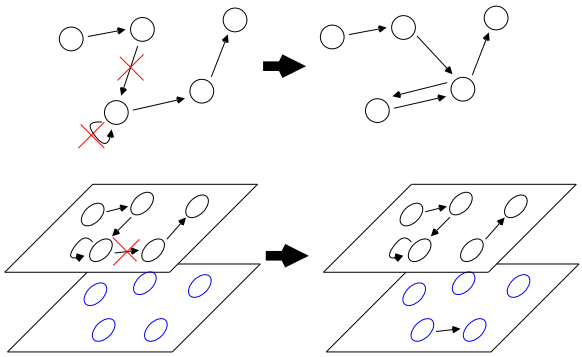

Figure 3: The two mutations are: deleting some portion of the path (top-left, crossed out in red) and replacing that part with another set of edges (top-right), and deleting some edges in the path (bottom-left) and replacing deleted edges with their children in our hierarchy (bottom-right)

4.2.2 Mutating a Path

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

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

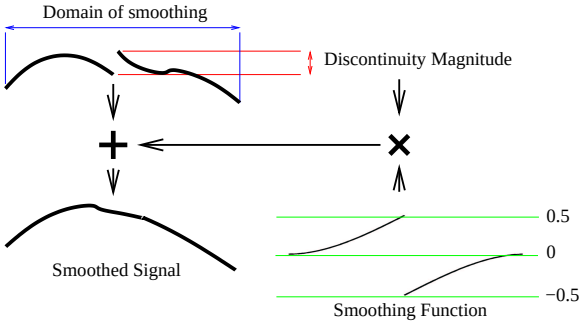

Figure 4: In the synthesized motion, discontinuities in orientation are inevitable. We deal with these discontinuities using a form of localized smoothing. At the top left, a discontinuous orientation signal, with its discontinuity shown at the top right. We now construct an interpolant to this discontinuity, 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.

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 that 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 ± 30 frames (or one second of animation) around the discontinuity and

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

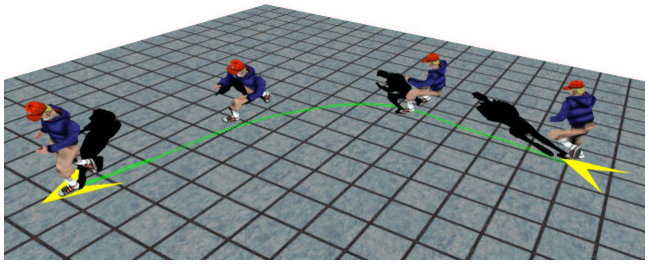

Figure 5: Body constraints allow us to put "checkpoints" on the motion: in the figure, the arrow on the right denotes the required starting position and orientation and the arrow on the left is the required ending position and orientation. All constraints are also time stamped forcing the body to be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

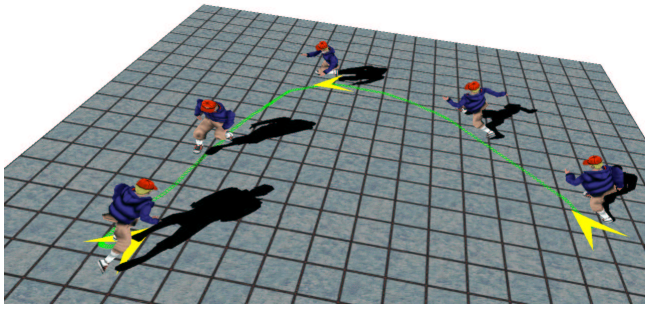

Figure 6: We can use multiple "checkpoints" in a motion. In this figure, the motion is required to pass through the arrow (body constraint) in the middle on the way from the right arrow to the left.

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 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 (fig-

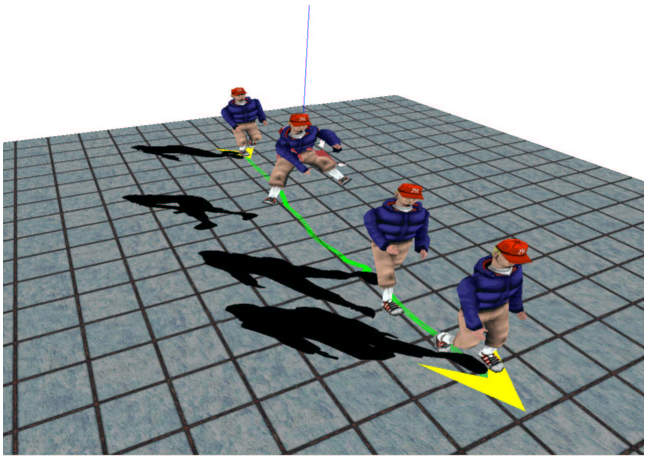

Figure 7: In addition to body constraints, joint constraints can be used to further assign "checkpoints" to individual joints. In this figure, the head of the figure is also constrained to be high (indicated by the blue line), leading to a jumping motion.

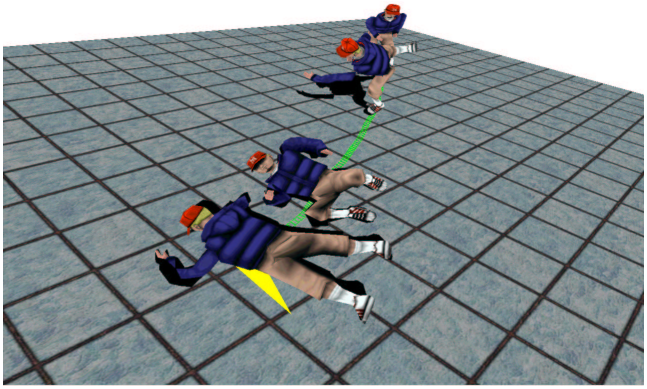

Figure 8: Using hard constraints, we can force the figure to perform specific activities. Here, we constrain the end of the motion to be lying flat on the ground at a particular position/orientation and time. Our framework generates the required tipping and tumbling motion in real-time.

ure 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 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 τ (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 applicable 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 $T(s_i) + \widehat{R(s_i)} \cdot p$, where $T(s_i)$ is the 3×1 translation of the torso and $R(s_i)$ is the 3×1 rotation of the torso and $\widehat{R(s_i)}$ 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.

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

$$L = max_{s,i}(|D'^{\mathsf{T}}_{s_i,s_i} D'_{s_i,s_{i+1}}|) \quad (4)$$

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

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

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 $cost(e) = C(s_i,t_j)$. τ 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 multibody 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.

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.

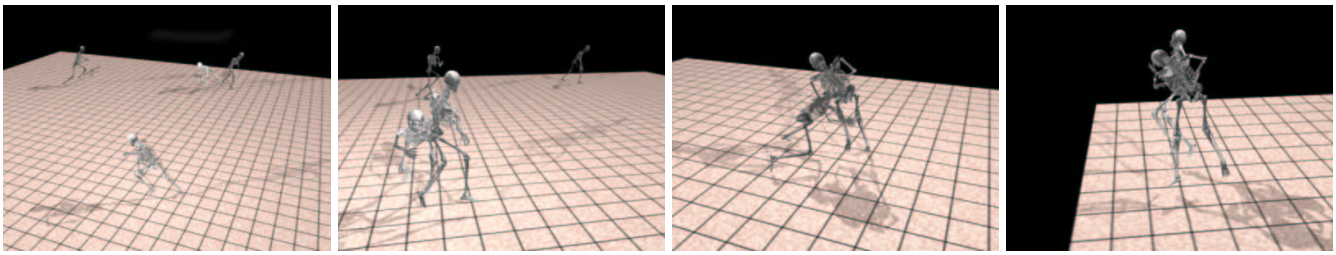

Figure 9: Using combinations of constraints, we can generate multiple motions interacting with each other. Here, the skeletons are constrained to be tackling each other (using hard constraints) at specified times and specified positions/orientations. We can then generate the motion that connects the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

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-likefigures. 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.