Motion Synthesis from Annotations

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

University of California, Berkeley

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.

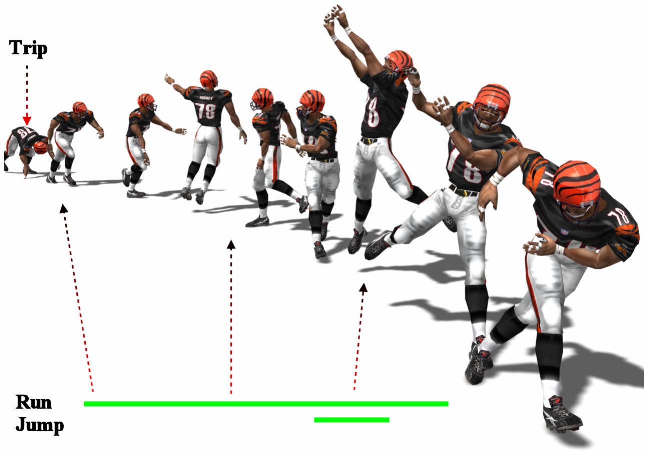

Figure 1: In this automatically synthesized motion, the figure is constrained to be tripping, then running, then jumping while still running.

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

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

a new motion. These algorithms [Arikan and Forsyth 2002; Lee et al. 2002; Li et al. 2002; Pullen and Bregler 2002; Molina-Tanco 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 reoptimized. 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.

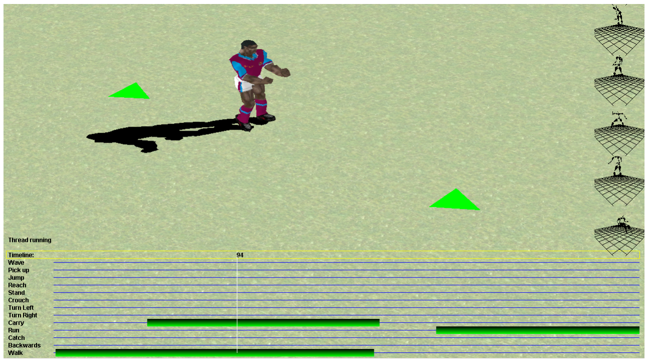

Figure 2: The user interface allows the user to see each available annotation label (bottom of the screen), and paint positive annotations (green bars) and negative annotations (blue bars). The frames that are not painted are interpreted as *don't care*. The user can manipulate geometric constraints 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,

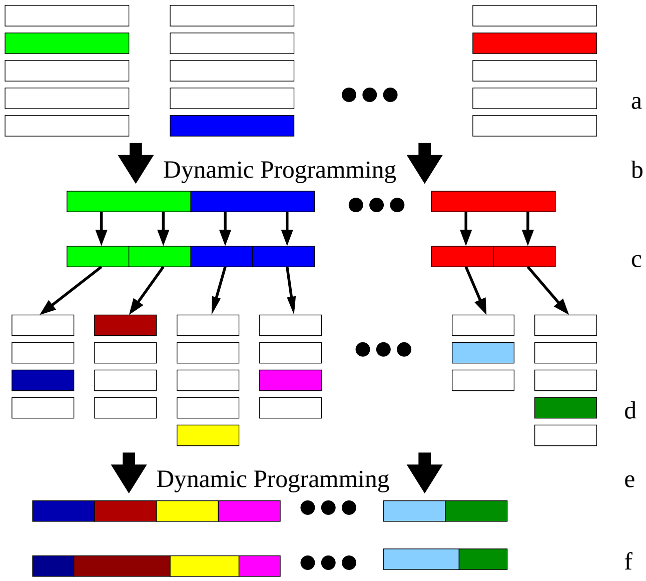

Figure 3: The motion is constructed out of blocks of subsequent frames from a motion database (see section 3.3). (a) 32 frame blocks of motions are selected at random with stratification to be the active set for each time slot that needs to be synthesized. (b) Dynamic programming is used to choose an 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:

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

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:

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

$$J(1, f_j) = D(1,A(f_j)) \tag{4}$$

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 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.

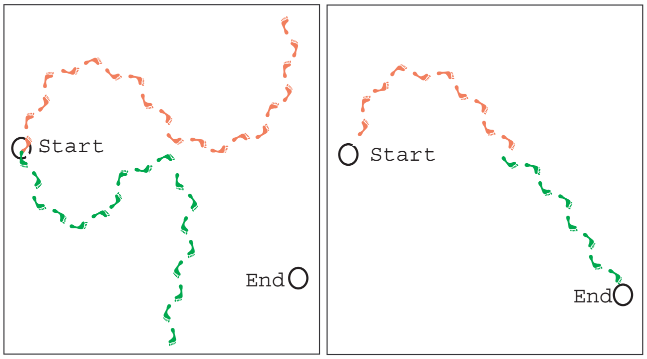

Figure 4: Left- Two motions in the pool of synthesized motions for positional constraints, neither going to where we want it to go. Right- The final motion that is created by putting the end of one motion after the beginning of the other motion achieves the positional constraints.

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.

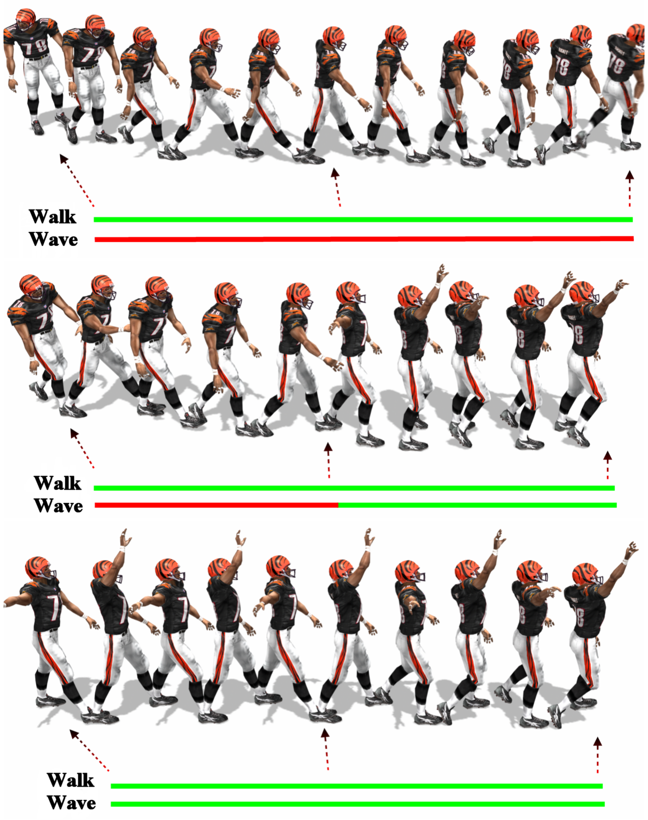

Figure 5: The framework can synthesize motions that match a given set of annotations marked on the timeline. For example, the top figure shows a synthesized motion for walking but not waving. The middle picture is synthesized for walking but waving only for the second half of the motion. The bottom motion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

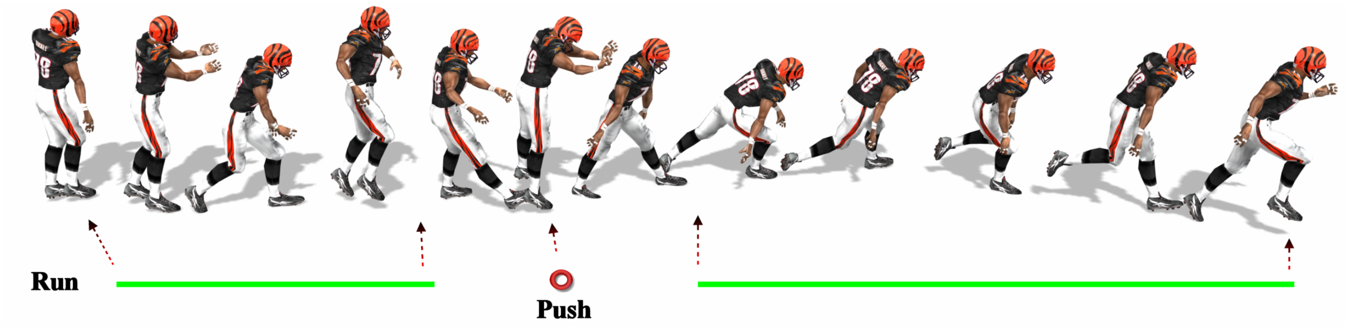

Figure 6: In addition to matching the annotations, a specific frame or motion can be forced to be used at a specific time. Here, the person is forced to pass through a pushing frame in the middle of the motion while running before and after the pushing.

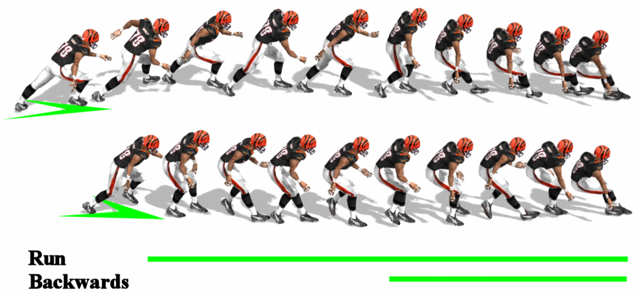

Figure 7: The search can also take positional constraints into account while synthesizing motions for given annotations. Here, the figure is constrained to be running forward and then running backwards. We enforce position constraints indicated as green arrows. For clarity, the running forwards section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