The Visual Computer

Constrained synthesis of textural motion for articulated characters

Shmuel Moradoff,
Dani Lischinski

School of Computer Science and Engineering,
The Hebrew University of Jerusalem, Israel
E-mail: danix@cs.huji.ac.il

Published online: 4 March 2004

Obtaining high-quality, realistic motions of articulated characters is both time consuming and expensive, necessitating the development of easy-to-use and effective tools for motion editing and reuse. We propose a new simple technique for generating constrained variations of different lengths from an existing captured or otherwise animated motion. Our technique is applicable to *textural* motions, such as walking or dancing, where the motion sequence can be decomposed into shorter motion segments without an obvious temporal ordering among them. Inspired by previous work on texture synthesis and video textures, our method essentially produces a reordering of these shorter segments. Discontinuities are eliminated by carefully choosing the transition points and applying local adaptive smoothing in their vicinity, if necessary. The user is able to control the synthesis process by specifying a small number of simple constraints.

Key words: Animation – Articulated characters – Constraints – Motion editing/synthesis/reuse – Textural motion

1 Introduction

High-quality motion control of articulated figures is one of the most challenging tasks in computer animation. Such motion may be specified manually by skilled animators with the aid of sophisticated software tools, generated using simulation, or captured using optical or magnetic tracking. All of these creation processes can be tedious, time consuming, and expensive. Therefore, there is a real need for a variety of easy-to-use and effective tools for motion editing and adaptation to facilitate motion reuse.

In this work we describe a new tool for generating constrained variations of different lengths from an existing captured or otherwise animated *textural motion*. By this term we refer to motion that can be regarded as a stationary signal at some scale. Informally, textural motion is decomposable into segments whose duration is typically small with respect to that of the entire motion such that by looking at any given segment it is impossible to say which part of the motion it came from.[1]

Many human motions are either entirely textural or have large textural parts. One example is walking since it can be decomposed into single step cycles, which can typically be reordered without making a noticeable difference to an observer. Dancing is another, more interesting, example. A dance sequence can often be decomposed into short choreographic elements that may be reordered to yield a different sequence that would still look like a natural dance sequence to an observer. A pole vaulting sequence, however, is nontextural since it does not consist of elements that could be reordered.

Our tool has many possible applications. Using an existing library or relatively short motion sequences, an animator could use our technique to generate a much larger variety of motions subject to animator-specified constraints. An existing motion can be made longer or transformed into a loop by placing an identical constraint at the beginning and at the end of the synthesized motion sequence. A group of characters may be easily animated by assigning each character a variation of the same single captured motion. A motion may be fine-tuned to a new script or adapted to a new soundtrack by appropriate placement of a small number of constraints. In all of these tasks we do not attempt to modify the motion's style or alter any other high-level characteristics; quite the

[1] Our definition of textural motion resembles that of a quasiperiodic signal, but it is slightly more general since we do not require that the motion elements be similar to each other.

The Visual Computer (2004) 20:253–265
Digital Object Identifier (DOI) 10.1007/s00371-003-0231-1

contrary, we attempt to remain as close as possible to the input motion sequence.

Our approach is inspired by previous work on texture synthesis [3, 4, 10, 27] and video textures [25]. Given a motion sequence and a small set of simple constraints, we essentially produce a reordering of short segments present in the input sequence that satisfies the specified constraints. Such a reordering inevitably introduces discontinuities into the synthesized motion. To eliminate these discontinuities, we find transition points at which the magnitudes of the discontinuities are minimized and apply an adaptive smoothing scheme in their vicinity.

The main contribution of this work is a new simple and fast tool for editing and reusing existing motions. Our approach enables the animator to specify hard constraints, thus providing low-level, fine-scale control over the resulting motion. By avoiding the construction of explicit high-level statistical models of the input data, our approach is applicable even to short yet complex motions and does not require large training sets.

2 Previous work

In the past few years the problem of editing and reusing existing motion has attracted considerable attention. Several researchers have explored ways of modifying motion by means of signal processing techniques [7, 26, 29]. This approach may be used to generate entire families of realistic motions from a single input sequence with a small amount of input from the animator. Our approach is complementary to these techniques: we also provide a tool for easily generating variations from a given motion, but we operate by essentially "reshuffling" the input sequence rather than applying signal processing operations on it.

Another relevant and powerful paradigm is constraint-based motion editing. A survey and comparison of such methods, which may be used to alter existing motions so as to satisfy a set of spatial or geometric constraints, is provided by Gleicher [13]. In particular, much work has been done by Gleicher and coworkers on retargeting motion to new characters [12], motion adaptation [15], and motion path editing [14], as well as by Lee and Shin [19] on interactive motion editing. Other notable representatives of this paradigm are spacetime constraint methods [8, 11, 28], which satisfy constraints by optimizing over an entire motion.

All constraint-based motion editing methods employ inverse kinematics solvers and often involve constrained nonlinear optimization. Such methods also require the animator to specify a large set of constraints such as constraints for the foot to touch the ground at each time slot. Our approach employs neither of the above, operating in a fundamentally different manner. It is not intended to be as general or as powerful as these methods; rather, it is a simple-to-use and fast alternative for editing textural motions. Again, we see it as a complementary tool rather than as a replacement for any of these methods.

Brand and Hertzmann [6] introduced *style machines*, probabilistic finite-state machines augmented by a multidimensional style variable. A style machine is constructed using unsupervised machine learning from a training set of several motion capture sequences. The learning process automatically distinguishes between the structure of a motion (its choreography) and its style. Once a style machine has been learned, it is possible to change the style of a given new motion, assuming it consists of the same set of structural elements. It is also possible to generate an entirely new motion from a style machine by specifying the choreography as a reordering of the machine's state sequences and choosing values for the style variables. The reordering can be performed by an animator or generated using a random walk. A similar, although somewhat less sophisticated, approach was independently developed by Bowden [5]. Style machines are an excellent high-level tool for motion editing and reuse. However, they do not provide an animator with low-level, fine-scale control, such as the ability to specify hard constraints: "I want the character to reach the highest point of its jump exactly at time t". In practice, both high-level and fine-scale controls are necessary for effective motion editing. For example, an animator could take a dance sequence and change its style using a style machine and then fine-tune the timing of the choreography by feeding the restyled motion to our method, along with a few constraints.

Our approach resembles the one described by Schödl et al. [25] for the generation of *video textures*. In particular, they discuss *video-based animation*, where a user is provided with high-level controls for guiding video texture synthesis. However, to our knowledge, their approach has not been applied to 3D articulated figure motion synthesis, and it does not di-

rectly support hard constraints such as the ones used in our approach. We also borrow many ideas from recent work on texture synthesis [3, 4, 10, 27].

Several other researchers have applied texture synthesis techniques to the problem of motion synthesis. Probably the first step in that direction was taken by Perlin and Goldberg in their *Improv* system [22], which uses procedural noise to add realistic looking variability to motions.

Pullen and Bregler [23] describe a motion synthesis method inspired by De Bonet's texture synthesis algorithm [9]. Their approach decomposes the training data into frequency bands and synthesizes a new sequence, one frequency band at a time. The approach was applied to generate a realistic repetitive motion of a 2D character with three degrees of freedom (a hopping wallaby). Our approach is different since it is closer to more recent texture synthesis techniques [10, 27], which generally outperform De Bonet's algorithm in terms of the quality of the synthesized textures. In this paper we demonstrate our approach using a variety of complex motions performed by a 3D articulated character with 23 joints and end effectors.

Pullen and Bregler [24] extended their work by adding an input of a partial key-framed data, for example, an animation of the legs of some character. They match one or two degrees of freedom in the low-pass band of the captured motion to the key-framed data, split the data into short segments, and search for matches between the key-framed and the motion-captured segments. The high-pass band of the motion-captured sequences is then used to fill in the missing details in the partial key-framed data. In contrast, our technique does not require the extra key-framed data to create a new full animation sequence.

Molina Tanco and Hilton [21] construct a two-level statistical model from motion capture training data. The motion capture data are clustered, and the clusters define the states of a Markov chain, which encodes the high-level temporal behavior of the character. The second level relates the states of the Markov chain to segments of the original motion. Dynamic programming is used to generate the state sequence of a synthesized motion. The actual motion is then generated by copying segments from the second level and blending in the areas where these segments join. Li et al. [20] also describe a two-level statistical model. They divide the motion into segments such that each segment is associated with one texton. A texton is modeled by a linear dynamic system (LDS), while the texton distribution is represented by a transition matrix indicating how likely each texton is switched to another. Synthesis of motion is done by first generating a texton path (the user may edit the texton path) and then, for each texton, synthesizing a texton sequence that begins with two key poses (that are associated with each texton) and constraint to end with two frames that are the key poses of the next texton in the path. Our approach is different in that it does not construct an explicit statistical model of the input data. To satisfy the animator's constraints, our approach efficiently identifies good transitions directly in the original motion capture data. Thus our approach should reduce both the number of transition blends and the magnitudes of the discontinuities that must be blended.

Some recent work, concurrent to ours, introduced the concept of motion graphs. In general, a huge matrix of all transitions from one frame to another is created. A cost is associated with each transition. The transition graph is then created and pruned. Motions are synthesized by looking for low-cost paths in the graph, sometimes subject to constraints.

Kovar et al. [16] used this framework to create motion that follows a user-specified 2D path. They use a metric similar to ours to compute the distance between every two frames and create a distance matrix, which is then pruned by taking only local minima in the matrix and thresholding. Pruning is essential since the data that they are using contain many similar frames. A graph of all possible transitions is then created and pruned again to obtain the largest strongly connected component in order to avoid dead ends. Finally, a path that minimizes some user-defined cost function is found. This approach handles new 2D path synthesis very nicely, but it does not allow the user to specify hard constraints as we do, and the input data must contain many good transitions between frames.

Lee et al. [18] use a low-level Markov process to create a probability matrix for transitions between every two frames. Then they prune the matrix much like Kovar et al. [16], with the addition of pruning frames based on contact. They also create a higher-level statistical model that clusters the frames in a transitions graph such that a motion can be thought of as a path between clusters. They describe three ways to create a motion: the user may choose the next cluster to follow, sketch a 2D path (as in [16]), or act in front of a camera set. This method also requires a large mo-

tion database to work well and does not handle hard constraints.

The motion graphs framework was also used by Arikan and Forsyth [2]. Their database contains a large number of short motions. A transition matrix is constructed between each pair of motions. After pruning, clusters of elliptical shape emerge in the matrix. Each ellipse is clustered with k-mean clustering, which is actually a summarized graph. Each cluster is repeatedly split, yielding a hierarchical representation of the graph. A motion is represented as a path of clusters from different levels in the hierarchy. Only paths that meet user-defined hard constrains are allowed, and a path is chosen that best fits some soft constraints. To meet the hard constraints and to obtain the ellipses, the data should have many similar frame sequences.

All three of the methods mentioned above were designed to work with large motion databases and thus require lengthy preprocessing times (several minutes to several hours). In contrast, our approach was designed to work even with short yet complex captured motions in which it might be difficult to find many good transitions between frames or achieve good clustering. The total computation time required by our method is about the same as the length of the synthesized animation.

Finally, there is also some related work in the area of computer vision concerned with learning 3D human motion for tracking and gesture recognition (e.g., [30]). In these works, motion synthesis is typically a secondary goal, and therefore little attention is paid to animator control (constraints) and the quality (smoothness) of the synthesized motions.

3 Textural motion synthesis

3.1 Overview

Our method takes as input a textural motion sequence, a set of synthesis constraints, and the desired length of the synthesized output sequence. The input motion sequence consists of a character's skeleton hierarchy description followed by a sequence of *frames*. Each frame specifies the absolute pose of the articulated figure by means of six degrees of freedom for the root of the hierarchy (three rotation angles and a 3D translation) and three rotation angles for each joint. All rotations are internally represented using quaternions.

We call this representation the *joint angle representation*. This representation has the property that modifying various values, such as the translation of the root or the rotation angles of a joint, does not deform the structure of the skeleton. Therefore, this is the representation that we use when filtering frames. However, this representation is not appropriate for computing differences between frames: the orientation of each joint affects all joints below it in the skeleton hierarchy; as a result, a small change in the orientation might have a profound impact on the character pose when the joint is high in the hierarchy, while a change of the same magnitude on another joint near the bottom of the hierarchy might have a very small impact on the pose. Furthermore, the same change in an orientation of a particular joint might have a different impact on the overall pose, depending on the position of the joints below it in the hierarchy.

Therefore, we construct an additional representation for each frame: the *3D pose representation*, which is the set of 3D positions of all joints and end effectors. This is a much more reasonable representation for comparing frames, but, as expected, modifying values in this representation deforms the character's skeleton. Therefore, in our system we use both of these representations: the joint angle representation is used for filtering and modifying frames, while the 3D pose representation is used for computing distances (transition costs) between frames.

The constraints are simply a set of pairs; pair (i, j) means that the i-th frame in the input sequence is constrained to become the j-th frame in synthesized output sequence. Optionally, a new path is also specified for the synthesized motion.

The output of our method is a new motion sequence of the desired length. For the most part, it consists of subsequences of frames from the original sequence reordered in such a manner that all of the constraints are satisfied. An exception is the frames in the neighborhood of the transitions between the subsequences; such frames might be slightly modified by our local smoothing scheme for eliminating discontinuities, as described in Sect. 3.6.

In our current implementation the output length must be an integer multiple of the input length, but this is not a real limitation since in order to generate an arbitrary length sequence we can generate one of length rounded up to the nearest integer multiple and then trim a few frames from the ends.

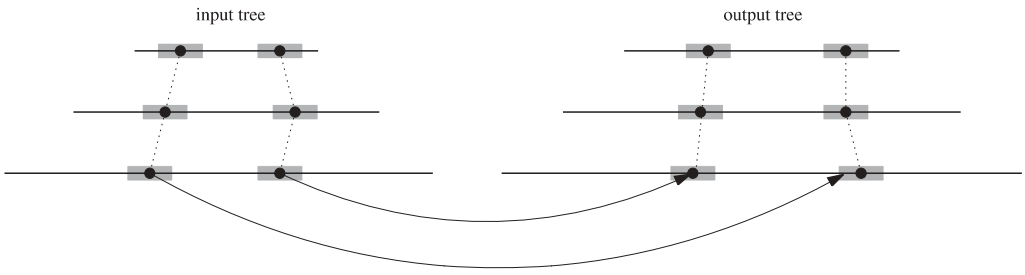


Fig. 1. Input and output trees. The *horizontal lines* represent levels in the trees. The bottom level in each tree contains the frames at the original temporal resolution. The next level up contains frames after low-pass filtering and subsampling, and so forth. *Arrows* indicate constraints: in the initialization stage each constrained frame and each of its ancestors in the input tree (shown as *black circles*) is copied to its designated location in the output tree along with a small neighborhood of siblings (shown in *gray*). It is now up to the synthesis procedure to fill in the remaining gaps, proceeding from the top level down

The synthesis procedure employed by our method is similar to the multiresolution constrained texture synthesis procedure described by Wei and Levoy [27]. The main steps of our method are:

1. Construct an *input tree*: a Gaussian multiresolution tree above the sequence of frames (in the joint angle representation) by iterative low-pass filtering and subsampling.
2. Apply forward kinematics to compute the absolute 3D pose representation of the articulated character for each frame on all tree levels.
3. Create an empty *synthesis tree* with the same number of levels as the input tree. The finest resolution level of this tree will eventually contain the output sequence.
4. Copy the constrained frames (and their ancestors) to their target locations in the synthesis tree. Each constrained frame (ancestor) is copied along with a small surrounding neighborhood (Fig. 1).
5. Starting from the coarsest level, fill in all the gaps left between the constraints by searching for best-matching neighborhoods at the corresponding level of the input tree, as described in Sects. 3.3, 3.4, and 3.5.
6. Apply adaptive local smoothing in the vicinity of transitions between original motion subsequences to eliminate discontinuities, if necessary (Sect. 3.6).
7. Repeat for the next level until the finest level has been filled and smoothed.
8. Construct a new root rotation and translation trajectory, as described in Sect. 3.8.

In the remainder of this section we describe in more detail the key steps of our method (steps 1, 5, 6, and 8 above).

3.2 Multiresolution tree

Constructing a multiresolution tree is required for the acceleration of the gap filling algorithm (described in Sects. 3.4 and 3.5). We create the tree by iterative low-pass filtering with a Gaussian filter, and subsampling.

The size of the filter should depend on the nature of the motion. A motion with sharp or fast movements will need a shorter filter because a long filter might use some very different frames to create a certain frame at a coarser level, which means that the "essence" of the frame might get lost. For our data we typically used a filter of size four.

The number of levels is also important: more levels leads to faster synthesis. On the other hand, if too many levels are used, the motion at the coarsest level might be just a series of frames with no affinity between them, since we subsample each level. For the results shown in this work, we used between three and five levels (four levels were used in most cases) without any noticeable difference in the quality of the resulting motion.

3.3 Neighborhoods and metrics

Gaps in the synthesis tree are regarded as sequences of empty "frame slots". To fill in an empty slot,

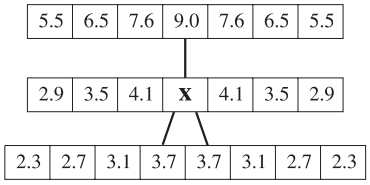


Fig. 2. The three-level neighborhood used in our implementation. The empty cell whose value is to be determined is marked by an $\boldsymbol{x}$. The neighborhood consists of $\boldsymbol{x}$, its ancestor and its children in the tree, and three more slots to the left and to the right on each level. The *numbers* in the cells are the relative weights used by our distance metric

we examine its neighborhood in the synthesis tree and look for similar neighborhoods around slots at the same level of the input tree. We use a three-level neighborhood that is similar, in principle, to the neighborhoods used by Wei and Levoy [27], but there are two important differences:

1. Slots from the next finer level are included – although this level has not been processed yet, some of its slots may have been filled by constrained frames during the initialization of the output tree. The synthesis algorithm must take these frames into account when searching for the best frame for the current slot.
2. We give different weights to different slots in the neighborhood (decreasing with distance from the center slot). The relative weights of the slots were empirically determined and are shown in Fig. 2.

The difference between two neighborhoods is defined simply as a weighted sum of the distances between corresponding pairs of slots. Pairs in which the output neighborhood slot is still empty are assigned a weight of zero, and all of the remaining nonzero weights are renormalized to sum to one.

The definition of distance between corresponding slots is slightly more involved. Consider two slots containing frames numbered i and j. Simply computing the distance (in some vector metric) between the corresponding 3D pose vectors v_i and v_j will not do: such a metric would fail to recognize the similarity between identical poses differing only by translation. Although translation invariance could be achieved by computing the position vectors with respect to a local coordinate frame, such a metric still does not account for the velocities of the joints, which could lead to unnatural direction reversals. Therefore, we compute the distance between the joint and end-effector velocity (temporal derivative) vectors $\dot{v}_i$ and $\dot{v}_j$ instead. Although in principle it is possible for two totally different character poses to have similar velocity vectors, in practice such a situation is extremely unlikely, and we have never encountered it in our experiments.

After experimenting with various standard vector metrics we concluded that a sum of $\|\cdot\|_\infty$ and $\|\cdot\|_{0.5}$ yields the best results. In other words, we take into account both the maximum difference among the velocity components and the sum of the square roots of the differences of all components. This metric was found to give slightly better results than the more familiar $\|\cdot\|_2$ norm in cases of small differences in many joints simultaneously.

3.4 Gap filling

We now describe in more detail how to fill in the gaps at a particular level of the tree. Each gap is a sequence of empty slots bounded by the constrained frames, either from one side or from both sides. We will focus on the latter case, which is the more interesting (and complicated) of the two.

One possible approach to filling such a gap is to start filling the empty slots, working our way from both ends toward the middle. However, choosing the middle of a gap as the meeting point is an arbitrary decision, and it will not necessarily result in the most continuous transition. Therefore, our strategy is first to find the best meeting point inside the gap and then work our way toward it from both ends. We start at the coarsest level of the tree. For each gap, we search for a meeting point by employing an exhaustive search. We consider all the interior slots of a gap as potential meeting points and perform the synthesis toward each such point from both ends, and then we record the resulting *sequence cost*. The sequence cost is defined as the sum of distances between the neighborhood of each synthesized slot and its best matching neighborhood in the input tree. The meeting point with the lowest sequence cost is then chosen.

The exhaustive search is expensive since for a gap of k slots we consider k different ways of filling it, each time searching for the best matching neigh-

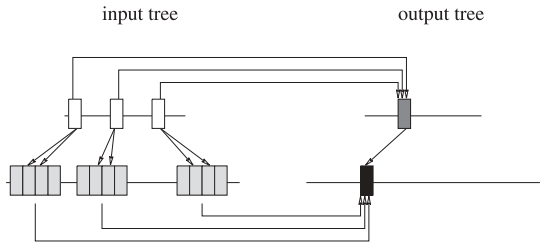


Fig. 3. Acceleration of the synthesis process. The *black frame* is the current slot to be synthesized. The *dark-gray frame* is the black frame's father. The *white frames* are the best three candidates for the *dark-gray frame*, and the *light-gray frames* are the white frames' sons and their neighbors. The *light-gray frames* are the only candidates for the *black frame*

borhood for each slot. Thus, if we have n different neighborhoods in the input sequence at that level, we end up evaluating the neighborhood distance metric k^2n times. However, at the coarsest level of the tree, both k and n are typically small. In the following (higher-resolution) levels we use the meeting point that we found in the previous level as an initial guess and examine only a small neighborhood of slots around this initial guess to find the new meeting point.

We use an additional optimization to speed up the synthesis at the finer levels of the tree. When filling an empty slot, instead of searching for a match among all of the neighborhoods at the corresponding level of the input tree, we only examine a constant number of neighborhoods: those around the children of the slot chosen in the previous level to fill in the parent of the current slot (Fig. 3). This optimization accelerates synthesis by at least one order of magnitude. The same idea was used to speed up texture synthesis by Bar-Joseph et al. [4] and by Ashikhmin [3].

So far, the approach we have described is entirely deterministic. In order to produce some variations between different runs of the algorithm (even with the same set of constraints), we are using a slightly randomized version of the approach: after determining the meeting point for a gap, we synthesize the sequence again, but this time instead of filling slots based on the best-matching neighborhood, we consider a small set of neighborhoods (typically two to three) that had the best matches and choose one of them at random. This randomization takes place at all levels of the synthesis tree except the last, finest level.

3.5 Finding a path in a graph

In the spirit of [2, 16, 18], we experimented with searching a transition graph for a lowest-cost path between constraints as an alternative to the gap filling method, described in Sect. 3.4. The path finding approach enables us to create sequences with more than one meeting point inside a gap between two constraints.

For the coarsest level, we create a matrix M, where M_{ij} is the distance between frame i and frame j. M_{ij} is calculated using the same metric as before, except that in this case both frames are taken from the original data. Now we build a pruned graph where each node corresponds to a frame, and for each M_{ij} smaller than some threshold there is an arc in the graph from node i to node j, with the weight M_{ij}. For each gap in the coarsest level, we search for a path that will start with the frame that is just before the gap. A legal path will have the length of the gap and will end with the frame just after the gap. We use DFS (depth-first search) to find all legal paths and select among them the one with the lowest cost (sum of weights). We accelerate the search by stopping the descent at branches whenever some maximum cost has been exceeded or if we have already encountered a path with lower cost than the current one. Note that while in the full transition matrix many legal paths exist, in the pruned graph the existence of a legal path is not guaranteed. Thus the path finding approach might not be able to satisfy the hard constraints if these were poorly chosen.

At the finer levels, we search the neighborhood around each meeting point from the previous level for the best path among all combinations of their children, using randomization as in Sect. 3.4. This time the cost of a path is calculated using distances between frames in the synthesized data, which means that we get a more accurate result since we consider the actual neighborhood that was synthesized and not the input data.

3.6 Local smoothing

Some input sequences do not contain a sufficient number of repeating character poses. Thus, discon-

tinuities may occur at transition points between subsequences of the original motion. The same problem was encountered in video textures by Schödl et al. [25].

Since we know precisely at which frames of the synthesized sequence such transitions take place, we adaptively apply local smoothing in a small neighborhood around such transitions. More specifically, let i and $i+1$ be two successive frames in the synthesized sequence that were not successive frames in the original sequence. The smoothing is performed as follows:

1. Numerically approximate the acceleration (second derivative) of each joint and end-effector 3D location at frames i and $i+1$.
2. Compute the average acceleration of each joint and end-effector 3D location over frames $i-3$ through $i-1$ and $i+2$ through $i+4$.
3. If the accelerations at frames i and $i+1$ are less than or equal to the average acceleration computed in the previous step, there is no need to perform any smoothing.
4. Otherwise, the trajectories of the joints are smoothed by applying a Gaussian filter of width 5 at the transition frames i and $i+1$. The filter is applied to the quaternions representing the joint rotations, after which the corresponding 3D location vectors are recomputed by forward kinematics.
5. Steps 1, 3, and 4 are repeated as many times as necessary until a discontinuity is no longer detected in step 3. At each iteration we increase the neighborhood over which smoothing is performed in step 4. Specifically, in the k-th iteration we convolve frames $i-k$ through $i+k+1$ with the Gaussian filter.

In our experiments, the smoothing algorithm did not exceed three iterations, that is, $k \leq 3$. This means that the smoothing filter was applied over six frames or less (at each meeting point).

There is no mathematical guarantee that the modified frames produced by the smoothing operation necessarily correspond to valid and natural character poses. However, since the smoothing is performed on the joint angle representation of the frames (using quaternions), the skeleton is never deformed by this operation. In practice, we typically smooth frames that came from the original motion (and thus correspond to valid natural poses). The transitions are chosen by our algorithm such that these frames are as similar to each other as possible, thus minimizing the chance of getting an unnatural pose after the smoothing. As shown by our results, this adaptive local smoothing scheme successfully eliminates visual discontinuities in the motion without introducing conspicuous artifacts.

3.7 Choosing the constraints

The animator must specify a set of constraints as input for our method. A poor or random choice of constraints will usually result in a poor output.

First, at least one constraint is required since our algorithm must have at least one frame from which to start the synthesis. This frame could come from anywhere in the synthesis sequence (but see remarks below). Of course, in order to better control the resulting motion, more than one constraint is necessary.

In general, it is not a good idea to choose constraints from the very beginning/end of the input sequence. Consider a frame from the beginning of the input sequence that is selected as the constraint and placed in the middle of the target sequence. Since this frame is only preceded by a few frames in the input sequence, this might make it impossible to find a good transition point between this constraint and the one preceding it in the target sequence. This problem is similar to the dead-end problem in video textures [25].

When setting constraints, the animator should avoid placing them too close to each other in the synthesized sequence. If, for example, the Gaussian trees have four levels, then setting two constraints with less than seven frames between them will cause a conflict between the two constraints at the coarsest level (where they correspond to the same frame). In fact, it is desirable to leave much larger gaps between constraints in the synthesized sequence since we copy a few of the neighboring frames along with each constraint.

As mentioned above, setting random constraints might not result in a good synthesized motion. Suppose the input motion is of a walking character, where each step cycle consists of about 40 frames. In other words, every 40 frames the feet are roughly at the same position. Now suppose that we constrain frame number 1 of the input motion to appear as frame number 1 of the synthesized sequence and also to appear at frame number 60 of the synthesis sequence. The resulting motion will then be forced to have either a single step cycle with 60 frames in it or two short step cycles. If the gap filling method

of Sect. 3.4 is used, this might result in a large discontinuity, which would require a lot of smoothing to fix. However, with the graph path finding method (Sect. 3.5) the algorithm might be able to find several meeting points in that gap, spreading the error between these points. Smoothing will then eliminate the small discontinuities at the meeting points. Of course, if the input motion is more diverse, containing step cycles of varying lengths, this problem will be alleviated, even for the first gap filling algorithm.

3.8 Root trajectory reconstruction

In our current implementation the user is able to specify a new animation path (root trajectory) for the cases where the input motion advances roughly along a straight line. For each synthesized frame, we take the root trajectory derivative (velocity) from the original motion and use this velocity to compute the new root translation in the synthesized sequence, taking into account changes in the local reference frame along the new path. This ensures that the character advances along the new path in a similar fashion to the original motion.

If a new animation path has *not* been specified by the user, we reconstruct one by taking the root trajectory derivatives from the original motion, as before, and simply integrating them to obtain a new root trajectory.

4 Results

We implemented our algorithm in C++ as a plug-in for the Maya animation system [1] and have been able to generate a variety of motions from different motion capture sequences.

Figure 4 and the accompanying movie (drunk.mpg) demonstrate an application of our method to modify and double the length of a drunk person walking sequence. Various statistics regarding this example (as well as the other examples in this section) are summarized in Table 1. All of the corresponding movies can be found at http://www.cs.huji.ac.il/labs/cglab/research/ltca. The top row of Fig. 4 shows a few frames from the resulting animation, where we have placed the original motion in the middle and the synthesized motions on the right and left. The plots below show the trajectories of two joint angles in the shorter original sequence (top plot) and in the longer synthesized ones (bottom plots). The constraints specified by the user for this case are indicated by the letters "A" through "H". The letters in each plot indicate the locations of the constrained frames in the corresponding sequence.

This is a rather challenging test case since the input sequence contains only 14 step cycles, almost none of which is very similar to the others because of the waving arms and the stumbling nature of the walk, and the pose of the character hardly ever repeats itself. Therefore, we believe it would be difficult to construct a good high-level motion model [5, 6, 21] from such a training set. Our method, however, is still able to generate visually continuous and natural-looking motions by locally smoothing over discontinuous transitions. The local and adaptive nature of the smoothing succeeds in preserving the characteristics of the original motion.

With this example we also used the graph path finding algorithm described in Sect. 3.5 (the resulting movie is drunk-path.mpg). Using fewer constraints we forced the algorithm to find paths with more than one meeting point inside a gap. This sometimes leads to lower-quality synthesized motions (note the jerks in the motion of the character on the right around seconds 15 and 26 of the drunk-path.mpg movie).

Our next example uses a high-wire walking input sequence (Fig. 5, top row and the movie high-wire.mpg). Again, the basic cycle of this motion is quite complex and does not repeat itself too much,

Table 1. Statistics for the synthesis examples. Times were measured in seconds on a 866-MHz Pentium III PC. The numbers in parentheses refer to the graph path finding method described in Sect. 3.5

Name	No. joints	Original length	New length	No. constraints	Synthesis time (s)
Drunk walk (A)	23	512	1024	7 (3)	37 (23)
Drunk walk (B)	23	512	1024	8 (3)	43 (21)
High wire	23	512	1024	6	32
Ballet walk	23	512	2048	9	81
Cool walk	23	512	1024	8 (3)	30 (13)

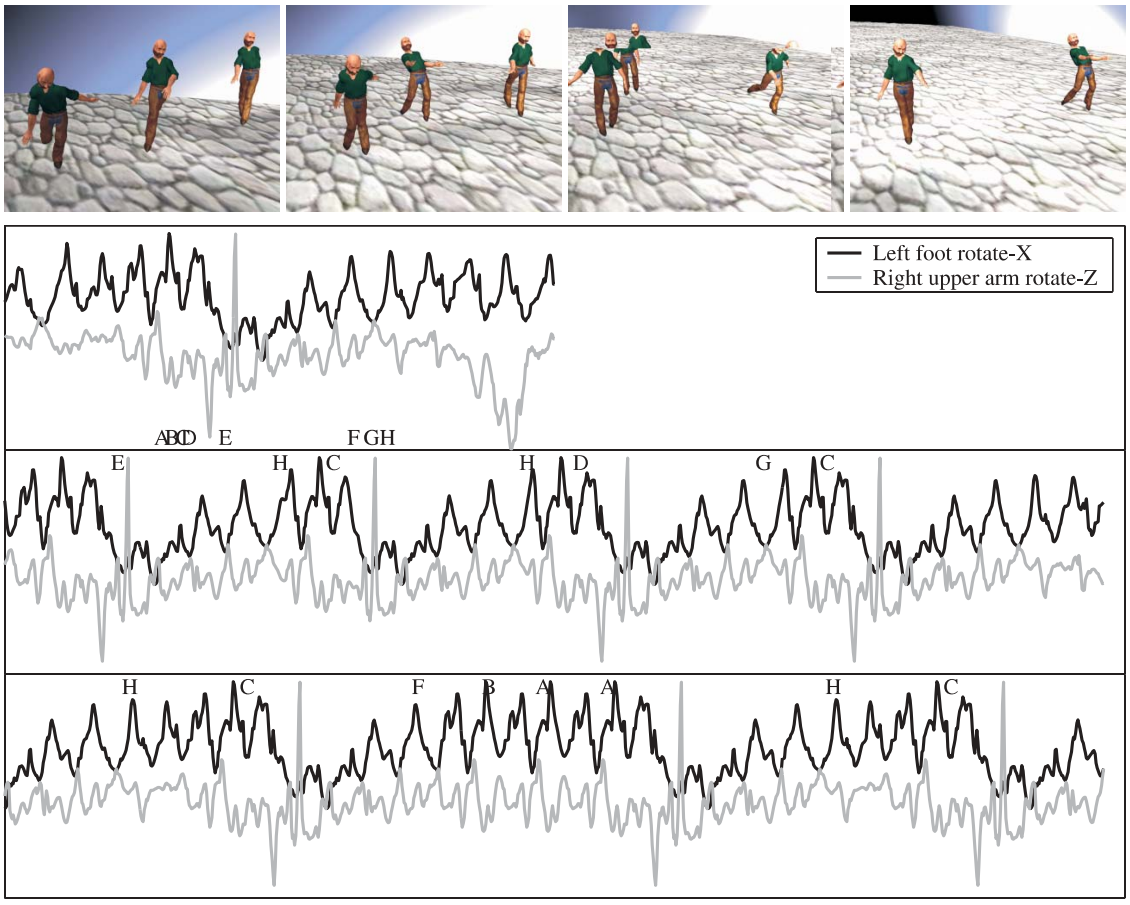


Fig. 4. Drunk walk. *Top row*: four frames from an animation we have generated (drunk.mpg). The original sequence is performed by the middle character; the other two were synthesized by our method. The three plots illustrate the results of the synthesis by means of the angular trajectories of two joint angles. The *top* plot corresponds to the original sequence. The letters *A* through *H* indicate the positions of the constrained frames in the original sequence and in each of the two synthesized ones

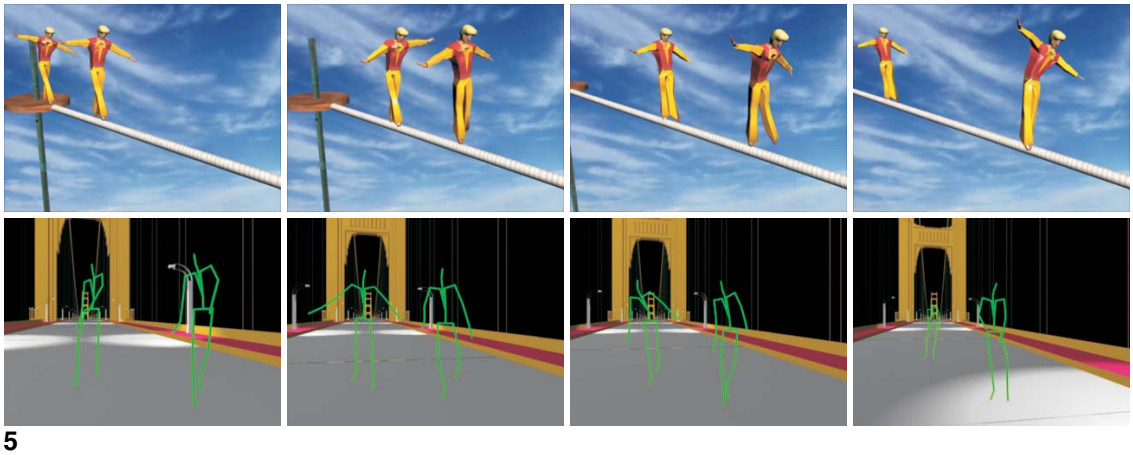
but our method succeeds in generating a longer sequence without noticeable discontinuities.[2]

Another example is a "cool walk" sequence (Fig. 5, bottom row and the movie cool.mpg). In this case the synthesized sequence was constrained to begin identically to the original sequence and then to continuously diverge from it.

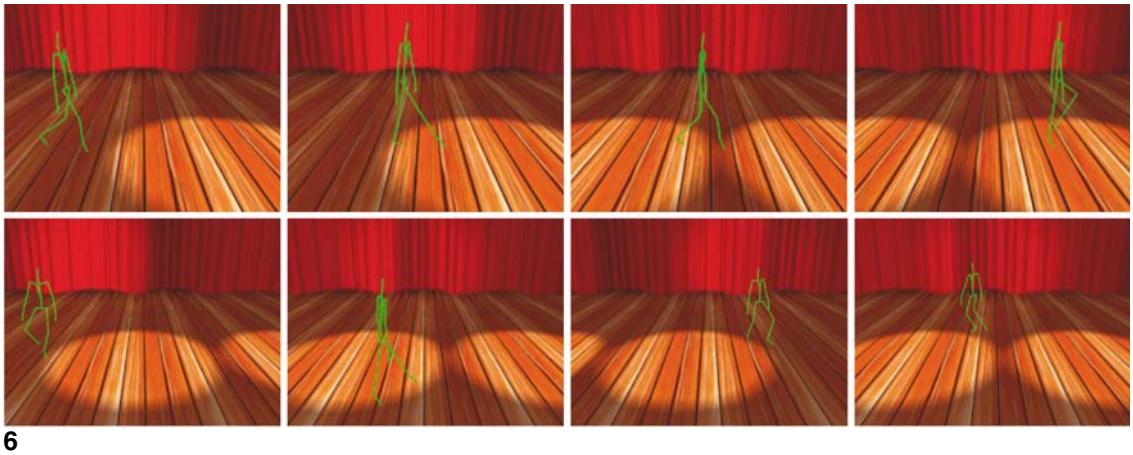
Here we also experimented with the graph path finding algorithm (cool-path.mpg). Note that with this method the synthesis time is much shorter. This is because there are not very many good transitions between frames, so when we search the graph for a path, many branches are excluded early in the search.

Finally, we apply our method to a "ballet walk" sequence. In the original sequence the character is walking in a roughly straight line (Fig. 6, top row, and the movie ballet-original.mpg). We constrained the synthesized sequence to begin and end with the same character pose and specified a new figure 8 path instead of the original straight one. The result is an animation loop of a person walking along the new path (Fig. 6, bottom row and the movie ballet-eight.mpg).

[2] A jerk in the right leg of the synthesized character can be noticed around seconds 8, 18, and 27. This flaw is present in the original motion, as can be seen by observing the original motion around second 6.



5



6

Fig. 5. *Top row*: High wire. Four frames from an animation (highwire.mpg). The original motion is performed by the character in the back. *Bottom row*: Cool walk (cool.mpg and cool-path.mpg). The original motion is performed by the character on the left

Fig. 6. Ballet walk. *Top row*: original straight line walk (ballet-original.mpg). *Bottom row*: synthesized motion loop along an figure 8 path (ballet-eight.mpg)

Limitations

As expected, our method appears to work best for motions containing many small segments similar to each other. For more complex motions with fewer repetitions it is more difficult for our method to find natural-looking transitions between pairs of constraints. Thus the animator must sometimes choose the constraints carefully, as discussed in Sect. 3.7.

Another limitation of our method is that it can be difficult to get interesting results from short input sequences.

Our method does not currently ensure various typically desirable properties, such as that the feet of a character do not penetrate the ground while walking. Such properties are best ensured using some of the previous constraint-based methods described in Sect. 2. Thus, we believe that in practice our method should be used in conjunction with these other techniques. For example, an animator might generate a new walking sequence using our technique and then apply spacetime constraints to ensure that the lowest point reached by the character's feet in each step cycle lies exactly on the ground, or he might

use Kovar and Gleicher's [17] work for footskate cleanup.

5 Conclusions

We have described a new tool for generating constrained variations from existing captured or otherwise animated textural motion sequences. Our technique is not intended as a replacement for previously developed tools for motion editing; rather it is meant to complement them, adding a new useful component to the animator's toolbox.

The graph path finding algorithm for gap filling seems to work faster and it is easier for the animator since he/she needs to set fewer constraints. However, with this algorithm the animator has less control over the outcome of the animation. Although we might get more than one meeting point inside a gap with this method, it searches the graph for a path with the lowest total cost; therefore, a path with multiple meeting points will be selected only if the sum of the transition costs is lower than the cost obtained with a single meeting point.

The graph path finding algorithm seems to work better on big databases, as explained in Sect. 2. As we have seen in the results, the meeting point search algorithm can work very well even if the input sequence is short and has few good transitions between frames.

In future work we plan to extend our method to mix together elements from several different input motion sequences, perhaps in a manner similar to the texture mixing algorithm of Bar-Joseph et al. [4]. We also plan to consider other, more sophisticated, types of constraints, such as the soft constraints described by Arikan and Forsyth [2].

Acknowledgements. This research was supported by the Israeli Ministry of Science and by the Israel Science Foundation founded by the Israel Academy of Sciences and Humanities. The motion capture sequences used as input in our experiments were obtained from Help3D.COM, Inc. (http://www.help3d.com) in BVH format and imported into Maya using the Dominatrix plug-in by House of Moves. The models were downloaded from 3DCAFE (http://www.3dcafe.com) and 3Dbuzz.com (http://www.3dbuzz.com).

References

1. Alias|Wavefront (2001) Maya 4.0
2. Arikan O, Forsyth DA (2002) Interactive motion generation from examples. In: Proceedings of SIGGRAPH 2002, San Antonio, 21–25 July 2002, pp 483–490
3. Ashikhmin M (2001) Synthesizing natural textures. In: Proceedings of the symposium on interactive 3D graphics, Research Triangle Park, NC, 19–21 March 2001, pp 217–226
4. Bar-Joseph Z, El-Yaniv R, Lischinski D, Werman M (2001) Texture mixing and texture movie synthesis using statistical learning. IEEE Trans Vis Comput Graph 7(2):120–135
5. Bowden R (2000) Learning statistical models of human motion. In: Proceedings of the IEEE workshop on human modeling, analysis, and synthesis (CVPR 2000), Hilton Head, SC, July 2000
6. Brand M, Hertzmann A (2000) Style machines. In: Proceedings of SIGGRAPH 2000, New Orleans, 23–28 July 2000, pp 183–192
7. Bruderlin A, Williams L (1995) Motion signal processing. In: Proceedings of SIGGRAPH 95, Los Angeles, 6–11 August 1995, pp 97–104
8. Cohen MF (1992) Interactive spacetime control for animation. In: Proceedings of SIGGRAPH 92, Chicago, 26–31 July 1992, Comput Graph 26(2):293–302
9. De Bonet JS (1997) Multiresolution sampling procedure for analysis and synthesis of texture images. In: Proceedings of SIGGRAPH 97, Los Angeles, 3–8 August 1997, pp 361–368
10. Efros AA, Freeman WT (2001) Image quilting for texture synthesis and transfer. In: Proceedings of SIGGRAPH 2001, Los Angeles, 12–17 August 2001, pp 341–346
11. Gleicher M (1997) Motion editing with spacetime constraints. In: Proceedings of the 1997 ACM symposium on interactive 3D graphics, Providence, RI, 27–30 April 1997, pp 139–148
12. Gleicher (1998) Retargeting motion to new characters. In: Proceedings of SIGGRAPH 98, Orlando, 19–24 July 1998, pp 33–42
13. Gleicher M (2001a) Comparing constraint-based motion editing methods. Graph Models 63:107–134
14. Gleicher M (2001b) Motion path editing. In: Proceedings of the 2001 ACM symposium on interactive 3D graphics, Research Triangle Park, NC, 19–21 March 2001, pp 195–202
15. Gleicher M, Litwinowicz P (1998) Constraint-based motion adaptation. J Vis Comput Animat 9(2):65–94
16. Kovar L, Gleicher M, Pighin F (2002) Motion graphs. In: Proceedings of SIGGRAPH 2002, San Antonio, 21–25 July 2002, pp 473–482
17. Kovar L, Schreiner J, Gleicher M (2002) Footskate cleanup for motion capture editing. In: Proceedings of the 2002 ACM symposium on computer animation, San Antonio, 21–22 July 2002, pp 97–104
18. Lee J, Chai J, Reitsma PSA, Hodgins JK, Pollard NS (2002) Interactive control of avatars animated with human motion data. In: Proceedings of SIGGRAPH 2002, San Antonio, 21–25 July 2002, pp 491–500
19. Lee J, Shin SY (1999) A hierarchical approach to interactive motion editing for human-like figures. In: Proceedings of SIGGRAPH 99, Los Angeles, 8–13 August 1999, pp 39–48
20. Li Y, Wang T, Shum H-Y (2002) Interactive motion generation from examples. In: Proceedings of SIGGRAPH 2002, San Antonio, 21–25 July 2002, pp 483–490
21. Molina Tanco L, Hilton A (2000) Realistic synthesis of novel human movements from a database of motion capture

examples. In: Proceedings of the IEEE workshop on human motion, Austin, TX, 6-7 December 2000, pp 137–142
22. Perlin K, Goldberg A (1996) Improv: a system for scripting interactive actors in virtual worlds. In: Proceedings of SIGGRAPH 96, New Orleans, 4–9 August 1996, pp 205–216
23. Pullen K, Bregler C (2000) Animating by multi-level sampling. In: Proceedings of Computer Animation 2000, Philadelphia, 3–5 May 2000, pp 36–42
24. Pullen K, Bregler C (2002) Motion capture assisted animation: texturing and synthesis. In: Proceedings of SIGGRAPH 2002, San Antonio, 21–25 July 2002, pp 501–508
25. Schödl A, Szeliski R, Salesin DH, Essa I (2000) Video textures. In: Proceedings of SIGGRAPH 2000, New Orleans, 23–28 July 2000, pp 489–498
26. Unuma M, Anjyo K, Takeuchi R (1995) Fourier principles for emotion-based human figure animation. In: Proceedings of SIGGRAPH 95, Los Angeles, 6–11 August 1995, pp 91–96
27. Wei L-Y, Levoy M (2000) Fast texture synthesis using tree-structured vector quantization. In: Proceedings of SIGGRAPH 2000, New Orleans, 23–38 July 2000, pp 479–488
28. Witkin A, Kass M (1998) Spacetime constraints. In: Proceedings of SIGGRAPH 88, Atlanta, 1–5 August 1988. Comput Graph 22(4):159–168
29. Witkin A, Popović Z (1995) Motion warping. In: Proceedings of SIGGRAPH 95, Los Angeles, 6–11 August 1995, pp 105–108
30. Zhao T, Wang T, Shum H-Y (2002) Learning a highly structured motion model for 3d human tracking. In: Proceedings of ACCV 2002, Melbourne, Australia, 23–25 January 2002

SHMUEL MORADOFF was born in Tel-Aviv, Israel in 1971. He received his B.Sc. in physics in 1998 and his M.Sc. in computer science in 2003, both from the Hebrew University of Jerusalem, Israel. His main interests include character animation and motion capture. The work presented in this paper is his M.Sc. thesis.

DANI LISCHINSKI received a B.Sc. and an M.Sc. in computer science from the Hebrew University of Jerusalem in 1987 and 1989, respectively, and a Ph.D. degree in computer science from Cornell University in Ithaca, NY, USA in 1994. He is currently on faculty of the School of Engineering and Computer Science at the Hebrew University of Jerusalem. Dr. Lischinski's areas of interest include computer graphics, scientific visualization, virtual reality, computational geometry, and scientific computing. In particular, he has worked on algorithms for photorealistic image synthesis, global illumination, robust triangulation and mesh generation, and interactive visualization of complex virtual scenes. Most recently he has been working on image-based modeling and rendering, texture synthesis, utilization of wavelets to perform fast multiresolution operations, and high dynamic range compression. For more information see http://www.cs.huji.ac.il/~danix.