

# Synthesis of Textural Motion with Hard Constraints

Shmuel Moradoff

Dani Lischinski

School of Computer Science and Engineering  
The Hebrew University of Jerusalem, Israel

## Abstract

*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 re-ordering 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.*

## 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 in a variety of easy-to-use and effective tools for motion editing and adaptation in order 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 which 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<sup>1</sup>.

Our tool has many possible applications. Using an existing library or relatively short motion sequences, an animator can use our technique to generate a much larger variety

<sup>1</sup>Our definition of textural motion resembles that of a quasi-periodic signal, but it is slightly more general, since we do not require the motion elements to be similar to each other.

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 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, 7, 17] and video textures [15]. Given a motion sequence and a small set of simple constraints, we essentially produce a re-ordering 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.

## 2. Previous Work

In the past few years the problem of editing and reusing existing motion has attracted considerable attention. Several researchers explored ways of modifying motion by means of signal processing techniques [6, 16, 18]. 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.

Brand and Hertzmann [5] introduced *style machines*, probabilistic finite-state machines augmented by a multi-dimensional style variable. 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  $t$ ".

Our approach resembles the one described by Schödl *et al.* [15] for generation of *video textures*. In particular, they discuss *video-based animation*, where a user is pro-

vided 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 directly support hard constraints, such as the ones used in our approach. We also borrow many ideas from recent work on texture synthesis [3, 7, 17].

Several other researchers applied texture synthesis techniques to the problem of motion synthesis. Pullen and Bregler [13, 14] used motion-captured data as a source for augmenting partially key-framed motions with high-pass band information and/or additional degrees of freedom. With our technique no extra key-framed data is necessary to create a new full animation sequence.

Molina *et al.* [12] and Li *et al.* [11] both construct a two level statistical model of the motion: a higher level that controls the overall motion and a lower level that handles the local dynamics. Our approach is different in that it does not construct an explicit statistical model of the input data. In order 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 transitions 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 transitions graph is then created and pruned. Motions are synthesized by looking for low-cost paths in the motion graph, sometimes subject to constraints. Kovar *et al.* [8] used this framework to create motions that follow a given 2D path. Lee *et al.* [10] added a higher statistical level to control the synthesized motion. Unlike our method, these two methods can’t handle hard constraints. Arikan and Forsyth [2] can handle strong constraints and weak constraints when synthesizing new motions, but their approach requires a very large database of captured motions. All of the three methods mentioned above were designed to work with large motion databases, and thus their pre-processing times are quite long (several minutes to several hours). In contrast, our approach was designed to work even with short captured motions, in which it might be difficult to find many good transitions between frames. The total computation time required by our method is about the same as the length of the synthesized animation.

### 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 the 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

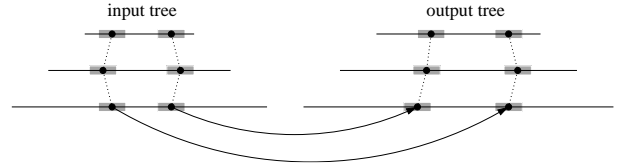


Figure 1. Input and output trees. The horizontal lines represent levels in the trees. Arrows indicate constraints: in the initialization stage each constrained frame and each of its ancestors in the input tree (shown as black circles) is copied to its designated location in the output tree, along with a small neighborhood of siblings (shown in grey).

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.

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 between 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 comparing 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 sub-sequences of frames from the original sequence reordered in such a manner that all of the constraints are satisfied. An exception are the frames in the neighborhood of the transitions between the sub-sequences; such frames might be slightly modified by our local smoothing scheme for eliminating discontinuities, as described in section 3.4.

The synthesis procedure employed by our method is similar to the multiresolution constrained texture synthesis procedure described by Wei and Levoy [17]. The main steps of our method are:

1. Construct an *input tree*: a Gaussian multi-resolution tree above the sequence of frames (in the joint angle representation) by iteratively low-pass filtering and subsampling. We typically use a tree with 3–5 levels. The purpose of this tree is to accelerate the synthesis algorithm, as will become apparent below.
2. Apply forward kinematics to compute the absolute

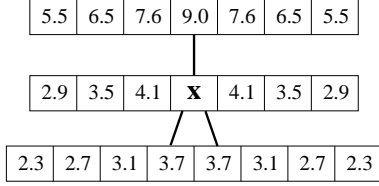


Figure 2. The three-level neighborhood used in our implementation. The empty cell whose value is to be determined is marked by an  $x$ . The neighborhood consists of  $x$ , its ancestor and its children in the tree, and three more slots to the left and to the right on each level. The numbers in the cells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

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 (Figure 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 sections 3.2 and 3.3.
6. Apply adaptive local smoothing in the vicinity of transitions between original motion sub-sequences to eliminate discontinuities, if necessary (section 3.4).
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 section 3.6.

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

### 3.2. Neighborhoods and metrics

Gaps in the synthesis tree are regarded as sequences of empty “frame slots”. In order to fill in an empty slot 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 which is similar, in principle, to the neighborhoods used by Wei and Levoy [17], 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 are shown in Figure 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 non-zero weights are renormalized to sum to one. We calculate the distance between corresponding slots as the distance between the joint and end-effector velocity vectors, using the 3D pose representation.

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.

### 3.3. 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) among the two.

Our strategy is first to find the best meeting point inside the gap, and then work our way towards it from both ends. In order to find such a meeting point for a gap at the coarsest level of the tree we employ an exhaustive search. We consider all the interior slots of a gap as potential meeting points, perform the synthesis towards each such point from both ends, and 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 neighborhood for each slot. Therefore, the exhaustive search is only done at the coarsest level. In the higher resolution levels we use the meeting point from the previous level as an initial guess, and only examine a small neighborhood of slots around this initial guess.

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 that was chosen in the previous level to fill in the parent of the current slot (see Figure 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.* [3].

### 3.4. Local smoothing

Some input sequences do not contain a sufficient number of repeating character poses. Thus, discontinuities may occur at transition points between sub-sequences of the original motion. The same problem was encountered in video textures by Schödl *et al.* [15].

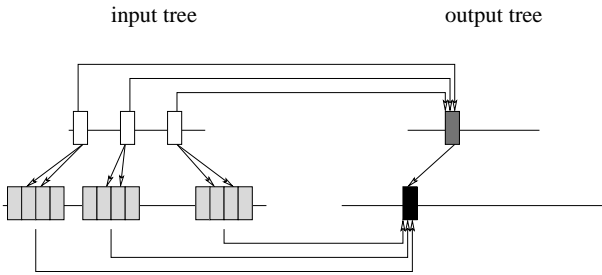


Figure 3. Acceleration of the synthesis process. The black frame is the current slot to be synthesized. The dark-gray frame is the black frame’s father. The white frames are the best three candidates for the dark-gray frame, and the light-gray frames are the white frames sons and their neighbors. The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

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. We start with a neighborhood of size two around the transition, and apply a Gaussian filter of width 5 for each frame in that neighborhood. If the neighborhood is smooth enough, we stop. Otherwise we increase the size of the neighborhood by two and apply the Gaussian filter again until we get a smooth transition.

To decide if a neighborhood is smooth enough, we first calculate the average acceleration of each joint and end-effector 3D location over the frames around the transition. Then after applying the Gaussian filter, we compare that average to the accelerations just before and after the transition. The neighborhood is considered smooth enough if the average is larger than the acceleration of both these frames.

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, the skeleton is never deformed by this operation. In practice, we are typically smoothing 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 can be seen in our results, this adaptive local smoothing scheme successfully eliminates visual discontinuities in the motion without introducing conspicuous artifacts.

### 3.5. 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 to start the synthesis from. This frame could come from anywhere in the synthesis sequence (but see the 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. This is because when synthesizing frames before/after those constraints respectively, the algorithm will be forced to find a meeting point at the frames between the first/last frame and that constraint. Since there are very few frames to choose from, the chosen meeting point might not be a good one.

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 forty frames. In other words, every forty frames the feet are roughly at the same position. Now suppose that we constrain frame number one of the input motion to appear as frame number one of the synthesized sequence, and also to appear at frame number sixty of the synthesis sequence. The resulting motion will then be forced to have either a single step cycle with sixty frames in it, or two short step cycles. This might not look natural, and may generate a transition with a large discontinuity (requiring a lot of smoothing to fix). Of course, if the input motion is more diverse, containing step cycles of varying lengths, this problem will be alleviated.

### 3.6. 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. A frame from each of the resulting animations is shown in Figure 4, and various statistics are reported in Table 1. All of the corresponding movies can be found at <http://www.cs.huji.ac.il/labs/cglab/research/lta>.

Name	Num. joints	Original length	New length	Num. constraints	Synthesis time (sec.)
Drunk walk (A)	23	512	1024	7	37
Drunk walk (B)	23	512	1024	8	43
High-wire	23	512	1024	6	32
Ballet walk	23	512	2048	9	81
Cool walk	23	512	1024	8	30

Table 1. Statistics for the synthesis examples. Times were measured in seconds on a 866 MHz Pentium III PC.



Figure 4. Frames from each of the animations we have generated. From left to right: Drunk Walk — The original sequence is performed by the middle character; the other two were synthesized by our method (drunk.mpg and drunk-path.mpg). High-Wire — The original motion is performed by the character in the back (highwire.mpg). Cool walk — The original motion is performed by the left character (cool.mpg and cool-path.mpg). Ballet walk — original straight line walk (ballet-original.mpg). Synthesized motion loop along an 8-shaped path (ballet-eight.mpg).

Figure 5 and the accompanying movie (drunk.mpg) demonstrate an application of our method to modify and double the length of a drunk person walking sequence. The plots 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 another 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 [4, 5, 12] from such a training set. Our method, however, is still able to generate visually continuous and naturally looking motions by locally smoothing over discontinuous transitions.

Our next example uses a high-wire walking input sequence (movie: highwire.mpg). Again, the basic cycle of this motion is quite complex and does not repeat itself too much, but our method succeeds in generating a longer sequence without noticeable discontinuities<sup>2</sup>.

Another example is a “cool walk” sequence (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.

Finally, we apply our method to a “ballet walk” sequence. In the original sequence the character is walking in a roughly straight line (movie: ballet-original.mpg). We con-

strained the synthesized sequence to begin and end with the same character pose and specified a new 8-shaped path instead of the original straight one. The result is an animation loop of a person walking along the new path (see movie: 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.

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. Thus, we believe that in practice our method should be used in conjunction with other techniques such as Kovar and Gleicher’s [9] work for foot-skate 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 into the animator’s toolbox.

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.* [3]. We also plan to consider other,

<sup>2</sup>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.

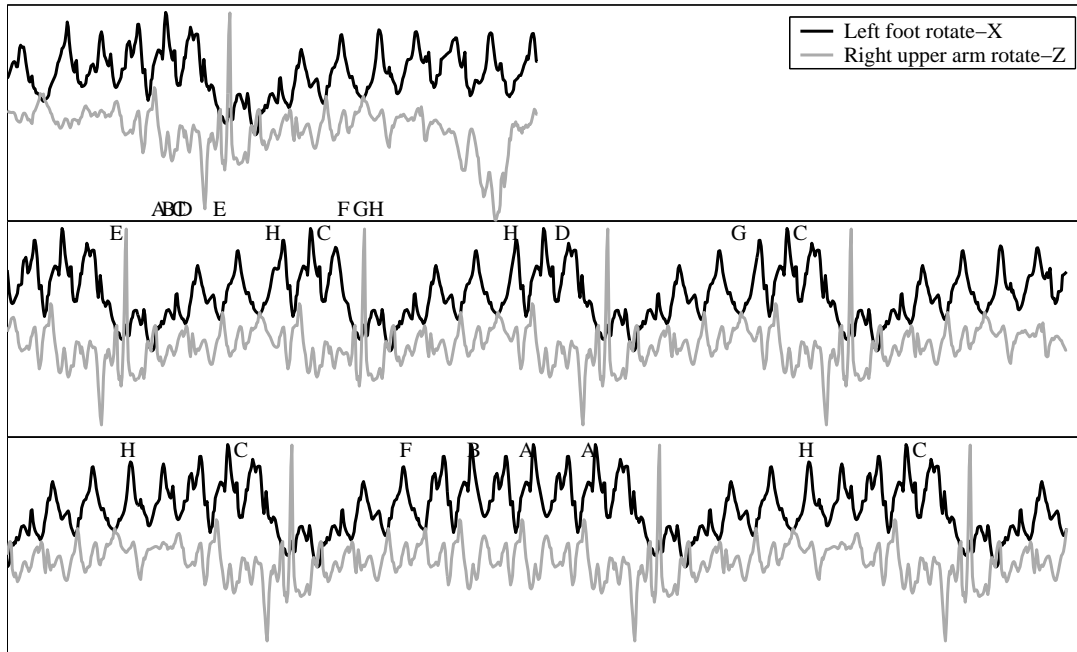


Figure 5. Drunk walk. The three plots illustrate the results of the synthesis by means of the angular trajectories of two joint angles. The top plot corresponds to the original sequence. The letters ‘A’ through ‘H’ indicate the positions of the constrained frames in the original sequence and in each of the two synthesized ones.

more sophisticated, types of constraints, such as the soft constraints described by Arikan and Forsyth [2].

### Acknowledgments

This research was supported 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 plugin 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. Maya 4.0. Modeling and animation software, 2001.
- [2] O. Arikan and D. A. Forsyth. Interactive motion generation from examples. In *Proc. SIGGRAPH 2002*, Annual Conference Series, July 2002.
- [3] Z. Bar-Joseph, R. El-Yaniv, D. Lischinski, and M. Werman. Texture mixing and texture movie synthesis using statistical learning. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):120–135, Apr./June 2001.
- [4] R. Bowden. Learning statistical models of human motion. In *Proc. IEEE Workshop on Human Modeling, Analysis, and Synthesis, CVPR 2000*, July 2000.
- [5] M. Brand and A. Hertzmann. Style machines. In *Proc. SIGGRAPH 2000*, Annual Conference Series, pages 183–192, 2000.
- [6] A. Bruderlin and L. Williams. Motion signal processing. In *Proc. SIGGRAPH 95*, Annual Conference Series, pages 97–104, Aug. 1995.
- [7] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *Proc. SIGGRAPH 2001*, Annual Conference Series, pages 341–346, Aug. 2001.
- [8] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *Proc. SIGGRAPH 2002*, Annual Conference Series, July 2002.
- [9] L. Kovar, J. Schreiner, and M. Gleicher. Footskate cleanup for motion capture editing. In *Proc. 2002 ACM Symposium on Computer Animation*, 2002.
- [10] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive control of avatars animated with human motion data. In *Proc. SIGGRAPH 2002*, Annual Conference Series, July 2002.
- [11] Y. Li, T. Wang, and H.-Y. Shum. Interactive motion generation from examples. In *Proc. SIGGRAPH 2002*, Annual Conference Series, July 2002.
- [12] L. Molina Tanco and A. Hilton. Realistic synthesis of novel human movements from a database of motion capture examples. In *Proc. IEEE Workshop on Human Motion*, 2000.
- [13] K. Pullen and C. Bregler. Animating by multi-level sampling. In *Proc. Computer Animation 2000*, May 2000.
- [14] K. Pullen and C. Bregler. Motion capture assisted animation: Texturing and synthesis. In *Proc. SIGGRAPH 2002*, Annual Conference Series, July 2002.
- [15] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa. Video textures. In *Proc. SIGGRAPH 2000*, Annual Conference Series, pages 489–498, July 2000.
- [16] M. Unuma, K. Anjyo, and R. Takeuchi. Fourier principles for emotion-based human figure animation. In *Proc. SIGGRAPH 95*, Annual Conference Series, pages 91–96, Aug. 1995.
- [17] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proc. SIGGRAPH 2000*, Annual Conference Series, pages 479–488, July 2000.
- [18] A. Witkin and Z. Popović. Motion warping. In *Proc. SIGGRAPH 95*, Annual Conference Series, pages 105–108, Aug. 1995.