Directable Motion Texture Synthesis

A Thesis presented

by

Ashley Michelle Eden

to

Computer Science

in partial fulfillment of the honors requirements

for the degree of

Bachelor of Arts

Harvard College

Cambridge, Massachusetts

April 8, 2002

Abstract

We present an algorithm for creating animation by keyframing, stylistic sequencing, random sampling, or by a combination of these methods. We first "learn" a motion model from existing animation data (such as motion capture), and optionally label specific frames as corresponding to specific "styles." The system can then generate a new motion in the style of the training data, but matching the pose constraints and/or style constraints specified by an animator.

Realistic motion in animation is difficult to synthesize—it often requires either specific physical calculations based on the character being animated, or it must be hand-generated or captured. Animations are used, however, in a number of applications that would benefit from a more general approach to creating realistic motion, e.g. in video games that require constraints over terrain, keyframes, and general style, but have no restrictions on specific paths, locations, or characters. We propose a method for motion synthesis that, given a character's hierarchical skeleton and bone lengths, will generate realistic motion similar, but not limited to, the content of sample motions. The new motion may also be constrained by keyframed location/orientation values and stylistic content.

Contents

Chapter 1

Introduction

Despite the widespread use of computer animation, the actual practice of animation by keyframing or motion capture remains quite difficult and time-consuming. Furthermore, an existing motion may be difficult to reuse without further time-consuming alterations. Although an animator or director will always be involved in the process, much research has focused on using automation to minimize the amount of tedious effort required.

One recent approach is to "learn" a motion model for a character, and then use this model when creating new motions. As long as the representation used for learning is sufficiently powerful, arbitrary new motions may be synthesized without requiring return trips to the motion capture lab. However, existing systems allow relatively little control to an animator, since they do not allow an animator to specify keyframe constraints or to specify which styles in the training data should be used when.

In this paper, we show how a learned motion model can be modified to support keyframing and stylistic constraints. We first extend the "motion texture" synthesis idea of Pullen and Bregler[16] to model the position and pose dynamics of input sequences. We then describe how novel motions can be sampled randomly from this model, or generated according to pose and style constraints specified by an animator. Pose constraints correspond to standard keyframe constraints; style constraints are a novel feature of our system that allow the animator to specify different motion styles or features for specific parts of the motion.

Our system offers substantial flexibility for animating from motion data: one may capture an actor performing various moves, then generate new animations of that actor according to specified keyframe and style constraints. High-level behaviors can be designed by hand (e.g.[1, 13]) to create scriptable and autonomous characters; our system could be used as a rendering substrate for these systems, with the actual motion determined by positional and stylistic constraints provided to our system by the behavioral controller.

Chapter 2

Previous Work

Realistic character motion has many applications, such as video games and movies. Most of the time the motions used are created by motion capture or by an animator. Motion capture is the process of recording the location and orientation of the joints of a character (human or animal). While this assures that the motion captured is realistic, it is sometimes hard to get certain motions that, for example require specific skills or, in the case of animals, get the character to perform a given motion at all. Also, often times the same basic motion is needed but with several stylistic or positional differences, e.g. jumping higher or longer, or walking on an emotional scale from happy to sad. Even if an animator attempts to capture each stylistically different motion, he/she has no control over what style of motion the actor actually performs. Handmade animation, while it assures stylistic control, is highly time intensive and not necessarily physically accurate. In addition, there is the same problem of having to create new animations for each new positional or stylistic difference.

Motion synthesis addresses these problems by automatically synthesizing new motions from sample motions. There is a wide variety of existing motion synthesis techniques: some retarget motion across characters, some create new motion through mathematical models, and some learn and interpolate existing data. We describe the previous work in motion synthesis, and describe our approach. First we summarize some of the most widely used methods of motion synthesis and their drawbacks. We will concentrate on example-based techniques for motion synthesis, and will generally describe our approach as an example-based analogy to texture synthesis. We will then briefly describe several methods for texture synthesis, explaining why we choose the one we implement for motion.

2.1 Spacetime Constraints

One of the more prominent approaches to providing high-level control of animation involves allowing an animator to specify some constraints and a physically-based energy function, and then searching for the motion that optimizes the energy, subject to the constraints[21]. Witkin and Kass first proposed synthesizing motions using such spacetime constraints. This approach is different than previous methods in that it poses a problem over an entire motion as opposed to individual frames. Synthesizing a motion per frame is problematic because it does not take into account the relationship between multiple frames. This is particularly applicable to the use of constraints—if the animator specifies that a character be in certain poses at specific times, individual frame techniques will not give natural transitions to and from the constraints.

The Witkin and Kass technique, however, requires the use of physical constraints. Using a physical model to synthesize motion often means taking into account the entire musculoskeletal structure of the character. Gleicher[7] first suggested using spacetime constraints without the physical constraints, thus allowing spacetime constraints to be applied to non-physical motions. He extended this non-physical approach to retargetting[8] motions from one character to another, thus introducing the concept of preserving styles in non-physical motions. There are several shortcomings of spacetime constraints in relation to retargetting, which, more generally, are problems for motion synthesis on the same skeleton. First, some visual properties, e.g. "grace," are too complex or impossible to code mathematically. Second, given an imaginary character, we still need to know its mass distribution and the physical laws of its imaginary world. Third, the representation of constraints and style in a motion may be different per motion. These shortcomings, coupled with the fact that they do not use a physical model, sometimes lead to motions that are not natural or do not follow the constraints specified. We desire a method that does not depend on the mathematical determination of style or the physical properties of the world or character, yet is physically natural.

Popović and Witkin[14] introduced a method for transforming character based animation sequences while preserving the physical properties of the motion. They use spacetime constraints over the physical model, yet with dynamics formulation so that the animator still has full stylistic control. Their method of motion synthesis is more specifically a motion transformation, which is useful in transforming a specific motion from one skeleton to another possessing slightly altered physical properties. For example, they might transform the motion of a human running to a human running with a limp by restricting the range of motion of the knee joint. They first construct a simplified physical model for the character, and fit the motion of the simplified model to the sample data. They then calculate the spacetime optimization that includes the body's mass properties, pose, footprint constraints, muscles, and

objective function. To edit the motion, they alter these parameters and constraints and recompute the motion. One major shortcoming of this method, despite the use of a physical model, is that the model simplification is done manually. In addition, these decisions directly affect the types of motions that can be made. For example, creating the motion of a person running while waving is not as simple as adding the waving arm to the running sequence. For example, the force of inertial changes in the moving limb need corresponding movement in the center of mass of the body. We desire a method that does not use a physical model, and that will be able to create and combine motions automatically.

Our approach is actually quite similar in spirit to physically-based animation. The main difference is that, instead of using a hand-made energy functional, we use a learned energy-function that can capture more nuanced style. The new goal is to create a motion that fits the constraints, while "looking like" the example data as much as possible. Our approach, however, does require suitable training data.

2.2 Learning Motion

One technique for synthesizing motion with style is to learn motion styles for sample data—i.e. use the example motions directly in synthesizing new motions, and not just for creating a model. Rose et al[4] suggested interpolating between existing motions, but one problem with this is that it requires a set of example motions performing the same basic sequence of events as the result motion. It is able to correctly interpolate to new styles, not necessarily given by an example motion, but it is unable to combine motions in a single frame, such as walking and waving, unless the motions appeared together in an example.

Brand and Hertzmann[3] described a system for learning the motion and style Hidden Markov Models (HMMs) from example motion, and interpolating to produce new styles. Because each entire timestep is treated as a variable, this technique can not learn the dynamics of individual parts of the body, and therefore can not apply different styles to different joints in a timestep. Again, the example of walking and waving does not work with this technique. Similarly, the animator can not specify constraints or styles over individual keyframes, and instead the new style created applies to the entire motion.

Pullen and Bregler[16] first suggested viewing motion as an image. They proposed synthesizing a set of motion curves for joint angles and translations statistically similar to the input data, but not necessarily the same. The new motion would appear realistic because it preserved the important features of the sample motion. They achieved this by using example motions that were highly repetitive, and decomposing the data into frequency bands. This method does not allow for stylistic constraints,

and works only for joint angles. Its advantages, however, are that it does not require a physical model, yet can create realistic looking in-place motion with constraints. We show how their approach can be extended to synthesize 3D motion rather than just joint angles, and to incorporate 3D positional and stylistic constraints. In addition, we do not directly calculate the frequency of the motion—in texture synthesis, this leads to functionality dependent on the levels of frequency in the sample data.

2.3 Texture Synthesis Techniques

As suggested in Pullen and Bregler, texture synthesis techniques can be applied to motion. Some methods of texture synthesis have limitations on the example textures used. Heeger and Bergen[10] put a random-noise image into example textures in order to create a resulting image with the same underlying texture but containing random variations. This method worked well for highly stochastic texture, i.e. textures without highly defined texture elements, but had trouble with structured patterns such as bricks. DeBonet[2] also uses a filter-based approach, where the texture is considered over a multiresolution pyramid. Each finer level of the texture depends on elements at the next coarser level, and randomness is introduced such that these dependencies are preserved. Again, this method works the best for highly stochastic textures. It also tiles the example texture in order to initialize result textures larger than the example, which assumes that the texture can be tiled naturally. Simoncelli and Portilla[15] use wavelet analysis to produce compelling results, but the algorithm is still better for more stochastic patterns, and has trouble with some high-frequency information.

Efros and Leung[6] propose a non-parametric sampling technique for texture synthesis, which creates new textures similar to examples. Their method is desirable because it is good at capturing statistical processes that are difficult to model mathematically. It makes no assumptions about the content of the example, and works well for a wide variety of textures. It also does not depend on the shape of the example or resulting texture, and is therefore good for constrained texture synthesis. Because it preserves local structure, it also creates no discontinuity between the constrained and newly synthesized areas.

The Efros and Leung technique thus presents a robust method for synthesizing a wide variety of natural textures. As with textures, natural motions are neither highly regular nor stochastic, and a synthesis technique that does not depend on the characteristics of the input will work the best. In addition, the hole-filling capabilities of Efros and Leung can be applied to constrained motion synthesis, in which the resulting motion has several poses already specified at some number of timesteps. Because Efros and Leung does not depend on the shape of the sampled or synthesized

texture, there may similarly be any number and placement of constraints on the result motion.

Subsequent to Efros and Leung, Wei and Levoy[20] have shown methods for accelerating the process, including multiresolution synthesis. Hertzmann et al[11] demonstrate a way to constrain the synthesis over defined styles, where each example image is divided into different stylistic areas, and the resulting image is constrained by some pattern of these styles, similar to "painting by numbers." When applied to motion, this method would enable an animator to create a motion where specific poses are not known or necessary, but the overall style of the motion can be described. Efros and Freeman[5] also suggest a method for combining the styles from two textures into one image, without having it look like the textures are simply overlayed.

There has also been some work done in the area of video textures[17, 18]. Here, the texture is 1D, where each frame in the animation is a point in the texture, viewed over time. Thus, it does not take into account the rotations and translations of individual joints.

Chapter 3

How I Learned To Stop Worrying And Love Motion Texture Synthesis

In this chapter, we describe the basic algorithm for motion synthesis, additional features such as constraints, and style. In each section, we will describe a new addition to the basic algorithm, and redefine the necessary equations.

3.1 Outline of Algorithm

We propose a method for synthesizing motion as if it were a texture. Instead of synthesizing pixels, we will synthesize quaternion values. Instead of placing the value at a 2D pixel location (x, y), we will place it at a 2D location $\langle joint, timestep \rangle$. A motion may be visualized as a set of joint values for every timestep in the animation. Each joint value is the value of a particular joint at a given timestep in the animation. For our purposes, we will synthesize natural motion, where the skeleton remains constant across the entire animation—i.e. there is no morphing. Since the skeleton is the same at every timestep, the joints in the skeleton are also constantly defined across the animation. If we number the joints in a particular order, we may use the same numbering at every timestep. Each joint number ℓ will refer to the same joint type, e.g. the mid-spine, at every timestep. Thus, if we think of the joints as numbered along one axis, and time along another, we will have a constant 2D frame of reference to describe the motion in. This 2D framework allows us to consider motion as a texture.

We present a motion synthesis technique based on the basic Efros and Leung[6] example-based texture synthesis algorithm, with additional multiresolution synthesis[20]

for speed, and stylistic constraints[11]. By the end, we will have described a method for motion synthesis that does not depend on a physical model or example images that exactly match the desired motion, yet allows for the specification of pose constraints and style.

The resulting motion is physically natural, both overall and over constraints. Motion texture synthesis uses both local and sample information. The local information will allow constraints (and style) to affect which sample timestep is chosen for a particular joint. Because the result motion is taken entirely from sample motions, the result is likely to be physically natural. At the same time, the result motion can be made up of motions that are not exactly specified in any one sample motion. For example, it may be a combination of several different sample motions. Also, because the motion is synthesized one joint at a time, a timestep may include joint values from several different timesteps in the samples. Because of the preservation of local structure, each joint value placed in the result motion at a given timestep (and a finite range of times around that timestep) will influence the choice of the rest of the joint values in that timestep. Thus the choice of one joint value will affect the choice of the next such that the motion in that neighborhood is the most natural.

Our goal for motion texture synthesis is to be able to synthesize a motion from existing motions. This means that the new motion will consist entirely of motion information from a training set. The motion information consists of information about the orientation and position of each joint in the skeleton at each timestep in the animation.

3.2 Pose Model and Motion Representation

There are several ways to represent information about the joints. First, we may give the exact xyz position in space of each joint. Because we want our motions to be invariant to rotations, this method is inefficient—if information from one motion were to be copied to another, the position of each joint in the skeleton would have to be optimally rotated and translated. In this case, the rotation and translation would be optimal in relation to the local position and orientation of the character in the result motion. For example, say we want to exactly copy the style of a sample motion to a new result motion, where the only difference between the motions is in the starting position and direction of the skeleton. In other words, we want the result motion to be different from the sample motion only by some constant rotation and translation. We would have to rotate and translate the xyz value of every joint at every timestep. More importantly, we also want to be able to preserve limb lengths and constraints on rotations, e.g. not allowing the head to rotate 360 degrees or the knee to bend in other than the usual direction.

Instead, we use a hierarchical skeleton structure with one root node, and all other joints arranged in a tree structure as its children. This way it is possible to determine the position of each joint only when it comes time to render the animation; each non-root joint is given a rotation, and the root joint is given a rotation and translation. The renderer is given these joint data, the skeleton hierarchy, and bone lengths. It may then apply the rotations for each joint in hierarchical order, starting with the root joint. Because we are hierarchically rendering the joints, we will always know the position of the parent of the current joint we are rendering. The rotation of the current joint represents the necessary rotation of the parent joint in order to get the orientation of the current joint. Since we also know the bone length, this will also give the position. In the case of the root joint, both a rotation and translation are stored—the rotation represents the rotation of the root joint from the world coordinates, and the translation represents the translation of the root position from the world origin.

Consider again the previous case where a result motion differs from a sample motion by a constant rotation and translation. Since this rotation and translation are the same across all joints, only the position and direction of the skeleton in relation to the world are different. With this new hierarchical representation, only the root joint has a value in relation to the world—all other joints are in relation to the local coordinate system. Therefore, in creating the result motion for this particular case, we would only have to change the value of the root node, and we could directly copy the values at all other joints.

We decided to use quaternions to represent rotations, since quaternions avoid any singular or special configurations[9]. A quaternion is a 4-dimensional number (w, x, y, z), where $w = \cos(\theta/2)$, and $(x, y, z) = \sin(\theta/2) * \vec{v}$. A rotation may be thought of as a rotation of θ degrees around an axis $\vec{v}$.

3.3 Motion

In this section, we introduce the different components of motion texture synthesis, and their corresponding algorithms. We begin by describing a single-scale motion synthesis algorithm using only joints, where each joint is represented by a quaternion. We then explain a more complicated (and accurate) version of the algorithm that includes the "root joint" in the motion information. The root joint is not actually a joint, but is the root of all joints in the skeleton hierarchy and is represented by both a quaternion and a vector describing its orientation and position in the world. Next, we introduce the concept of constraints for keyframes, and style. Finally, we describe a method of multiresolution synthesis that makes the algorithm faster and more robust.

3.3.1 Problem Statement

We will first describe the basic motion texture synthesis problem for single-scale synthesis over one target motion, where all joints are represented only by orientation. The basic motion texture synthesis problem is as follows: given a motion A', generate a new curve B' that appears similar to the example curve. Our single-scale motion synthesis problem is an extension of the Efros and Leung image texture problem[6] for processing motion. Here, instead of looking over pixels, we look over the joint orientations for each timestep. The following cost function is applied to each joint in the output motion B':

$$E(B') = \sum_i \sum_\ell \min_j d(B', i, A', j, \mathbf{w}) \tag{3.1}$$

This energy function states that we desire joint orientations for the neighborhood around i in B' that looks like the neighborhood around some j in A'. The neighborhood contains the joint orientations for the entire skeleton over some range of timesteps. Thus, the joint orientations for all joints in the skeleton contribute to the calculation of the distance function for a given joint $\ell \in L$, where L is the set of all joints in the skeleton. Because the joints in the skeleton will have different influences on each other joint, we need to weight joints differently. We define a vector of weight $\mathbf{w}$ of size $|L|$ for each ℓ, where each position ℓ' in $\mathbf{w}$ represents how much influence joint ℓ' has on ℓ.

The distance metric $d(B', i, A', j, \mathbf{w})$ gives a difference value between a neighborhood in B' and A'. Each neighborhood is represented by a set of K samples taken in unit length increments around a given timestep. The neighborhood around timestep j in A' is $A'(k + j)$, where $k = [-\frac{K-1}{2} \ldots \frac{K-1}{2}]$. (Note that k must be an odd value, since we will want $|k|$ to equal K.)

For now, assuming that all joint values are represented as quaternions and treated equally, the distance metric may be represented as:

$$d(B', i, A', j, \mathbf{w}) = \sum_m \sum_{\ell'} \mathbf{w_G}(k)\mathbf{w}_\ell(\ell') \| A'_{\ell'}(m + j) - B'_{\ell'}(m + i) \|^2 \tag{3.2}$$

where ℓ' represents some joint in L, and $m \in k$. ℓ' is different than ℓ because ℓ' represents some joint in A', and ℓ represents the joint in B' that we are currently calculating d for. The set of possible joints is the same for ℓ and ℓ' because they refer to the same skeleton. $\mathbf{w}_G$ is not related to the $\mathbf{w}$ passed into d, but is another weight vector relating to offset of k, which we will describe later. We will use the $\mathbf{w}$ passed into d, however, to determine how much each ℓ' influences ℓ. The weight vector $\vec{w}$ is represented in d by $\mathbf{w}_\ell$ since $\mathbf{w}$ is a vector of values of influence on joint ℓ. $A'_{\ell'}(m + j)$ and $B'_{\ell'}(m + i)$ represent the orientation value of joint ℓ', timestep

$m + j$ and joint ℓ', timestep $m + i$ in A' and B' respectively. Because $A'_{\ell'}(m + j)$ and $B'_{\ell'}(m + i)$ return quaternions, $\| \cdot \|$ is overloaded to mean the quaternion angle difference between $A'_{\ell'}(m+j)$ and $B'_{\ell'}(m+i)$. (The method for finding this difference is given in Appendix A. Note, from now on whenever we refer to a joint, timestep combination, it will be of the form $\langle joint, timestep \rangle$.)

We will now describe a more general version of the motion texture problem that incorporates the "root joint." The actual motion texture problem uses a slight variation on the above energy and distance functions. First, we must treat the root joint differently than the rest of the joints, since it is represented by an orientation and a position (as a quaternion and a point respectively), whereas the rest of the joints are just quaternions. Thus, we compare the root joint separately from the rest of the joints in the distance metric. In addition, we would like the comparison of the root joint to be invariant to rigid transformations in the xy plane—e.g. we would want an algorithm to create a newly synthesized animation that is not necessarily moving in the exact direction and position as any of the training motions. We only want xy transformations, because we assume that the character will always be moving with respect to the floor. (See Section3.3.3.) When comparing neighborhoods within a given A', the distance metric should be invariant to its location and orientation in the motion. Any two training motions will also have different rotations and translations, thus invariance to rigid transformations is also essential for combining motions. Thus, the new distance metric is:

$$\begin{aligned} d(B', i, A', j, \mathbf{w}) \;=\; \min_{R,t} \Big(& \big(\sum_m \sum_{\ell'} \mathbf{w_G}(k) \mathbf{w}_\ell(\ell') \| A'_{\ell'}(m + j) - B'_{\ell'}(m + i) \|^2 \big) + \\ & + \big(\sum_m \mathbf{w_G}(k) \mathbf{w}_\ell(rootquat) \| R A'_{rootquat}(m + j) + t - \\ & \qquad - B'_{rootquat}(m + i) \|^2 \big) + \\ & + \big(\sum_m \mathbf{w_G}(k) \mathbf{w}_\ell(rootpoint) \| R A'_{rootpoint}(m + j) + t - \\ & \qquad - B'_{rootpoint}(m + i) \|^2 \big) \Big) \end{aligned} \tag{3.3}$$

where R and t together define a rigid transformation, where R and t are an xy rotation and translation respectively, and ℓ' is now over all joints except for the root joint. Since the quaternion and point values of the root joint may be weighted differently, we separate them in 3.3. It will be described in more detail later how R and t are determined. Note that in the *rootpoint* term, $A'_{rootpoint}(m + j)$ and $B'_{rootpoint}(m + i)$ return points, so $\| \cdot \|$ is just a point difference.

The distance metric in 3.3 still does not take into account constraints and styles, but these will be introduced later.

One more generalization of the basic problem is that there may be more than one sample motion. Thus we would want to change the cost function in 3.1 to look over all sample motions. Thus, E would actually be written as:

$$E(B') = \sum_i \sum_\ell \min_{j,n} d(B', i, A'_n, j, \mathbf{w}) \tag{3.4}$$

where $n \in 1..N$, and there are N sample motions.

3.3.2 Algorithm

We will now describe the basic single-scale motion synthesis algorithm, i.e. without constraints or style. At the beginning of synthesis, the result motion has no joint information at any of the timesteps. Thus, we copy root node information from a random timestep in A' to B', and then copy over information for the first non-root joint from the same timestep in A' to B'.

The remainder of the motion is generated one timestep at a time—all the joints within that timestep are synthesized before moving on to the next timestep. It is done using nearest-neighbor sampling. The goal is to pick the value for each joint ℓ at the current timestep in B', $B'_\ell(i_{last}+1)$, that minimizes 3.4 while holding fixed the motion already synthesized.[1] Similar to texture synthesis[6], we do this by comparing the value v of the neighborhood around $i_{last}+1$ in B' to the value $v*$ of some neighborhood in A'.

For a given $\langle \ell, i \rangle$ in B' we want to optimize, we do the following: We iterate over all timesteps j in A' within the corresponding neighborhood. We only look at those timesteps, however, that are within half the maximum neighborhood length from the boundaries of the motion. (We say the maximum neighborhood, because the actual size of the neighborhood is determined by the information in B', based on the maximum bounds defined by k.) For each j we compute the new value of joint ℓ from the neighborhood in A' whose v^* best matches v. In other words, for each j in A', we want to minimize $d(B', i_{last}+1, A', j, \mathbf{w})$. Let the timestep j^* represent the

[1] It is not actually guaranteed that one step of this algorithm will reduce $E(B')$. It is possible that the algorithm could increase the energy at some steps. There is no guarantee of convergence. For example, say we find some minimal d at timestep i adjacent to timestep i'. Since i is in the neighborhood of i', the choice of value at timestep i will influence the result of the distance metric. Thus, it is possible that our choice at i will cause d at i' to not be fully minimized. This seems possible, for example, if there are artifacts in the sample motion. It might be the closest match to just follow some pattern of motion that varies from its defining cyclic patterns, but when the sample motion reaches the end, it will be hard to find an appropriate value that smoothly goes from the artifact back into the cyclic pattern. Despite this reservation, however, we are encouraged by the success of previous similar applications[6]. In addition, trapping local minima will be somewhat smoothed out with multiple passes.

timestep j whose neighborhood is the closest match. We set the value of $\langle \ell, j^* \rangle$ in A' to $B'_\ell(i_{last}+1)$. (See Figure 3.2 for a visualization.)

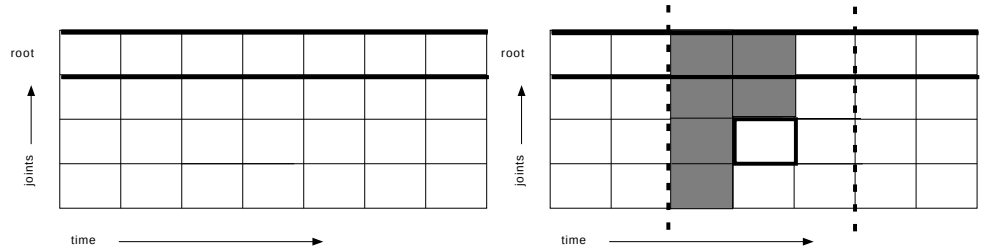


Figure 3.1: (left) A visualization of the motion texture. This would be a visualization of the data stored for one image, where each row contains the values of the same joint for each timestep, and each column contains the values of all joints in the same timestep.

Figure 3.2: (right) An example neighborhood for comparison. The dark square in the middle shows the timestep currently being synthesized. Because we perform synthesis in a consistent order—down each column, the already synthesized values will be those shaded in. Thus, the actual shape of the neighborhood will be the shape containing the shaded squares. The dotted lines represent the bounds of the neighborhood.

The method of neighborhood searching is similar to Efros and Leung[6] except that the neighborhoods must be compared under rigid transformations for the root node. Note that R and t are optimized before d. The optimal R and t are the rotation and translation representing the rigid transformation that best fits a neighborhood of root information in A' to a neighborhood of root information in B'. Thus we must first compute the optimal rotation and translation R, t that align the existing neighborhood sample $A'(k+j)$ to $B'(k+i)$ using standard point matching techniques[19, 12]. The actual algorithm will be described in more detail in Section 3.3.3.

In the basic case algorithm, a causal-neighborhood is also used: the points in each neighborhood in B' after $\langle \ell, i_{last} \rangle$ (assuming the joints are always calculated in the same order) have not been calculated yet. We thus omit them from the neighborhood, and from subsequent comparison with A'. More generally, any $\langle joint, timestep \rangle$ that has not already been synthesized in the B' will not be included in any comparisons, and thus the shape of the neighborhood will exclude their relative position. (See Figure 3.2 for a visualization of neighborhood shape.)

We also introduce randomness in the same way as Efros and Leung[6]—we uniformly sample from the set of s values for which the neighborhood distances are within some threshold of the optimal choice. One problem we ran into was that, since the

synthesis begins with data exactly copied from the sample motion, the minimum distance measurement will begin at 0. Our threshold, however, was some multiple of the minimum distance, $t(\min_j d)$. Thus, the threshold range would $= 0$, and B' would copy over the value for the joint at j^* to i. Then, when the $\langle joint_{next}, i\rangle$ was synthesized, it would copy from $\langle joint_{next}, j^*\rangle$. In other words, B' would just be some copied chunk of motion from A'. There would not even be rotation and translation of the root joint. In order to get around this problem, we decided to choose a new threshold function t'. If $\min_j d = 0$, we would use $t'(\min_j d) = t(\min_j d \neq 0)/2$.

The entire basic single-scale algorithm may be summarized with the following pseudocode:

function $SynthesizeMotion(A')$:
choose a random timestep j' in A'. initialize $\langle rootjoint, 0\rangle$ in B' with
  $\langle rootjoint, j'\rangle$ from A', for both the quaternion and point values.
copy over the value of the first non-root joint ℓ_0 at j' in A' to $\langle \ell_0, 0\rangle$ in B'.
for each i in $|B'|$, the desired length of B'
  determine the maximum range of times k in the neighborhood,
    where $k + i \in [0..|B'| - 1]$, and $|k| \leq K$.
  for each ℓ in L
    for each j in $|A'|$
      if $\ell ==$ *rootjoint*,
        determine the range $k' \subset k$ around j of consecutively filled root values
        compute the optimal R_j, t_j that aligns ${A'}_\ell(k' + j)$ to ${B'}_\ell(k' + i)$ in the
          xy plane
      if $\ell \neq$ *rootjoint*,
        find the neighborhood *win* around $\langle \ell, i\rangle$ of all filled non-root joints.
        compute d_j using timesteps and joints contained in *win*.
    $j^* \leftarrow arg\min_j d_j$
return B'

Note that the terms in d relating to the root joint may be calculated once and stored for use with every other joint in the same timestep. For each i, always look at the root joint first. This way the necessary rotation and translation information may be used in the calculation for each joint in the same i—the root sum, which is the last two terms of equation 3.3, is the same for every other joint in i. (Note that the root value at i will not be used in the root sum value calculated for the neighborhood around i.) When applying root sum to ℓ, the root sum is still multiplied by the appropriate matrix of influence value.

Once an initial motion is determined, it may be refined by multiple passes, or through multiresolution synthesis described in Section 3.6.

3.3.3 Optimal Rotation and Translation

As previously mentioned, we want to find the rotation and translation that gives the optimal rigid transformation of the neighborhood ${A'}_{rootjoint}(k'+j)$ to ${B'}_{rootjoint}(k'+i)$ for specific values of i and j. This transformation is only in the xy plane, since we assume all sample motions (and the result motion) are in relation to some floor at $z = 0$. Unless the character is for example jumping, it will have at least one joint "on the floor" (or near it, depending on where the joints are located on the skeleton) at all times. We assume that the floor is constant across all motions, and is given by $z = 0$. Thus, there is no need to solve for a transformation in the z direction. R and t, the xy constrained rotation and translation, may be found with the following equation:

$$\begin{aligned}(R^*, t^*) \quad = \quad & \arg\min_{R,t} \sum_{m'} \mathbf{w}_{m'}(\mathit{rootpoint}) \| R\mathbf{p}_{A'}(m', \mathit{rootpoint}) + t - \\ & - \mathbf{p}_{B'}(m', \mathit{rootpoint})\|^2 \\ & + \mathbf{w}_{m'}(\mathit{rootquat}) \| R\mathbf{p}_{A'}(m', \mathit{rootquat}) - \mathbf{p}_{B'}(m', \mathit{rootquat})\|^2 \end{aligned} \quad (3.5)$$

where $m' \in k'$, and where k' is the range of timesteps with consecutively filled root joint values in B'. (Note that for now, if the quaternion is filled in for a timestep, the point will be also. This will not necessarily be the case with more complicated versions of the algorithm, so for generality, a timestep is filled if either the quaternion or the point is filled in.) $\mathbf{p}_{A'}(m')$ and $\mathbf{p}_{B'}(m')$ represent the xy parts of the quaternion and point values given by ${A'}_{rootjoint}(m'+j)$ and ${B'}_{rootjoint}(m'+i)$ respectively. Because there are both quaternion and position values at every root joint, $\mathbf{p}_{A'}(m', \mathit{rootpoint})$ represents the position value in A' at timestep $m' + j$, ${A'}_{rootpoint}(m' + j)$, and so on. The quaternion and point values must be treated separately because they can have different weights, and because the quaternions are not translated. Note that because we use the same weights to calculate the optimal rotation and translation as we do in calculating the distance metric, each $\mathbf{w}_{m'}(\mathit{rootpoint}) = w_G(m') + \mathbf{w}_\ell(\mathit{rootpoint})$, and $\mathbf{w}_{m'}(\mathit{rootquat}) = w_G(m') + \mathbf{w}_\ell(\mathit{rootquat})$. Because the same root sum is used in calculating d for all joints in the same timestep, it is clear that $\mathbf{w}_\ell(\mathit{rootpoint})$ should be the same for all ℓ, and similarly for $\mathbf{w}_\ell(\mathit{rootquat})$.

We transform each quaternion $q = (w, x, y, z)$ into (x', y') by first changing q into a matrix mat_q, and multiplying mat_q by the vector $(1, 0, 0)$, $= (x_{new}, y_{new}, z_{new})$. We then let $(x', y') = (x_{new}, y_{new})$. (x', y') is actually a vector in the xy direction.

To get the point $p = (x, y, z)$ into (x', y'), we first take the 2D point (x, y) directly from p. Because the points are sensitive to translations, they have to first be mean subtracted before being passed into the point-matching algorithm. In order to mean

subtract, we first need to find the weighted mean (x_ω, y_ω) of all (x, y) in m':

$$(x_\omega, y_\omega) = \frac{\sum_{m'} \mathbf{w}_{m'}(rootpoint)(x(m'), y(m'))}{\sum_{m'} \mathbf{w}_{m'}(rootpoint)}$$

where $\mathbf{w}_{m'}(rootpoint)$ is the weight used in the distance metric d (and point-matching) for the root point values, and $(x(m'), y(m'))$ refers to the (x, y) values for a particular m'. We then set $(x', y') = (x, y) - (x_\omega, y_\omega)$.

The (x', y') for both the quaternion and the mean-subtracted point are then copied to $\mathbf{p}$ for every m', and the $\mathbf{p}$ for A' is matched to the $\mathbf{p}$ for B'.

Point matching means given two sets of point patterns, $\{a_i\}$ and $\{b_i\}$, $i = 1..n$, we want to find the rotation and translation that give the least mean squared error between the two[19]. The standard equation for finding the least mean squared error is

$$e^2(R, t) = \frac{1}{n} \sum_{i=1}^{n} \|b_i - (Ra_i + t)\|^2$$

.

Note that even though, for our purposes, the point patterns are all 2D, the rotation and translation is over 3D space. In our case, the point patterns consist of 2D representations of the quaternion and position at every root joint in the neighborhood. Only the points in a and b representing the root position are translated. Also, the points representing quaternion vectors may be weighted differently than the points representing the position values. Thus, a more accurate error function for our purposes is

$$e^2(R, t) = \frac{1}{2n} \left(\sum_{i=1}^{n} \mathbf{w}(i) \|b_i - (Ra_i + t)\|^2 + \sum_{i=n+1}^{2n} \mathbf{w}(i) \|b_i - Ra_i\|^2 \right) \tag{3.6}$$

where $n =$ *the number of timesteps in the neighborhood being point matched*, $i = 1..n$ in a_i and b_i correspond to position values, and $i = n + 1..2n$ in a_i and b_i correspond to each quaternion vector.

First we may solve for t by taking the partial derivative of $e^2(R, t)$ with respect to t, and setting the result equal to 0. We get that

$$t = \frac{\sum_{i=1}^{n} \mathbf{w}(i) b_i - R \sum_{i=1}^{n} \mathbf{w}(i) a_i}{\sum_{i=1}^{n} \mathbf{w}(i)}$$

Substituting back into $e^2(R, t)$, we see that the points in a and b corresponding to the root position are mean-subtracted. Thus, to find the optimal translation, we must first mean-subtract the 2D root points. (See Appendix B for a proof.)

The new equation after substitution of t is

$$e^2(R) = \frac{1}{2n}(\sum_{i=1}^{n} \mathbf{w}(i)\|b_i^{ms} - Ra_i^{ms}\|^2 + \sum_{i=n+1}^{2n} \mathbf{w}(i)\|b_i - Ra_i\|^2)$$

where b_i^{ms} represents each mean-subtracted root point value in b, and similarly for a.

3.3.4 Picking Weights for Joints

Each weight value $\mathbf{w}_\ell(\ell')$ refers to how much joint ℓ' influences ℓ. Consider each $\mathbf{w}_\ell(\ell')$ to be the value of some matrix of influence M at (ℓ, ℓ').

It is possible to have a skeleton where the position of one joint has widely varying influence on different joints. The rotation of the knee joint of a person, for example, has little influence on the rotation of the elbow joint, but much influence on the rotation of the hip joint.

Our implementation allowed for influences to be manually specified for the skeleton, or for default values to be set. It was unclear what influences to set even for a specific skeleton without much tweaking, so we tried to come up with a good general default M.

At first we tried choosing small values for $M[a, b]$, where $a \neq b$, and then some larger value for $a = b$. This worked fairly well for a while, but had to be constantly tweaked if the algorithm was altered, sometimes across different sets of sample motions.

It seemed for the most part that the influence of b on a was inversely proportional to the distance from b to a. We tried using Dijkstra's algorithm to find the shortest path from b to a, and then set the weights to be inversely proportional to this distance. This method gave much more natural motions. In order to get more natural motions, we also tried looking at the joints in different (but consistent) orders. In particular, we tried looking in breadth first and depth first order down the skeleton. The order in which the joints were synthesized over a timestep seemed to make no difference in the influence of one joint on another.

In performing a neighborhood comparison, we will be comparing the corresponding joints for all timesteps $k + i$ in B' to all timesteps $k + j$ in A'. The sum of the joint comparisons for each $m + i$, where $m \in k$, should be weighted differently, based on the offset of m from 0. Because i corresponds to the timestep in which the joint we are synthesizing belongs, we want all joint comparisons in i to be weighted more than those in timesteps that are not in i. More specifically, the farther away the timestep a joint comparison is from i, the smaller the weight it should be given. To do this, we set $\mathbf{w_G}(m)$ to be the value of m on the Gaussian falloff centered at 0.

3.4 Motion Synthesis with Constraints

We describe a way to include constraints in a motion texture. Constraints allow an animator to specify keyframes—in other words, orientations (and/or positions in the case of the root joint) for the skeleton at specific timesteps in B'. Any number of joints in a given timestep may be specified, as well as either/both the quaternion and point values of the root joint. We consider two different types of constraints: hard and soft. A hard constraint on $\langle \ell, i \rangle$ forces the final value of $\langle \ell, i \rangle$ to be the constrained value specified for $\langle \ell, i \rangle$. A soft constraint on $\langle \ell, i \rangle$ will be more of a suggested final value, whose importance is based on the weight of the constraint. We will refer to the originally constrained value on $\langle \ell, i \rangle$ as $C(\ell, i)$.

3.4.1 Soft Constraints

Soft constraints specify that the value of $\langle \ell, i \rangle$ in B' should pass near $C(\ell, i)$, but does not necessarily have to be exactly the same. Each soft constraint has a weight $w_C(\ell, i)$ representing the amount of influence $C(\ell, i)$ has on the calculation of the value of $\langle \ell, i \rangle$ in B'. (Note that for the root joint, there will be separate weights for the quaternion and position values.) This weight $w_C(\ell, i)$ will be used instead of $\mathbf{w}_\ell(\ell)$. (Note that $\mathbf{w}_\ell(\ell)$ refers only to the influence that the joint we are currently synthesizing has on itself.) In other words, if we are synthesizing $\langle \ell, i \rangle$ that has been constrained, $\langle \ell, i \rangle$ will already have an original value $C(\ell, i)$. Thus, when performing neighborhood comparison, it is possible to compare $\langle \ell, i \rangle$ with the corresponding $\langle \ell, j \rangle$ in A'. When weighting the difference at that entry in the neighborhood, we use $w_C(\ell, i)$ instead of $\mathbf{w}_\ell(\ell)$. We can still apply the appropriate weight from $\mathbf{w_G}$, which in this case will be $\mathbf{w_G}(i)$. Because constraints may be over root joints and non-root joints, soft constraints affect both the optimal rotation and translation and distance calculation.

The distance function with the new weights taken into account may be written as before, except with an added conditional term for the weight.

3.4.2 Hard Constraints

Hard constraints specify that the final value of $\langle \ell, i \rangle$ in B' should be exactly $C(\ell, i)$. Similar to soft constraints, if $\langle \ell, i \rangle$ is a hard constraint, we will be able to compare $\langle \ell, i \rangle$ with the corresponding $\langle \ell, j \rangle$ in A'. Because we will not want to change the value of $\langle \ell, i \rangle$ from its original constrained value, there is no need to actually perform this comparison. It is necessary, however, to still find the optimal rotation and translation, if ℓ is the root joint. The same root sum is used when calculating the distance metric for all other joints in timestep i, and the distance metric will be used if any of the non-root joints in i are not constrained.

3.4.3 Algorithm with Constraints

Note that the single-scale synthesis is not robust enough to naturally synthesize widely varying constraints, e.g. constraints that suggest orientations from several different sample motions. In the final version of the algorithm, we use a multiresolution synthesis algorithm to try to counteract this problem. (See Section 3.6.) We will describe the algorithm with constraints in terms of single-scale first for clarity.

The algorithm for motion synthesis with constraints is similar to the algorithm without constraints. In addition to the example motion A, a set of constraints and their weights are provided. We first initialize B to include these constraints. Now we have a motion with several $\langle joint, timestep \rangle$ values filled in, and with holes where no constraints were specified. Because d is performed over a finite neighborhood, it is possible that any given neighborhood will only include constrained values in one timestep. We want the motion to synthesize smoothly between timesteps, however, so we want information at each $C(\ell, i)$ to know about $C(\ell, i_{prev})$ and $C(\ell, i_{next})$, where i_{prev} and i_{next} refer to the timesteps adjacent to i with constrained values for joint ℓ. Thus, before beginning the rest of the basic synthesis algorithm, we linearly interpolate between each i and i_{next}. That way, when we look at a given neighborhood containing $C(\ell, i)$, it will now also include information about $C(\ell, i_{prev})$ and $C(\ell, i_{next})$, if they exist, even when $i_{prev}, i_{next} \notin k + i$. For this single-scale version of the motion synthesis algorithm, we assume that the root joint and first non-root joint at timestep 0 are constrained. A more general version may synthesize forward and backwards from the first constraint, not necessarily at timestep 0.

The size/shape of the actual neighborhood used is also different for constraints. The bound k is still the same, but it still applies that if a neighborhood point $B'_\ell(m+i)$ is omitted, then it is omitted from the corresponding $A'_\ell(m'+j)$, and thus from what we define as the neighborhood. The introduction of constrained and interpolated values means that the final neighborhood computed is not just causal, as it was before. If $\langle \ell, i \rangle$ in B was specified as a hard constraint, then we will want to keep $\langle \ell, i \rangle = C(\ell, i)$. Thus, for hard constraints we can just skip over the calculation of d. However, if ℓ is the root joint, then its value is also used in the calculation of the optimal rotation and translation of each $A'_\ell(k + j)$ to $B'_\ell(k + i)$. The size of the neighborhood used in the calculation of the root sum depends on the number of consecutive root values filled, and we synthesize all the joints in one timestep before synthesizing any joints in the next. Thus, the same rotation and translation will be used in the calculation of d for all subsequent non-root joints for timestep i in B', and d will be calculated unless all other joints in i are also hard constraints. We will still want to calculate the optimal rotation and translation of the root neighborhood centered at $\langle \ell, i \rangle$.

If $\langle \ell, i \rangle$ is a soft constraint, we will calculate the optimal rigid transformation and

distance metric like we did without constraints, but with $\langle \ell, i \rangle$ and any other filled values in the neighborhood. The weight used in d will be $w_C(\ell, i)$ instead of $\mathbf{w}_\ell(\ell)$. Note that for multiple passes, it is necessary to use the original constraint value $C(\ell, i)$ when comparing $\langle \ell, i \rangle$, but only when that comparison is for the synthesis of $\langle \ell, i \rangle$ itself—all other values, even if they are constrained, will be compared as they appear currently in B'. The newly synthesized value (per pass) of some $\langle \ell, i' \rangle$ may be used when it appears in a neighborhood, but $\langle \ell, i' \rangle$ itself is not actually being synthesized. This is slightly different for multiresolution, which we will discuss later in Section3.6.

The new distance metric for synthesizing a constrained $\langle \ell, i \rangle$ is the same as 3.3, but the $\|\|$ is overloaded with an added conditional—if ℓ is the same as ℓ', use $C(\ell', m+j)$ instead of $B'_{\ell'}(m+j)$, and $w_C(\ell, i)$ instead of $\mathbf{w}_\ell(\ell)$. Do the equivalent if ℓ equals *rootquat* or *rootpoint* as well.

A soft constraint with infinite weight acts the same as a hard constraint. A soft constraint with 0 weight still has influence on the motion, however, since the values between the keyframes are interpolated in the beginning.

If $\langle \ell, i \rangle$ exists but is not a constrained value (i.e. it was originally interpolated, or it is generated during a multiple pass), then we do not use its value in the neighborhood comparison.

3.5 Motion Synthesis with Style

Sometimes it is not essential that the skeleton passes through or near specific poses, but it is rather desired that different areas of B' contain different styles of motion. The user may define a set of τ styles such that each timestep in the sample motion A' is associated with at least one style in the set. Each timestep j in A' is labelled with $\mathbf{S}_{A'}$, where $|\mathbf{S}_{A'\gamma}| = \tau$, and each $\mathbf{S}_{A'\gamma}(j)$ indicates whether timestep j has style γ.

B' is also assigned a vector $\mathbf{S}_{B'\gamma}$ at every timestep i, where $\mathbf{S}_{B'\gamma}(i)$ which indicates whether B' should emulate style γ at timestep i. There is an additional vector $\mathbf{w}_{\mathbf{\Phi}}$ of size $|B'|$, where $\mathbf{w}_{\mathbf{\Phi}}(i)$ indicates the importance that $B'(i)$ emulate the styles indicated in $\mathbf{S}_{B'i}$. (There need not be any other information filled in B' for $\mathbf{S}_{B'\gamma}(i)$ and $\mathbf{w}_{\mathbf{\Phi}}$ to be filled.)

Also included, but general across motions, are τ vectors of size L, the number of joints in the skeleton, which indicate how much influence a particular style has on a joint. This weight of style γ on joint ℓ is referred to by $\mathbf{w}_{\mathbf{S}\gamma}(\ell)$.

The style and style weighting information for both A' and B' are passed in with the rest of the joint information for the respective motions. When d is calculated comparing $\langle \ell, i \rangle$ in B' to $\langle \ell, j \rangle$ in A', it will now include an extra term describing the sum of the difference in style between $B'(i)$ and $A'(j)$ for each style channel γ,

weighted by the influence of γ on ℓ, and the influence of style in general on $B'(i)$.

The new distance metric would thus be:

$$\begin{aligned} d(B', i, A', j, \mathbf{w}) \;=\; \min_{R,t} \Big(& \big(\sum_m \sum_{\ell'} \mathbf{w_G}(k) \mathbf{w}_\ell(\ell') \| A'_{\ell'}(m+j) - \\ & \qquad - B'_{\ell'}(m+i) \|^2 \big) + \\ & + \big(\sum_m \mathbf{w_G}(k) \mathbf{w}_\ell(rootquat) \| RA'_{rootquat}(m+j) + t - \\ & \qquad - B'_{rootquat}(m+i) \|^2 \big) + \\ & + \big(\sum_m \mathbf{w_G}(k) \mathbf{w}_\ell(rootpoint) \| RA'_{rootpoint}(m+j) + t - \\ & \qquad - B'_{rootpoint}(m+i) \|^2 \big) \Big) + \\ & + \big(\sum_\gamma \mathbf{w_\Phi}(i) \mathbf{w_S}(\ell) \| \mathbf{S}_{A'\gamma}(j) - \mathbf{S}_{B'\gamma}(i) \| \big)^2 \end{aligned} \tag{3.7}$$

including the conditional for constraints. Here, $\mathbf{S}_{A'\gamma}(i)$ refers to the information in A' at timestep j about style channel γ. Note that since $\mathbf{S}_{A'\gamma}(j)$ and $\mathbf{S}_{B'\gamma}(i)$ are just floats, $\| \cdot \|$ just means absolute value for style.

3.5.1 Algorithm with Style

Note that again, we will describe the algorithm with style (and constraints) in terms of single-scale synthesis. Actually, no additional changes to the algorithm are needed once the style information is stored and passed in with the motions. The only difference comes in computing the additional term in 3.7.

3.6 Multiresolution Synthesis

There are several problems with the single-scale algorithms previously described. First, we would like to use large neighborhood bounds K. Since the algorithm is approximately $O(|B'||A'||k||L|)$, increasing $|k|$ could be extremely computationally intensive. Also, to go from linearly interpolated constrained values to a natural motion, many passes are necessary. Because interpolated values are not necessarily natural, d may not find the best match in a sample motion, and B' could appear choppy. This behavior could be avoided with several passes, but the algorithm is already fairly slow as is. After trying multiple passes, we also determined that it would require more than just a few to propagate constraints and style smoothly.

Instead, we use a multiresolution synthesis algorithm, similar to that used in Wei and Levoy[20]. The idea is to generate a Gaussian pyramid of motions for both A' and B'. The pyramid $\mathbf{A}'$ for A' would be a set of motions ${A'}_\mu$, such that ${A'}_{\mu+1}$ is a subsampled and blurred version of ${A'}_\mu$, and ${A'}_0 = A'$. The pyramid for B' has an equivalent definition. The number of levels in all pyramids is ζ. We define the length of the motion at each level ρ to be half the length of the motion at $\rho - 1$.

3.6.1 Problem Statement

Here we will describe how we use the Gaussian pyramid of motions to synthesize our final motion B'. We will divide our explanation over each level of complexity in the single-scale motion synthesis algorithm—i.e. we will first describe multiresolution for motion synthesis without constraints or style, then with constraints, then with constraints and style. For each of these sections, we first explain what is done in general terms, and then describe the actual algorithmic details.

In multiresolution motion synthesis, we synthesize from the coarsest level ${B'}_{\zeta-1}$ to the finest level ${B'}_0$. Once the coarsest level has been initialized, it is synthesized using single-scale motion synthesis techniques. Then, each finer level ρ is synthesized using information from the motions both at level ρ and $\rho + 1$.

3.6.2 Basic Multiresolution Synthesis

Multiresolution synthesis requires that a pyramid of motions be created, both for the sample A' and the result B'. The motion at each level of the pyramid must also be initialized with the values corresponding to the analogous relationship of the motions between the levels. For basic motion synthesis, i.e. synthesis without constraints or style, the original B' is empty. Because the information in every level of the pyramid corresponds to the information in the level above it, all levels of the pyramid for B' begin empty.

In order to create the pyramid for A', we fill in each level from the finest ${A'}_0$, which is already given by A', to the coarsest. For each new level we initialize, we use information from the previous finer level, and fill from the first to the last timestep. Consider the creation of level ρ. We want to fill the level one timestep at a time, starting with timestep 0 and incrementally increasing. Because the information in level ρ is a subsampled and blurred version of the information in level $\rho - 1$, we traverse sequentially over every timestep in level $\rho - 1$ and try to copy information to the corresponding position in ρ. More specifically, for each $\langle \ell, j \rangle$ in ρ, we want to average the values at $\langle \ell, j-1 \rangle$, $\langle \ell, j \rangle$, and $\langle \ell, j+1 \rangle$, and place the resulting value in $\left\langle \ell, \frac{j}{2} \right\rangle$ in ρ. (For odd values of j, we just use $\frac{j-1}{2}$. Define j_{sub} as the timestep in level ρ that corresponds to timestep j in level $\rho - 1$.) We only average values in the

bound of the motion, e.g. ignoring $\langle \ell, j-1 \rangle$ if $j-1 < 0$ or $\langle \ell, j+1 \rangle$ if $j+1 > |A'_\rho|$, and averaging only over those values within the bound. The average is an average of quaternions, which we compute by linearly interpolating a quaternion halfway. When there are three values in the average, we first average the outer two quaternions, and then average the result with the middle quaternion.

Because there is not a one-to-one relationship between the information in $\rho - 1$ and ρ, there may already be a value at $A'_\rho(j_{sub})$. If this is the case, we want to average the already existing value with the new value we want to copy in.

This process is performed sequentially over all levels in the pyramid until the coarsest level is filled.

Algorithm without Root

We will first describe the algorithm used to synthesize B' treating all the joints the same—i.e. ignoring the *rootpoint* value, and the optimal rotation and translation of the root joint. Then we will explain the synthesis algorithm using the actual format of the root joint.

Once the pyramids are initialized, we want to synthesize each level in the B' pyramid from the coarsest to the finest. The coarsest level $B'_{\zeta-1}$ is synthesized using the standard single-sample motion synthesis algorithm. The only difference is that the sample motion used in comparison is $A'_{\zeta-1}$. We set the bound k of the neighborhood to be the same at all levels.

Once the coarsest level is synthesized, we synthesize the next finest level and so on up, until B'_0 is filled. To synthesize B'_ρ, where $\rho \neq \zeta - 1$, we want to use information from both level ρ and $\rho + 1$. We begin synthesis using the standard single-scale algorithm over B'_ρ. When synthesizing $\langle \ell, i \rangle$ in B'_ρ, however, we compare both the neighborhood around $\langle \ell, i \rangle$ at level ρ, and the neighborhood around $\langle \ell, i_{sub} \rangle$ at level $\rho + 1$. The neighborhood bound K_{sub} used when comparing level $\rho + 1$ (for the synthesis of level ρ), is $\frac{K-1}{2} + 1$, keeping in mind that K is always odd. Note that since $B'_{\rho+1}$ has already been completely synthesized, the neighborhood compared will be the maximum possible size. The two comparison values both contribute to the calculation of d for $\langle \ell, i \rangle$, though the comparison at level $\rho + 1$ may be weighted by w_{Multi}.

Thus, the distance metric for basic multiresolution synthesis where all joints are treated as quaternions is:

$$d(\mathbf{B}', i, \mathbf{A}', j, \mathbf{w}, \rho) = \sum_{m^\star} \sum_{\ell'} \mathbf{w_G}(k^\star) \mathbf{w}_\ell(\ell') \|\mathbf{A}'_{\ell'}(m^\star + j^\star) - \mathbf{B}'_{\ell'}(m^\star + i^\star)\|^2 \qquad (3.8)$$

where ρ is the level of the current $\langle \ell, i \rangle$ we are calculating d for. The offset $m^\star \in k^\star$, where $k^\star = k$ when we are comparing values in level ρ, and $k^\star = k_{sub}$ when we are

comparing values in level $\rho+1$. Similarly, j_M corresponds to either timestep j or j_{sub} whether we are comparing level ρ or $\rho+1$ respectively. Note that equation 3.8 will be performed over both A'_ρ and $A'_{\rho+1}$ for the appropriate timestep $j^\star$.

Algorithm with Root

The multiresolution algorithm when we use the correct format for the root joint is similar to the algorithm where all joints are treated as quaternions, except that we must also compare neighborhoods at two levels when finding the optimal rotation and translation. When synthesizing $\langle \ell, i \rangle$ where ℓ is the root joint, we need to compare the root separately from the rest of the joints in the distance metric, and we need to determine the optimal rotation R^* and translation t^*. Before, we found R^* and t^* by finding the optimal translation from the neighborhood $A'_{\ell root}(k'+j)$ to $B'_{\ell root}(k'+i)$. In multiresolution synthesis, we want to also compare the corresponding neighborhood in the next coarser level. Thus, if we are synthesizing level ρ, we also want to include the root values for the corresponding $\rho+i$ neighborhood, $k'_{sub}+j_{sub}$ in the calculation of R^* and t^*. We represent the new distance metric as

$$\begin{aligned} d(\mathbf{B}', i, \mathbf{A}', j, \mathbf{w}, \rho) \;=\; \min_{R,t} \Big(& \big(\sum_{m^\star} \sum_{\ell'} \mathbf{w}_{\mathbf{G}}(k^\star) \mathbf{w}_\ell(\ell') \| \mathbf{A}'^{\star}_{\ell'}(m^\star + j^\star) - \\ & \qquad - \mathbf{B}'^{\star}_{\ell'}(m^\star + i^\star) \|^2 \big) + \\ & + \big(\sum_{m^\star} \mathbf{w}_{\mathbf{G}}(k^\star) \mathbf{w}_\ell(rootquat) \| R\mathbf{A}'^{\star}_{rootquat}(m^\star + j^\star) + t - \\ & \qquad - \mathbf{B}'^{\star}_{rootquat}(m^\star + i^\star) \|^2 \big) + \\ & + \big(\sum_{m^\star} \mathbf{w}_{\mathbf{G}}(k^\star) \mathbf{w}_\ell(rootpoint) \| R\mathbf{A}'^{\star}_{rootpoint}(m^\star + j^\star) + t - \\ & \qquad - \mathbf{B}'^{\star}_{rootpoint}(m^\star + i^\star) \|^2 \big) \Big) \end{aligned} \tag{3.9}$$

where the notation is the same as in 3.3 and 3.8.

The calculation of R^* and t^* is now given by

$$\begin{aligned} (R^*, t^*) \;=\; \arg\min_{R,t} \sum_{m'^\star} & \mathbf{w}_{m'^\star}(rootpoint) \| R\mathbf{p}_{\mathbf{A}}(m'^\star, rootpoint) + t - \\ & \qquad - \mathbf{p}_{\mathbf{B}}(m'^\star, rootpoint) \|^2 + \\ & + \mathbf{w}_{m'^\star}(rootquat) \| R\mathbf{p}_{\mathbf{A}}(m'^\star, rootquat) - \mathbf{p}_{\mathbf{B}}(m'^\star, rootquat) \|^2 \end{aligned} \tag{3.10}$$

where the notation is again the same as in 3.8. For clarity, we let $\mathbf{p_A} \equiv \mathbf{p}_{A'^\star}$, and $\mathbf{p_B} \equiv \mathbf{p}_{B'^\star}$.

3.6.3 Multiresolution Synthesis with Constraints

When there are constraints given in single-scale motion synthesis, the constraints are copied to B', and interpolated before the rest of the algorithm is started. In the multiresolution case, the constraints themselves are copied to B'. This corresponds to the finest level ${B'}_0$. Because we begin synthesis on the coarsest level, we do not want to interpolate the values at the finest level. Instead, we need to subsample and blur B' at every consecutive level, where ${B'}_0$ contains only the constrained values. The method for creating the initial $\mathbf{B'}$ is the same as the method for creating $\mathbf{A'}$, except when a value is not filled in a ${B'}_\rho$, it is also ignored in the average. Once we have created the ${B'}_{\zeta+1}$, we may interpolate between the subsampled and blurred keyframe values at the coarsest level, and then continue with synthesis as in the basic multiresolution case.

Any hard constraints $C(\ell, i)$ in ${B'}_0$ are turned into soft constraints with a high $w_C(\ell, i)$. The $w_C(\ell, i)$ will also be averaged together, but at some fraction of their value in the previous level.

The distance metric for multiresolution synthesis with constraints is just 3.9 with the conditionals from the single-scale synthesis with constraints, and appropriate weight values depending on which level is being currently compared in the sum.

3.6.4 Multiresolution Synthesis with Style

Applying style to multiresolution synthesis means creating a pyramid for the style values $\mathbf{S}_{B'}$, the same way we create the pyramid for the motions. Each style γ is treated like a joint, and the number of timesteps in $\mathbf{S}_{B'}$ is the same as in B'. Then, when comparisons of style are done in d, the appropriate level of the pyramid is used. As with the motion itself, when calculating the style term in d, both the neighborhood in ρ and the smaller neighborhood in $\rho + 1$ are used. A pyramid is also created for the vector $\mathbf{w_\Phi}$ of $|B'|$ timesteps.

Thus, the distance metric for multiresolution synthesis with style is

$$\begin{aligned} d(\mathbf{B'}, i, \mathbf{A'}, j, \mathbf{w}, \rho) \;=\; & \min_{R,t}((\sum_{m^\star}\sum_{\ell'} \mathbf{w_G}(k^\star)\mathbf{w}_\ell(\ell')\|\mathbf{A'}^\star_{\ell'}(m^\star + j^\star) - \\ & \qquad - \mathbf{B'}^\star_{\ell'}(m^\star + i^\star)\|^2) + \\ & + (\sum_{m^\star} \mathbf{w_G}(k^\star)\mathbf{w}_\ell(rootquat)\|R\mathbf{A'}^\star_{rootquat}(m^\star + j^\star) + t - \end{aligned}$$

$$
\begin{aligned}
&\quad - \mathbf{B'}^{\star}_{rootquat}(m^{\star} + i^{\star})\|^2) + \\
&+ (\sum_{m^{\star}} \mathbf{w_G}(k^{\star})\mathbf{w}_{\ell}(rootpoint)\|R\mathbf{A'}^{\star}_{rootpoint}(m^{\star} + j^{\star}) + t - \\
&\quad - \mathbf{B'}^{\star}_{rootpoint}(m^{\star} + i^{\star})\|^2)) + \\
&+ \sum_{\gamma} \mathbf{w_{\Phi}}^{\star}(i^{\star})\mathbf{w_S}(\ell)\|\mathbf{S}_{A\gamma}(j^{\star}) - \mathbf{S}_{B\gamma}(i^{\star})\|^2 \qquad (3.11)
\end{aligned}
$$

with the conditional for constraints and the same notation as in 3.8, and $\mathbf{S}_A = \mathbf{S}_{A'_M}$, and $\mathbf{S}_B = \mathbf{S}_{B'_M}$ for clarity.

Chapter 4

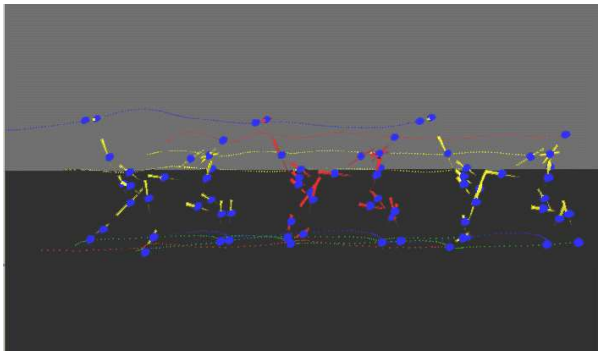
Results

Here we explain and discuss experiments, and suggest possible reasons for some of the problems.

4.1 Experiments

First we show how one sample motion can be used as a motion texture patch in order to make a similar motion of any length. In Figure 4.3, we generate new dog walking motions from the sample dog walking motion, 99 frames in length, given in Figure 4.1. Figure 4.3 is a walking motion of the same length, and Figure 4.4 is a longer motion, 150 frames. In each figure, the skeleton of the dog is shown at several timesteps. Even though Figure 4.3 is the same length as Figure 4.1, and both are of walking, there are random variations in the joint values between the two. Note that there would be more variation if we figured out a way to get around the problem of the original minimum difference being 0. Another way the motions are different is in the path they take in relation to the world—the motion path shown in the figures is only from left to right (or vice versa) because the "viewer" coordinates are rotated before capturing the image for clarity.

We can also see that the synthesis algorithm is not just taking chunks of timesteps from the sample for the result. Figure 4.4 is longer than the sample walking motion, yet the path of motion is smooth over all timesteps. Because Figure 4.4 is longer, we are guaranteed that it does not only contain consecutive timesteps. Figure 4.5 is a shorter walking motion, but created with multiresolution synthesis. In this case, we used two levels. The resulting walking motion is somewhat wobbly, however. We can see this in a still image by looking at the streamers that track the motion of a few key joints. Notice that the paths of the streamers in general are not as smooth as the paths of the streamers in Figure 4.1. In particular, we see that the motion



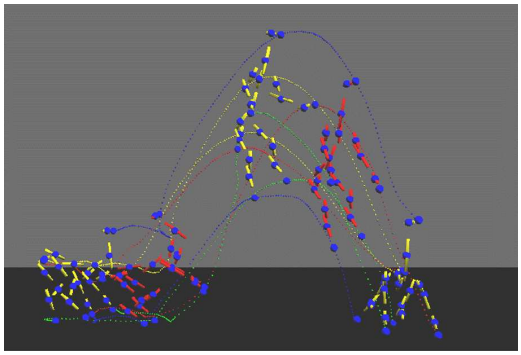


Figure 4.1: (left)Motion capture of a dog walking for 99 timesteps.

Figure 4.2: (right)Motion capture of a dog jumping for 50 timesteps.

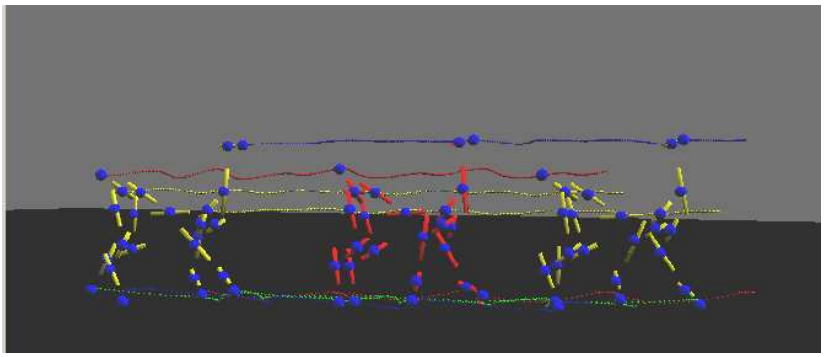


Figure 4.3: Dog walking motion synthesized over 1 level of resolution for 99 timesteps.

of the feet slide a little on the floor. One possible explanation for this behavior may be partly due to the small number of frames in the result motion, and the relatively large window size. Section 4.2 discusses this possible limitation in more detail.

Next we show several motions using keyframes. Keyframes specify constraints over the motion, may be either hard or soft, and be over any number of joints in the skeleton. Because the sample walking motion is fairly regular and cyclic, we show the effect of constraints on a more complicated example. In particular, we synthesized a walking+jumping motion from the sample walking motion, Figure 4.1, and a sample jumping motion, Figure 4.2. When we specified an entire body position, the resulting motions generally looked unnatural and forced. When we tweaked parameters specifying different weights and thresholds we would get better motions, but this method was inefficient. Essentially, we wanted a walking motion, followed

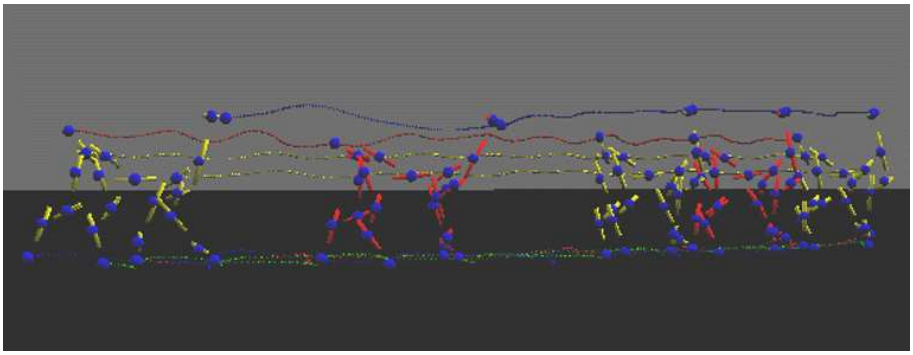


Figure 4.4: Dog walking motion synthesized over 1 level of resolution for 150 timesteps.

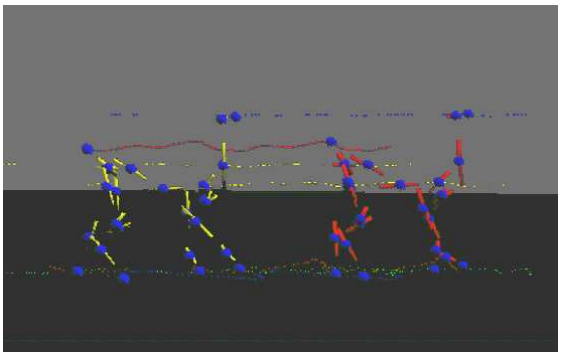


Figure 4.5: Dog walking motion synthesized over 2 levels of resolution for 50 timesteps.

by a jumping motion. Because walking may be characterized by the root of the dog moving insignificantly in the z-direction, and jumping may be characterized by the dog moving in an arc in the positive z-direction, we decided to keyframe based on the value at the root joint. More specifically, we took only the position value at the root joint and not the quaternion. The way the root of the dog twists along the walk+jump path is not characteristic of a walk+jump motion.

We set constraints at 4 timesteps in the result motion: 1) at the very beginning, 2) at the beginning of the jump, 3) at what we wanted to be the maximum z-value in the jump, and 4) back on the floor after the jump. The path of the root is shown in Figure 4.6, with each of the described keyframes corresponding with keyframes in the image.

The values we used for the constraints came directly from the sample motions. Note that this is not necessarily the best way to generate constraints. We wanted

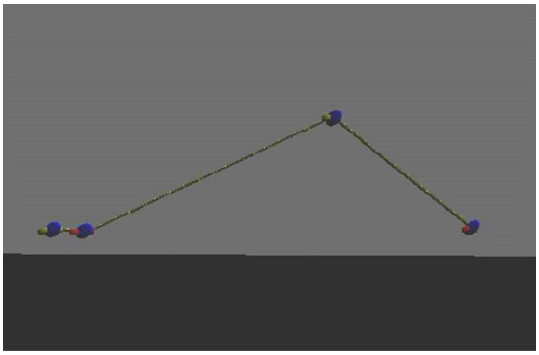


Figure 4.6: The interpolated motion of the keyframed root positions, corresponding to the motion of a dog walking then jumping, over 50 timesteps. The keyframes in the image correspond to the actual keyframed positions used in the synthesis of full-body motions.

values that were, for example in the walking part, a distance in z from the floor that is natural given the height of the dog, and where its root falls when it walks. The height in z of keyframe 3, however, does not have to be a height in the path of the sample jump. In fact, it is probably desirable to create new jump motions with different heights. Note, however, that the jumping motion in the walk+jump is not the same jump path as in the jumping motion—even though the root position values were taken directly from the sample jump motion, the number of timesteps between, e.g., keyframes 2 and 3, is not the same number of timesteps between the frames with the corresponding values in the sample jump motion.

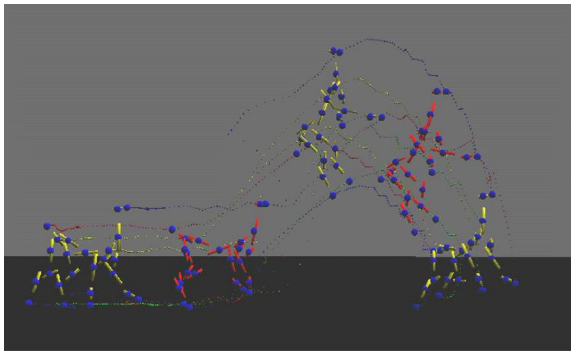


Figure 4.7: Dog motion synthesized using hard constrained root positions over 1 level of resolution and 50 timesteps. The keyframes correspond to the positional path that the root node would take if the dog were to walk then jump.

Figure 4.7 shows a synthesized motion using the keyframes from Figure 4.6 as hard constraints. Notice that it generally follows a walking then jumping path, but the motion is very wobbly and somewhat unnatural. Occasionally, the dog slides forward and backward, while still moving all of its limbs in a walking or jumping style. Even when the dog begins to jump, it just lifts into the air on its hind legs, as opposed to crouching first as it would do naturally. Also notice, by looking at the streamers in Figure 4.7, that the dog falls from the peak of its jump in a stepping fashion, and too slowly to be physically natural. The reason for this behavior may be as such: The reason for the slow fall may be because there are more timesteps between keyframe 3 and keyframe 4 than there are in the sample motion between the timestep with the same z height as keyframe 3, and a timestep with a position corresponding to the dog positioned on the floor. Even though there are many timesteps where the dog is positioned on the floor, there is only one case in the example motions where the dog goes from some other, more positive z height back to the floor. This problem might be fixed by a larger set of training motions, however. Because a jump is constrained in nature by the physical constraints of the surrounding world, it might be that given a certain height in z, and a direction and velocity of the dog, there is only a small range of timesteps during which the dog may land. The direction and velocity of the dog are given by the root quaternion and the relative positions of previous frames in the result motion. With a larger training sample, we would have more quaternion values and motion speeds from which to choose. In addition, the speed of the fall in Figure 4.7 would not be unnatural if the speed of the walking part were the same rate. Again, a larger set of training data would help this. Getting more training data is not necessarily the best solution, however. It might be desirable to create motions at new speeds without having to have a large training set. Another possible reason for the unnatural behavior might be that the constraints are hard constraints. Thus, the algorithm is forced to use those exact values, and can not choose other values that would lead to a more natural path of motion, e.g. a more natural falling path. This is probably not as large a factor in this case, however, since the keyframes came directly from the sample motions anyway.

Next we show a synthesized motion using the same keyframes from Figure 4.6, but this time as soft constraints. The resulting motion, given in Figure 4.8 is a bit more natural than in Figure 4.7, partly because the algorithm is not forced to use the values specified in the keyframes, and thus has more choice in choosing joint values. Again, the behavior in general is wobbly and unnatural, possibly for the same reasons suggested above. The weight given to the constraints was moderately large, meaning that the keyframed values were "strongly" suggested, but nearly as strong as if hard constraints were used. When using a soft constraint weight of 0 with the same keyframes, the resulting motion was quite similar to that in Figure 4.8. Even though the weight of the constraints was 0, they still had an influence because the

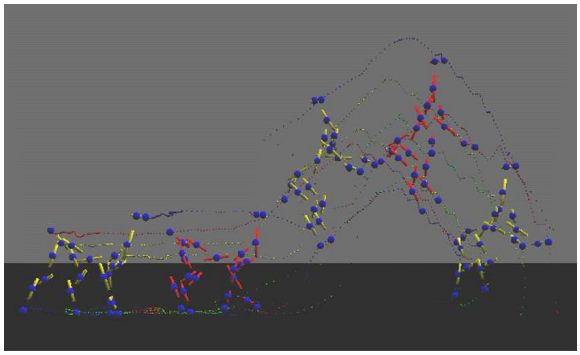


Figure 4.8: Dog motion synthesized using soft constrained root positions over 1 level of resolution and 50 timesteps. The keyframes correspond to the positional path that the root node would take if the dog were to walk then jump.

constrained values were used in the neighborhood search, and the result motion was initialized with the interpolated keyframe values.

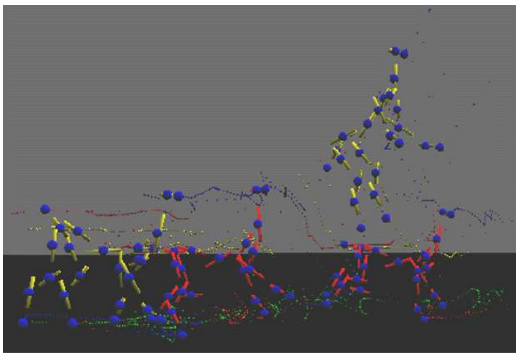


Figure 4.9: Dog motion synthesized using soft constrained root positions over 2 levels of resolution and 50 timesteps. The keyframes correspond to the positional path that the root node would take if the dog were to walk then jump.

All of the previous constrained motions were generated over one pass, at one level of resolution. If generated with multiple passes, some of the jerkiness and sliding artifacts might disappear. The fact that the synthesized motions looked even vaguely natural, if not also recognizably walking+jumping, after only one pass is reassuring. Figure 4.9 is a motion synthesized with the same soft constraints as in Figure 4.8, but over 2 levels of resolution. Notice that the motion is very jerky—the streamers following the joints go all over the scene. Possible reasons for this problem are explained

in Section 4.2. Generally, it is probably because the result motion is too small to deal with multiple levels of resolution where each level is half the length of the previous. In practice, we would want sample and result motions containing more frames than those we used in the experiments.

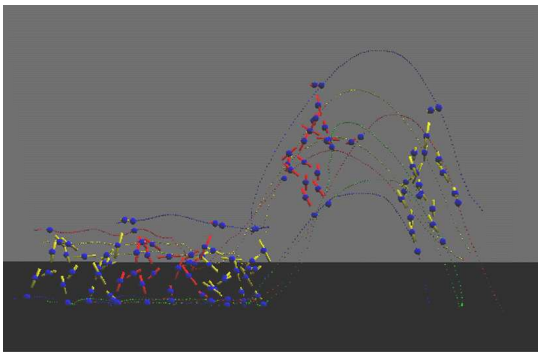


Figure 4.10: Dog motion synthesized with style constraints, over 50 timesteps. The style specified in the result motion corresponds to a walking style, walking and jumping, then just jumping.

Finally we show a synthesized motion given a specified style in Figure 4.10. First, we defined two style channels—walking and jumping. We then specified which styles were used at each timestep in each sample motion. For the walking motion, all timesteps were said to be just walking. For the jumping motion, the first 5 timesteps were designated as walking with no jumping, the next 4 as walking and jumping (at the point where the dog is still moving its legs in a walking fashion, but crouching down to get ready for a jump), and the rest as just jumping. We set the style for the result motion to be walking for the first 20 frames, walking and jumping for the next 3, and jumping for the rest. Coincidentally, the length of the result motion is the same as the length of the jumping motion. We weighted the influence of each style on each joint the same, and weighted the influence of style in the first 20 frames with a small value, and the weight of the rest of the frames with a large value. We did this because there are more sample walking cycles, i.e. motions where the dog is moving along the floor, than there are jump cycles. Thus, even if comparisons with neighborhoods in the sample jump motion are not a very good match, it is still very important that the values from the jump motion probably be picked in the timesteps where jumping is a part of the style. Notice that the synthesized motion is actually very natural and quite smooth. Even when the dog begins its jump, it gets into a crouching position first. It is just a bit shaky, as can be seen from the streamers in Figure 4.10 not being as smooth as in Figure 4.1 or Figure 4.2, but nevertheless is quite convincing.

4.2 Implementation Issues

Here we describe some of the issues we encountered when implementing the algorithm, which either caused us to rethink parts of the algorithm, or led to the results having strange behavior. In the cases of strange behavior, we suggest possible methods for getting around the problems.

4.2.1 Keyframing Smoothly

First consider a test where use two sample motions, one of a dog walking and one of a dog jumping. We keyframe three positions in B': a "walking" position at timestep 0, a "walking" position at timestep i_1, and a "jumping" position at timestep i_2. If we make only one pass, especially with a small neighborhood, it is possible that the resulting B' would just look like the dog walking, but with a sudden snap into the "jumping" position at timestep i_2. Even though we interpolated the values between the keyframes, they were not constraints themselves. When starting the synthesis, values from the walking motion give the best matches for d. This is still true when the synthesis reaches i_1, and the originally interpolated values begin to look somewhat like jumping. Because the values between timesteps i_1 and i_2 are not actually constraints but interpolated values, they can be overwritten and have no special weighting. Also, since the interpolated values do not necessarily match up very well to the jumping motion, but the timesteps previous to the i_1 match up well to the walking motion, it is likely that the walking motion will continue to give the best matches until we are actually synthesizing timestep i_2, at which point the jumping motion will suddenly give the best match.

This problem is fixed slightly with multiresolution, since at coarser resolutions the motion will be smoother, and the number of timesteps between constrained keyframes will be smaller. Also, this problem mainly exists when only the root point is keyframed. When the rest of the joints are also keyframed, or even just the root quaternion, then the other joints will also be interpolated and will thus be filled in the neighborhood when the value of the root joint at i_1 is calculated. Therefore, it is more likely that the value of the root joint at timestep i_1 will be a value from the jumping motion, and thus is also more likely that all subsequent joint values will be from the jumping motion, up until timestep i_2. We would still like correct functionality when only the root node is keyframed, however. It is possible that in application, only the general position of the character is known at different timesteps, and the orientation of the skeleton does not matter. Perhaps this can be simulated by making the constrained root joint have a high weight, and making the weight of the other joints small. One way we tried to simulate the importance of the other joints is to weight the quaternion values much more than the position values for the root joint

during multiple passes or multiresolution, and to weight the position values very high on the first pass (on a current level of multiresolution.) That way, on the first pass it will get some vague jumping motion correct—because the position of the skeleton is weighted heavily, it will at least move the skeleton smoothly through the air, instead of it suddenly snapping at timestep i_2. Then, on multiple passes, or when comparing a coarser level, it will correct which sample motion is used between i_1 and i_2, backwards from i_2. Because the quaternion values of the root joint in the jumping motion will be much closer to the quaternion value of the root joint at timestep i_2, if the quaternion is weighted more at the root joint, then the calculation of any joint in the neighborhood of i_2 will be highly influenced by the choice of the jumping motion at i_2. When more passes or levels of resolution are calculated, this influence will trickle back to timestep i_1.

4.2.2 Neighborhood Size

One potential problem is the determination of the size of the neighborhood in multiresolution synthesis. The way the algorithm is currently implemented, the maximum number of timesteps for a neighborhood will be K for any level being currently generated, and $\frac{K-1}{2} + 1$ for comparison of the next coarser level. One problem with this is that, because each level will have half as many timesteps as the previous, the number of timesteps in each level will get small very quickly. Thus, a large neighborhood size may lead to only a few possible neighborhoods to compare at the coarsest level, for synthesis where the motion at the finest level is not extremely large. One possible way to deal with this problem is to scale the neighborhood size at each level. This leads to a similar problem, however. Since halving the length of the neighborhood will lead to a very small neighborhood size very fast, the size of the neighborhood would have to be very large at the finest level.

4.2.3 Patterns of Motion

If the sample motions are not cyclic enough, there are not always good transitions from one motion to another, or even between frames in the same motion. For example, say we want a result motion to synthesize walking and jumping, given a sample walking motion and a sample jumping motion, similar to the examples in Section 4.1. Now say that the sample jumping motion has no walking frames at the beginning. It will be hard to find a good transition from a walking pose to a "beginning to jump" pose, and may even be impossible. Even if there are only a few walking frames at the beginning of the sample jumping motion, they might not be the same style as the sample walking motion, and therefore might not influence the distance metric enough to choose a walking pose from the jumping motion at or near the timestep where

the jump is keyframed to begin. Another problem is that the algorithm currently will not compare any timesteps in the sample motions that are not $K/2$ timesteps from the boundaries, where K is the maximum bound of the neighborhood. Thus, if these few walking frames appear at the beginning of the sample jumping motion, there is an especially small probability that there will be a good match. Instead, it is necessary to have many frames transitioning from one motion to another. That way the transition frames will not be as affected by the neighborhood boundary problem. Also, given a style of motion W, in this case walking, being transitioned to a new style J, in this case jumping, there is a greater chance that there will be similar poses in both the sample motion mainly featuring motion W, and the sample motion mainly featuring motion J. This may also be a major reason why the results we got were not as smooth as we would have liked. We attempted the above case of combining walking and jumping. The jumping motion, however had very few walking frames at the beginning, and those frames were mostly the transition from a walking pose to a "getting ready to jump pose," so there was only actually one frame that could be considered a walking pose in the jumping motion. One possible solution to this problem might be to use inverse kinematics to determine previous walking poses in the sample jumping motion.

Sample motions not being cyclic also affects the synthesis of singular styles of motion. For example, say we want to synthesize a long animation of a character jumping, in the simplest case with no constraints on position or style. Now say that the sample jumping motion only has one jump, and no transition to a new jump or new motion after the character lands. Because the result motion is long, the algorithm will attempt to synthesize many jump cycles. Because there is no transition after the jump, however, it will not be able to transition to the next jump in the result motion smoothly. We also ran into this same problem in practice.

4.3 Technical Details

The synthesis was performed on a 500MHz Pentium III processor with 256MB of RAM, and running Windows 2000. The code was written in Visual C++, and not at all optimized—it took about 300 seconds to synthesize a 100 frame motion at one level and one pass, with a neighborhood bound size of 9 frames, and one sample motion. The code could probably run a lot faster after even a few simple optimizations.

The rendering was done using a Java renderer from the Media Labs at MIT, slightly altered to output keyframes and streamers. The sample motion was motion capture data from Modern Uprising.

Chapter 5

Discussion and Future Work

Here we briefly discuss the implications of the results, and describe areas for future work.

5.1 Discussion

As seen from the experiments, the motions synthesized using the motion texture synthesis algorithm are at the very least recognizable as the style of motion desired. Synthesizing a new motion from only one sample motion, i.e. synthesizing a motion texture of a different size than an original sample motion texture, gives very natural and compelling results. When synthesizing motions as combinations of sample motions, the result can sometimes be a bit choppy and unnatural, but this is probably mostly due to a lack of sufficient training data, the small number of passes performed, and the lack of a systematic way to tweak parameters. In particular, synthesizing combinations of motions given keyframe constraints has a somewhat poorer performance, partly because it is probably affected more by the above factors. Most of the influence on the distance metric when doing constrained synthesis comes from weights within the joint-by-joint comparison, and so tweaking the weights correctly is essential. The style information is added completely separately to the distance sum, and thus changing the weight of the style has a more intuitive and direct effect. As such, combining motion synthesized with specified style values is quite smooth and natural.

Another reason for the relative choppiness of some of the synthesized motions may be that in most cases, only one synthesis pass was performed. In cases where more passes were performed, without multiresolution synthesis but instead just looping, the motion became smoother. Multiresolution synthesis should give the same results as multiple passes in a shorter amount of time, but for reasons such as short motions

and relatively large window sizes, synthesis at multiple levels does not perform well.

Nevertheless, there seems to be a possible reason for each example of poor behavior, which leads us to the conclusion that with some work, motion texture synthesis could be a useful and powerful method for generating natural motions without a physical model.

5.2 Future Work

5.2.1 Speed/Scalability Issues

Our motion synthesis algorithm is very slow because it compares every sample at every timestep for each timestep generated in the result motion. It would be useful to scale the process so that it does not look over every timestep, e.g. if some of the comparisons are pre-processed. Wei and Levoy[20] propose a method of vector quantization in order to speed up the Efros and Leung[6] texture synthesis algorithm. An analog of this to motion would have some problems, however, mainly because of the difference in the way we compute comparisons. In pixel texture synthesis, a comparison is just the difference between pixel values. In our algorithm, we first optimally rotate and translate each portion of the sample motion to best fit the area in the result we are currently generating. Because the rotation and translation themselves must be optimized locally, comparison value can not be simply stored in a vector tree.

5.2.2 Real-Time Synthesis

Real-time synthesis would be useful in applications where the exact motion is not known prior to when it has to be created. One major example of this is in video games. In many types of video games, the user has control over some main character. In the past, this control usually meant being able to choose actions over some finite set of possible actions. Thus, individual movements were scripted—if the user wanted his character to go from point A to point B, the character would move from point A to point B over some pre-designated path. As video games become more advanced, environments and non-user-controlled characters are becoming autonomous. The number of ways a user may now achieve a goal in the game is becoming infinite. Because most actions require some sort of motion, this means that the set of possible motions can no longer be pre-defined. Now a character that needs to get to point B no longer necessarily starts at point A, and if it does, then the path from point A to point B depends on the state of the environment (e.g. terrain, etc), and the placement of the other characters. This not only applies to motions that are a result of some

sort of user command, but also to motions that are reactions of the environment around the character. For example, consider the scenario where a character gets hit by another computer-controlled character. Assuming that the event is not scripted, the hit may occur at any place on the character's body. To maintain realism, the character should react differently based on where the hit occurs—if the character is hit in the shin, it would be unrealistic for it to grab its foot, but at the same time we do not want to have to store animations for every possible variation of the character grabbing a part of its leg. Essentially, for autonomous situations, it is necessary to be able to create motions on the fly. Most of the time the motions will follow some standard set of motions, e.g. moving to point B will require some sort of walking motion, and grabbing a part of a leg will require some sort of bending and grabbing motion. Thus, a system that synthesizes motions in real time from some example set would be highly useful.

One problem with the current method of motion texture synthesis for such real-time applications is that the algorithm is very slow. For the calculation of each joint at each timestep, comparisons must be made over the size of the window for every timestep. Thus, it is linear over the size of the result motion. Thus, an area for future work would be in getting the algorithm to work in real time. Wei and Levoy[20] suggested a method of vector quantization for texture synthesis which made the Efros and Leung algorithm much faster. This could possibly be applied to motion synthesis for the same result.

5.2.3 Motion Transfer

The idea of motion transfer is similar to texture transfer, as discussed in Hertzmann et al[11]. In texture transfer, an analogy between two images A and B is given, where the two images have the same content but different styles, e.g. a photograph and a watercolor of the same scene. Then, the algorithm is able to transfer this analogy to a new image A' and create B', e.g. it takes a new photograph and creates a new watercolor rendition of it. The extension to motion would be as follows: given a motion for one character, find the same motion for another character.

There are a couple of potential problems in automating this. First, there is currently no easy way to find a relationship between two characters without directly calculating the relationship between analogous joints. If the relationship between two characters A and B is determined, and one would like the same motion to be transferred between A and B, this could be done by adding another comparison term to every joint value, as was done with style. Now, instead of calculating a positional and stylistic difference when finding the best match, there would also be a comparison between the values indicating the analogy (e.g. rotation and translation) from the sample joint in A to the result joint in B. This type of transfer is similar to the Efros

and Freeman[5]. A more complicated case involves applying analogies: given the motion of a person walking and a dog walking, and the motion of a person running, find the motion of a dog running. How this would be done is not obvious.

5.2.4 Keyframing By Position

In the current version of the algorithm, the constraints must be given as quaternion values for the non-root joints, and quaterion/position values for the root joint. Often times, however, the constraints are only known as position values at the joints. It would be possible to go from position values to quaternions with inverse kinematics, but only if positions for all of the joints are known. If positions for only some of the joints are known, there are many possible orientations of their parents that would lead to their resulting positions. Because it is unclear what positions their parents would be in, it is unclear what quaternions would represent their rotation from the parent joints. Thus, one area of future work is in coming up with a way to set quaternion constraints of just a few joints based on positional information. Note that now we are able to set only some subset of joints based on positional information if that subset is of sequential joints from and including the root joint.

5.2.5 More Sophisticated Invariances

For the current version of the algorithm, we assume that all terrain is invariant in the z-direction—in other words, we assume that all motions will be naturally along the floor, unless the character itself exerts some upward force, and that floor will always be at $z = 0$. It may be, however, that the floor is not always at the same place. For example in the case of video games, the character may be travelling over rocky terrain, in which case the reference for the "floor" will constantly be changing. Because our optimal rotation and translation is only in the xy direction, we would have to alter our algorithm to include a new error metric and distance comparison.

Acknowledgements

First I would like to thank my advisor Shlomo for all of his help and guidance, and for being a wonderful teacher. I would also like to thank Aaron for being such a huge help during all stages of this thing—from algorithmic ideas to nasty bugs, he always found time to teach and help me out. I must also thank the Synthetic Characters group at the Media Labs, for piquing my interest in dog motion, providing me with a renderer and motion capture, and for their eagerness to help me see this research through.

I'd like to thank my parents for all their guidance, support and sacrifice, and providing me with so many opportunities to develop my interest in science and computers at an early age. I'd also like to thank my brother, Greg, for his reassuringly constant enthusiasm for computer graphics.

I'd also like to thank my wonderful friends who have been so supportive and understanding throughout this whole endeavor: to Ben who provided me with an outlet to vent and a DVD player; to Brady and Kiera, my roommates, whom I've barely seen in months; to Allie who let me take over her apartment in Seattle during the last crucial weeks; and to Jesse and Dylan for helping me edit and understand LaTeX.

I would finally like to thank my readers for taking the time to read this thing!

Appendix A

Quaternion Difference

We define the difference between two quaternions Q_1 and Q_2 to be the angle between them—i.e. the angle part of the quaternion Q_{diff} which, when applied to Q_1, will give Q_2. (Note that we want to make sure that Q_1 and Q_2 are normalized first.)

The relationship between Q_1, Q_2, and Q_{diff} is as follows:

$$Q_1 * Q_{diff} = Q_2$$

where $*$ means quaternion multiplication. Quaternion multiplication is defined as

$$Q_1 * Q_2 = \begin{bmatrix} w_1 \\ \vec{v_1} \end{bmatrix} \begin{bmatrix} w_2 \\ \vec{v_2} \end{bmatrix} = \begin{bmatrix} (w_1 w_2 = \vec{v_1} \cdot \vec{v_2}) \\ (w_1 \vec{v_1} + w_2 \vec{v_2} + v_1 \times v_2) \end{bmatrix}$$

Solving for Q_{diff} gives $Q_{diff} = Q_2 * {Q_1}^{-1}$, where ${Q_1}^{-1}$ is the inverse of Q_1. From the definition of quaternion multiplication, it is also evident that ${Q_1}^{-1} = (-w_1, x_1, y_1, z_1)$.

Because $w = \cos\frac{\theta}{2}$, the angle difference between Q_1 and Q_2 is

$$\theta_{diff} = 2 * \arccos(w_{diff})$$

where w_{diff} is the w element of Q_{diff}.

The difference θ_{diff} might not be the angle of the shortest path from Q_1 to Q_2, however. There are two quaternions that represent the same angle rotation about an axis, a quaternion Q and its antipode $-Q$, where $-Q = (-w, -x, -y, -z)$. We thus need to find Q_{diff} for both Q_2 and $-Q_2$. $Q_{diff\ anti} = -Q_2 * {Q_1}^{-1}$. The antipodal angle difference $\theta_{diff\ anti} = 2 * \arccos(w_{diff\ anti})$.

The angle that represents the shortest path will also be the smallest angle magnitude wise, so the final quaternion difference will be $\min(|\theta_{diff}|, |\theta_{diff\ anti}|)$.

Appendix B

Error Function

Given the equation

$$e^2(R,t) = \frac{1}{2n}(\sum_{i=1}^{n} \mathbf{w}(i)\|b_i - (Ra_i + t)\|^2 + \sum_{i=n+1}^{2n} \mathbf{w}(i)\|b_i - Ra_i\|^2)$$

solve for t by taking the partial derivative of $e^2(R,t)$ with respect to t, and setting the result equal to 0.

We get $0 = \frac{1}{2n}\sum_{i=1}^{n} 2\mathbf{w}(i)(b_i - (Ra_i + t))(-1)$. Simplifying this equation gives

$$\begin{aligned} 0 &= -\sum_{i=1}^{n} \mathbf{w}(i)(b_i - Ra_i - t) \\ \sum_{i=1}^{n}(\mathbf{w}(i)b_i) &= \sum_{i=1}^{n}(\mathbf{w}(i)(Ra_i + t)) \\ \sum_{i=1}^{n}(\mathbf{w}(i)(b_i - Ra_i)) &= t\sum_{i=1}^{n}(\mathbf{w}(i)) \end{aligned}$$

Now,

$$\frac{\sum_{i=1}^{n} \mathbf{w}(i)(b_i - Ra_i)}{\sum_{i=1}^{n} \mathbf{w}(i)} = \frac{\sum_{i=1}^{n} \mathbf{w}(i)b_i - R\sum_{i=1}^{n} \mathbf{w}(i)a_i}{\sum_{i=1}^{n} \mathbf{w}(i)}$$

Bibliography

[1] Blumberg, B. M., and Galyean, T. A. Multi-level direction of autonomous creatures for real-time virtual environments. *Proceedings of SIGGRAPH 95* (1995), 47–54.

[2] Bonet, J. S. D., and Viola, P. Poxels: Probabilistic voxelized volume reconstruction. In *Proceedings of ICCV* (1999).

[3] Brand, M., and Hertzmann, A. Style machines. *Proceedings of SIGGRAPH 2000* (July 2000), 183–192.

[4] C. Rose, B. Bodemheimer, M. C. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Application 18*, 5 (1998), 32–40.

[5] Efros, A. A., and Freeman, W. T. Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001* (August 2001), 341–346.

[6] Efros, A. A., and Leung, T. K. Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision* (September 1999), pp. 1033–1038.

[7] Gleicher, M. Motion editing with spacetime constraints. *1997 Symposium on Interactive 3D Graphics* (April 1997), 139–148.

[8] Gleicher, M. Retargetting motion to new characters. *Proceedings of SIGGRAPH 98* (July 1998), 33–42.

[9] Gortler, S. Quaternions and rotations. in unpublished chapter on computer graphics. url:http://www.fas.harvard.edu/∼lib175/lecture_fall_2001/chapters/quat.pdf.

[10] Heeger, D. J., and Bergen, J. R. Pyramid-Based texture analysis/synthesis. In *SIGGRAPH 95 Conference Proceedings* (1995), R. Cook, Ed., Addison Wesley, pp. 229–238.

[11] Hertzmann, A., Jacobs, C. E., Oliver, N., Curless, B., and Salesin, D. H. Image analogies. *Proceedings of SIGGRAPH 2001* (August 2001), 327–340.

[12] Horn, B., Hilden, M., Hugh, M., and Negahdaripour, S. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America. A, Optics and image science 5*, 7 (July 1988), 1127–1135.

[13] Perlin, K., and Goldberg, A. Improv: A system for scripting imteractive actors in virtual worlds. *Proceedings of SIGGRAPH 96* (1996), 205–216.

[14] Popović, Z., and Witkin, A. Physically based motion transformation. *Proceedings of SIGGRAPH 99* (August 1999), 11–12.

[15] Portilla, J., and Simoncelli, E. P. A parametric texture model based on joint statistics of complex wavelet coefficients. *Int'l Journal of Computer Vision* (2000).

[16] Pullen, K., and Bregler, C. Animating by multi-level sampling. *Proc. IEEE Computer Animation Conference* (2000).

[17] Schödl, A., Szeliski, R., Salesin, D. H., and Essa, I. Video textures. *Proceedings of SIGGRAPH 2000* (July 2000), 489–498.

[18] Soatto, S., Doretto, G., and Wu, Y. Dynamic textures. *Intl. Conf. on Computer Vision 8* (2001).

[19] Umeyama, S. Least squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence 13*, 4 (Apr. 1991), 376–80.

[20] Wei, L.-Y., and Levoy, M. Fast texture synthesis using tree-structured vector quantization. In *Siggraph 2000, Computer Graphics Proceedings* (2000), ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 479–488.

[21] Witkin, A., and Kass, M. Spacetime constraints. *Computer Graphics (Proceedings of SIGGRAPH 88 22* (1988), 159–168.