Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2004)
R. Boulic, D. K. Pai (Editors)

Momentum-based Parameterization of Dynamic Character Motion

Yeuhi Abe C. Karen Liu Zoran Popović

University of Washington

Abstract

This paper presents a system for rapid editing of highly dynamic motion capture data. At the heart of this system is an optimization algorithm that can transform the captured motion so that it satisfies high-level user constraints while enforcing that the linear and angular momentum of the motion remain physically plausible. Unlike most previous approaches to motion editing, our algorithm does not require pose specification or model reduction, and the user only need specify high-level changes to the input motion. To preserve the dynamic behavior of the input motion, we introduce a spline-based parameterization that matches the linear and angular momentum patterns of the motion capture data. Because our algorithm enables rapid convergence by presenting a good initial state of the optimization, the user can efficiently generate a large number of realistic motions from a single input motion. The algorithm can then populate the dynamic space of motions by simple interpolation, effectively parameterizing the space of realistic motions. We show how this framework can be used to produce an effective interface for rapid creation of dynamic animations, as well as to drive the dynamic motion of a character in real-time.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism-Animation

1. Introduction

Despite great advances in recent years, creating effective tools for synthesis of realistic human motion remains an open problem in computer animation. This is particularly true for synthesis of highly dynamic character motion such as running, leaping, jumping and other athletic and acrobatic maneuvers that frequently occur in feature special effects and video games. Synthesizing such motions can be challenging because any physical inaccuracies in these motions are particularly noticeable.

Both spacetime optimization and controller synthesis approaches have been proposed for direct synthesis of dynamic character motion. Although these methods do satisfy physical laws, they tend to appear overly smooth and at times robotic. Furthermore, these methods do not provide interactive control, often requiring considerable offline processing time before the animation sequence is generated. In addition, it is difficult to achieve a graceful degradation of realism for the purpose of greater control.

In contrast to direct synthesis, methods based on adaptation of motion capture data produce highly realistic motion, especially in the neighborhood of captured motion samples. They also run at interactive speeds, as they employ data interpolation techniques. Unfortunately, these methods require a large number of motion samples. If the animator wants to interactively control a specific parameter of the animation such as the landing foot position in a particular acrobatic stunt, the need for a large dataset is particularly pronounced: the interpolation techniques would require an already existing family of motion sequences where the only difference in motion is the landing foot position. Gathering such a datataset is not only laborious, but it also requires that the captured family of motions is similar in all other respects (e.g. other landing points, initial and final state, overall style) — an aspect that is quite hard to reproduce by real actors. In fact, the process of generating such parameterized motions is the most challenging aspect of data acquisition for video game production [Buc]. In addition, the animators often wish to create non-realistic motions that defy the laws

of physics, a space where motion capture simply fails to provide any samples.

We take the approach to acquiring similar motions is to adapt a single motion sequence several times to synthesize a family of motions that preserve physics constraints. Motions created in this manner can satisfy an animator's exact specifications with a minimum of deviation from the initial motion sequence. Ideally, we would like to use a minimal source of motion data, perhaps a single captured movement, to create a wide range of additional motions. Recently a number of dynamic motion adaptation methods have been proposed [PW99, ZH99, TSK02, SP04, SHP04], and the work presented in this paper falls into this category. In this paper, we describe the momentum-based motion editing technique. In contrast to the existing methods, our proposed framework is particularly robust to large-scale motion modifications. For example, we can adapt a forward leaping movement, to a collection of leaping movement in different directions including a backward leap, or a 360° leaping spin.

Using our motion editing framework, we show how a family of dynamic movements can be synthesized based on the animator's needs for interactive control. Because our family of motions samples the space widely, satisfies exact constraints, and otherwise deviates minimally from the original source sequence, we can use simple interpolation techniques to allow real-time exploration of this synthetic motion space. We describe a number of real-time animation tools that can be constructed using these synthetic motion families, such as interactive displacement of constraints (e.g. varying foot landing position), as well as inverse control examples such as the determination of the natural volleyball spike that would hit the ball arriving at a specific position in space. In addition, we describe how the same synthetic sampling/interpolation approach can be used to develop real-time controllers for leaping character motion, all synthesized from a single motion-captured leap.

2. Related work

Recent research in computer animation focused on techniques for remapping existing data to given specifications of a new scenario. In this paper, we build on the research in both physics- and interpolation-based motion editing methods.

2.1. Physics-based motion editing

Optimal trajectory methods introduced by Witkin and Kass [WK88] provide a powerful framework for enforcing dynamic constraints while searching for the most favorable motion judged by the objective function. Extending physics-based optimization to a full human figure, however, has presented a significant challenge mainly due to the non-linearity of the dynamic constraints, and sensitivity to the starting point of the optimization. The dependency on the initial point has been somewhat alleviated by starting out with the captured motion sequence. Popović and Witkin in 1999 developed a first method that transforms motion capture data while preserving physical properties [PW99]. They found solutions by performing optimizations on the reduced character model. More recently, editing motion capture data based on spacetime optimization has become a popular strategy for producing realistic character animations [RGBC96, SP04, SHP04]. These methods provide control for modifying data while retaining physically plausible properties of captured motion by restricting the optimization space with additional kinematic constraints (e.g. [RGBC96]), or by solving within the PCA-reduced space of motions [SHP04]. It has recently been shown that relying on simplifications of dynamic constraints is not necessary if proper scaling and estimation of joint angles, torques, and Lagrange multipliers are provided [SP04]. Our work uses a similar spacetime optimization framework. In contrast to other approaches, we formulate significantly simpler momentum constraints on a complex character model, without solving for muscle forces explicitly, similar to [LP02]. Since we do not compute internal torques for joints, scaling and convergence issues are less critical in our optimization framework.

Our physics-based motion editing approach is based on the momentum constraints introduced by Liu and Popović [LP02]. In that work, momentum constraints were used for synthesis of highly dynamic motion from simple animations that did not contain sufficient information to synthesize the full motion. As a result, transition poses had to be introduced to further restrict the optimization space. There are two main advantages of momentum constraints over the full dynamics constraints. First, since dynamic constraints are reduced to only global momentum patterns, we are solving for a much smaller set of unknowns, and over a much "better behaved" set of constraints. This allows us to find solutions quickly. Also, in our experience, these constraints do not suffer from many local minima, thus enabling us to find solutions significantly further away from the original motion. The second advantage of momentum constraints is that they encode more about the natural motion than just physical correctness. For example in natural motion, passive elements such as tendons and ligaments store and release energy during ballistic motion. To model this with a full dynamic system, one would have to include a complex muscle model. Momentum constraints effectively record the aggregate effect of the natural torque usage and energy storage/release in a specific momentum pattern. This additional information embedded within the momentum constraints ensures that adapted motion is not just physically correct, but that it also constrains the motion within the momentum exchange patterns observed in nature. In contrast to the original paper that introduced momentum constraints, our method applies momentum constraints directly on the motion capture data. Our algorithm does not require any additional pose constraints at

the transition points between flight and ground phases. Furthermore, we introduce a novel spline-based representation for the momentum patterns that can be used to intrinsically enforce the similarity between the resultant motion and the input motion.

Instead of formulating a physics-based optimization, dynamic filtering is an efficient alternative for motion editing of smaller amplitude. Per-frame based frameworks largely reduce the computation time, providing an interactive editing interface to the user [TSK02, SKG03]. Unfortunately, the per-frame approach means that animators can modify the spatial position of constraints, but not their position in time. Tak et al. applied Kalman filter to estimate an optimal pose for the current frame subject to the given constraints. The result of the estimation is then rectified by least-square-fit to ensure a physically sound motion [TSK02]. Shin et al. approximated the adjustment made to the original motion capture data by correcting the momentum of the character during flight and using the balance constraints on the ground [SKG03]. In general, these methods are geared toward the local modification compared to the overall motion, such as improving the balance, whereas our approach is able to handle global changes of the motion such as transforming a forward jump to a 360° backward spin jump. Another branch of dynamic filtering employs dynamic tracking [ZH99, PR01]. These methods combine motion capture data and dynamic simulation to retain human-like details from the data while presenting interaction with the environment. These methods produce motions that do not deviate significantly from the input motion, relying on the existence of captured motion that is similar to what the user intends to do.

2.2. Interpolation-based motion editing

Straightforward interpolation of joint angles usually fails to preserve physical realism from the original data. However, many methods have shown that small modification of the motion can be easily done by linear interpolation of joint angles [BW95, WP95, WH97]. Combining interpolation with kinematics constraints, Gleicher adapted original motion to a new character while maintaining environmental constraints such as foot contacts on the floor [Gle98]. A more sophisticated interpolation was presented using radial basis functions to blend motion sequences with various inverse-kinematic goals [RSC01] or different style [RCB98]. Unfortunately, data acquisition and post-processing for these methods present a significant challenge since motion sequences need to be carefully crafted so that they contain the same content yet different in style. Our approach only requires one single motion capture sequence as the seed. This seed is used to generate a family of motion sequences that parameterize the dynamic space.

Lee and Shin presented a multi-level B-spline representation by which they transform existing motion to satisfy desired constraints adaptively through direct manipulation [LS99]. Using B-spline representation, the motion edits can be limited to user-specified frequency bands, providing a more effective optimization framework. Our work adapts the idea of using spline-based representation to constrain the search of the optimization. We model the momentum curves by a B-spline representation which are fitted to the original motion so that the search space in the optimization is limited to solutions that have similar dynamic behavior of the original motion.

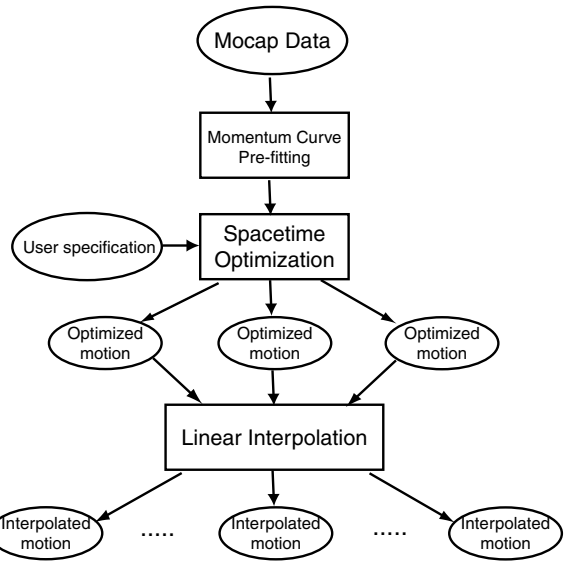

Figure 1: *System overview*

3. Overview

Our system is based on an optimization algorithm that can transform the captured motion to satisfy high-level user constraints while preserving physical realism. As input, the system takes a single motion capture sequence and the user-specified modification. We describe the algorithm in three separate components: Motion pre-fitting, optimization, and interpolation (see Figure 1). The pre-fitting optimizes a set of coefficients used to model momentum curves so that they are constrained to the similar shapes of the original motion. The system then formulates a spacetime optimization that solves for a new motion, where both high-level physical constraints and the user specification are met. With a family of such optimized motions that parameterize certain dynamic space, we can apply a simple linear interpolation to generate arbitrary new motion within the dynamic space in real-time.

4. Motion editing with momentum constraints

Our algorithm adapts the momentum-based constraints [LP02] for the task of motion editing. Instead of filling in

missing data, motion editing must solve the converse problem of preserving the original data while still satisfying animator-imposed constraints. There is no need for keyframing of any kind because the motion already starts in a good initial state. Any underlying physical model employed by the system must be flexible enough to precisely describe the initial state of the motion and, at the same time, rigid enough to maintain a semblance of the original motion throughout the editing process.

4.1. Motion pre-fitting

At the heart of our algorithm is a set of full-body angular and linear momentum curves. These curves constrain the edited motion to the realm of physical realism without the need to simulate expensive dynamical properties such as joint torques and contact forces. The momentum curves are parameterized by a set of coefficients that are pre-solved to closely match the input motion. The advantage of this approach is twofold. First, a good initial state of the momentum coefficients results in rapid convergence of the optimization. Second, the coefficients that control the shape of the curves can be fixed throughout the editing process, effectively performing a biased search for similar motions in the momentum space.

After the motion is captured using an optical system and processed to fit the character's skeletal structure, we employ the constraint detection technique described in [LP02] to partition the motion into ground-contact and flight stages. Since the the animator may at times wish to produce physically impossible jumps that are not constrained to the earth's gravity, and because the sampling rate varies for each input motion sequence, we also need to determine the time interval between two animation frames. Gravity and time step are directly related because we can equivalently choose to find the right gravitational constant that makes the motion realistic for a given unit time step. During free-fall stages, the linear momentum is only affected by gravity and the angular momentum remains constant. By observing that the center of mass (COM) of the model must follow a parabolic trajectory, $\mathbf{p}(t)$, we can compute the gravitational constant, $\mathbf{g}$, by solving a system of equations

$$\begin{aligned} \mathbf{p}(t) &= 1/2\mathbf{g}t^2 + \mathbf{v}_0 t + \mathbf{C}_0 \\ \mathbf{p}(t_n) &= \mathbf{C}_n \\ \mathbf{p}(t_{n/2}) &= \mathbf{C}_{n/2} \end{aligned}$$

where $t_{0..n}$ are time steps in the free-fall stage, $\mathbf{C}_{0..n}$ are corresponding values of the COM, and $\mathbf{v}_0$ is the unknown initial velocity of the COM.

When the body is in contact with external forces, the momentum curves can no longer be represented by a simple set of linear equations. Instead, we represent the momentum curves with a 3rd-order non-uniform B-splines for their flexibility and convenient knot based parameterization. In our spline representation, the first and last knots have duplicity 4 to ensure interpolation of the end points (see [FvDFH92]).

A defining characteristic of motion is the shape and magnitude of its momentum curve (see Figure 2). In the case of our spline representation, the control points determine the magnitude of the curve and the spacing of the knots influence the shape. We note that this formulation can capture a greater variability of momentum patterns than the previously used hardwired patterns [LP02]. This is especially important when dealing with motion capture data due to wide range of different maneuvers possible in the real world. To find a set of control points, $\{\mathbf{c}_i | i \in 1..k\}$, and knots, $\{\mathbf{u}_i | i \in 1..k+4\}$, that closely match the momentum pattern of the input motion, we solve the following constrained optimization problem for each momentum spline $\mathbf{S}(t, \mathbf{c}_{0..k}, \mathbf{u}_{0..k+4})$:

$$\min_{\mathbf{S}} \sum_{i=0}^{n} (\mathbf{m}_i - \mathbf{S}(t_i))^2 \qquad \text{subject to} \qquad \begin{cases} \mathbf{S}(0) = \mathbf{m}_0 \\ \mathbf{S}(n) = \mathbf{m}_n \\ \dot{\mathbf{S}}(0) = \mathbf{v}_0 \\ \dot{\mathbf{S}}(n) = \mathbf{v}_n \end{cases}$$

$$\mathbf{u}_i - \mathbf{u}_{i-1} < \varepsilon, \text{ for } i \in 1..k+4$$

where $\mathbf{m}_i$ is the momentum of the input motion at time step i, and $\mathbf{v}_i = \mathbf{g}M$, where $\mathbf{g}$ is the gravitational constant in the adjacent flight stage and M is the body mass of the character. In other words, we perform a least-squares regression over the momentum curve in the ground stage, while maintaining C^1 continuity through the transitions to the flight stages.

There are few exceptions to the problem described above. When there is no adjacent flight stage, we remove the constraint corresponding to $\mathbf{v}_i$ from the statement of the problem. Also, the constraint corresponding to $\mathbf{v}_0$ is entirely removed when pre-fitting the vertical linear momentum curve since the transition from a free-fall stage to a ground stage is typically dominated by impulsive forces, which are not C^1 continuous in the vertical momentum component.

4.2. Motion editing and optimization

In this section we discuss the process of editing motions using our system. As in [LP02] we model motion as an optimal dynamic process with a set of realistic constraints. In general terms, our condition for optimality is that the output motion be both as smooth, and as similar, to the original motion as possible. Constraints on the solution ensure that the character's limb do not bend unnaturally, that the character's feet do not pass through the ground, and that the character's full-body momentum curve follows the path of the pre-fit momentum splines. The degrees of freedom to be optimized are contained in $\mathbf{Q} \bigcup \mathbf{G}$, where $\mathbf{Q}$ is the set of joint angles through time describing the motion and $\mathbf{G}$ is the set of the control points controlling the momentum splines. In the initial state of the optimization, $\mathbf{Q}$ is a good initial guess at the target motion formed by linearly interpolating the original

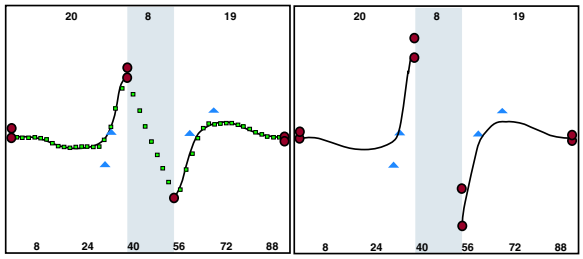

Figure 2: *Linear momentum of a jumping motion in vertical direction. The gray area indicates the flight stage. Left: The control points* $\{\mathbf{c}_i | i \in 1..k\}$*, visible as red circles, determine the magnitude of the curve. The spacing of the knots* $\{\mathbf{u}_i | i \in 1..k+4\}$*, visible as blue triangles, influence the shape. Pre-fitting phase solves for a set of control points and knots that closely match the momentum pattern of the input motion (shown as green squares). Right: During the spacetime optimization,* $\mathbf{u}_i$ *is held fixed while* $\mathbf{c}_i$ *is part of free variables. In this example, the optimized control points* $\mathbf{c}_i$ *result in a more energetic jumping motion.*

motion between user specified translations and orientations, and $\mathbf{G}$ contains the pre-fit momentum coefficients. In addition to the constraints and objectives used in [LP02], we also introduce a similarity objective and a pseudo balance objective as described in the following sections.

4.2.1. Similarity objective

The similarity objective is intended to keep the optimized motion as similar to the original as possible. We formulate this objective as the squared distance between the original vector of DOFs, $\mathbf{Q}_{init}$, and the solution vector, $\mathbf{Q}$. Each joint DOF is scaled by its natural bound. The energy function we wish to minimize is then,

$$E_s(\mathbf{Q}) = (\mathbf{Q}_{init} - \mathbf{Q})^2$$

4.2.2. Pseudo balance objective

Since we do not model the specific human preference to stay out of extreme leaning movements that in real life can often cause foot slipping on the ground, there are some instances when the resulting motion would leave the character unnaturally leaning without a means of support. To pull the optimized solution away from these unstable regions, we include a pseudo balance objective. The objective we use attempts to minimize the squared distance between the COM, $\mathbf{C}(t)$ of model in the first time-step, t_0, and last time-step, t_f, of the initial and final ground stages of the motion. For interior ground stages, we instead minimize the distance between the COM of the model in the middle frame of the stage, $\mathbf{C}(t_m)$, and the COM of the linearly interpolated input motion, $\mathbf{C}_{orig}(t_m)$, in the same frame. In other words, we introduce an additional objective function term, $E_b(\mathbf{Q}) = (\mathbf{C}(t_0) - \mathbf{C}(t_f))^2$, for the initial and final ground stage, and $E_b(\mathbf{Q}) = (\mathbf{C}_{orig}(t_m) - \mathbf{C}(t_m))^2$ for each interior ground stages. We find that the correct weight of these objectives do not vary much from motion to motion and, in fact, as long as the weight is well scaled w.r.t. other parts of the objective function, one value tends to "fit all".

4.2.3. Spacetime optimization

To summarize, the unknowns of our system, $\mathbf{Q}$ and $\mathbf{G}$, are the character DOFs and the control points for the momentum splines. Note that spline knots are omitted to maintain the similar momentum pattern of the original motion. The optimization enforces two types of constraints: environment constraints, $\mathbf{K}_e$, such as feet positions on the ground, and momentum constraints, $\mathbf{K}_m$. The following spacetime formulation finds the unknowns $\mathbf{Q}$ and $\mathbf{G}$ that minimize the objective function while satisfying all the constraints:

$$\min_{\mathbf{Q},\mathbf{G}} \quad E_s(\mathbf{Q}) + E_b(\mathbf{Q}) \quad \text{subject to} \quad \begin{cases} \mathbf{K}_e(\mathbf{Q}) & = & 0 \\ \mathbf{K}_m(\mathbf{Q},\mathbf{G}) & = & 0 \end{cases}$$

4.2.4. User interface

Our system provides several high level motion specification tools so that the animator never has to think of editing in terms of constrained optimization. First, motions are automatically partitioned into alternating flight and ground stages. Alternatively, the user can manually adjust the partitioning to make corrections. Next, the user manipulates ground stages with the mouse to translate their position and turns a dial to change the orientations as desired. The system treats these specifications as offsets from the original state of a ground stage. In other words, given the original translation, $\mathbf{q_T}$, and original orientation, θ, of the ground stage, the user specifies offsets $\Delta\mathbf{q_T}$ and $\Delta\theta$. The new translation and rotation of the ground stage is then altered to be $\mathbf{q_T} + \Delta\mathbf{q_T}$ and $\theta + \Delta\theta$, respectively. To form a good initial guess at the solution for the frames of the flight stages, the system linearly interpolates the offsets of the adjacent ground stages over each time step of the flight stage. The resulting motion is a crude approximation of the final result, but provides a good initial state for the spacetime optimization. The animator can also change the height of the trajectory in a flight stage by interactively shaping a visualization of the trajectory. This is particularly useful when creating non-realistic motion that defies gravity, as will be explained below. Once the user is satisfied with the edits, the optimization process takes between 1 to 5 minutes per motion. Alternatively, several motions can be generated together in a batch mode.

4.3. Populating the dynamic space of motions

In this section we describe a technique for generating a continuous ranges of physically plausible motions from a single motion capture sequence. The technique constructs an output motion in real-time by performing a simple weighted average over the DOFs values from a set of sample motions. A

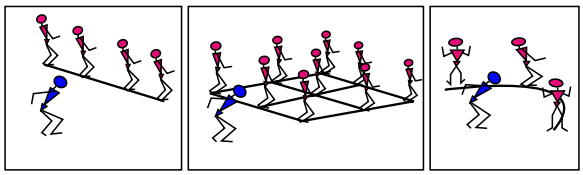

Figure 3: *Left: Line motion family that varies the translation of a ground stage along a line. Middle: Grid motion family that varies the translation of the ground stage along a 2 dimensional grid. Right: Circle motion family that varies both the translation and orientation of the ground stage along a semi-circle such that the orientation of the character is always aligned with the normal vector on the arc.*

family of motions can be populated from the input motion by systematically varying the position and orientation of one or more ground stages and then performing a sequence of similar optimization.

4.3.1. Motion families

We provide a user interface for the three most useful types of motion families(see Figure 3). The first type varies the translation of a ground stage along a line, the second type varies the translation of the ground stage along a 2 dimensional grid, and the third type varies both the translation and orientation of the ground stage along a semi-circle such that the orientation of the character is consistently aligned along the normal vector of the arc. The size of the sample space as well as the density at which it is sampled can both be adjusted as necessary. Other types of motion families can be easily added.

Once a motion family is populated, we are able to generate arbitrary intermediary motions by blending the nearest 2^n samples, where n is the number of dimensions in the parameterized space. We chose to use a simple linear blending method for several reasons. First and foremost, the algorithm is very fast and well suited to any application where the output motion must be generated "on the fly". Since motion families are produced offline, they can be as densely populated as necessary to increase the accuracy of the interpolation. Second, since the members of a motion family are produced by the same optimization setup, varying only in specific dimensions (e.g. landing positions, height, orientation, etc), it is often the case that they blend very well and need not be sampled very densely at all. In our results section, 9 samples is the most we ever required to adequately sample the dynamic space of a motion.

4.3.2. Foot glide

Although foot glide is among the most troublesome artifacts for most motion blending techniques, we find that it is imperceptible for both the line and grid motion families. However, when the global orientation and the translation of the motion are interpolated simultaneously, as is the case in the circle motion family, a very miniscule amount of foot glide becomes perceptible. A simple fix is to apply a per-frame inverse kinematic (IK) solver to slightly adjust the lower body to satisfy the positional constraints on each foot. Solving IK on the lower body not only has the effect of planting the foot firmly on the ground without changing the overall looks of the motion, but is also light-weight enough to converge in real-time, as the motion is being displayed.

4.3.3. Inverse control

So far we have shown how to populate the space of dynamic motion by interpolating between samples. Here we will discuss a more intuitive way of controlling these animations. In many applications the most important aspect to control is the position and time at which the character makes contact with an object in the environment. Consider the example of a soccer header motion, where it is required that the character's head always makes contact with the soccer ball at the correct moment in time. Starting from a single input motion we can generate an arbitrary header by creating a grid motion family that varies the translation of the landing stage. The joint configuration at each time-step in the output motion is then defined as a vector function $\mathbf{q}(x,y,t)$ of the landing position, (x,y), and the time-step, t. If we denote the position of the character's head by the function $\mathbf{h}(\mathbf{q})$, the problem of finding the motion that constrains the characters head to ball position $\mathbf{p}_c$ at time t_c, is reduced to that of finding values (x,y) such that $\mathbf{p}_c = \mathbf{h}(\mathbf{q}(x,y,t_c))$. This is, in turn, analogous to minimizing the energy function $E(x,y) = (\mathbf{p}_c - \mathbf{h}(\mathbf{q}(x,y,t_c)))^2$, which can be solved efficiently by a simple gradient descent method. The gradients are computed using finite differences. One caveat is that $\mathbf{q}$ is actually a piecewise function that performs a bi-linear interpolation of the 4 nearest sample motions. When one sample motion is replaced by another in the set of 4, $\mathbf{q}$ ceases to be C^1 continuous, causing convergence problems with the gradient descent method. A simple solution is to replace the linear blending functions $f(x) = x$ and $g(x) = (x-1)$ with smooth in/out functions such as $f(x) = sin^2(x)$ and $g(x) = cos^2(x)$, thereby maintaining C^1 continuity through the transitions.

4.4. Interactive control

One advantage of our motion generation algorithm is that it provides for a wide range of physically plausible animations in real-time. To demonstrate the full benefit of this approach, we have created a video game interface where the user controls the trajectory of a jumping character with a multi-directional control pad (see Figure 6). We start with a motion capture sequence of a character making two consecutive jumps. The interesting aspect of this motion is that the character must exhibit foresight in the motion of the first jump, so that the correct contact forces can be generate in the intermediate ground stage, to create the necessary momentum for the second jump. The spacetime approach is

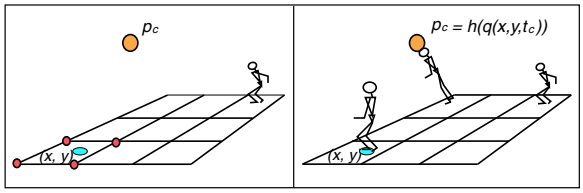

Figure 4: *In the example of the soccer header motion, the user specifies the contact point of the head and the soccer ball* $\mathbf{p}_c$ *at timestep* t_c*. Inverse control mechanism is able to immediately determine the four nearest neighbors among the sampled motions as well as their weights that interpolate the desired motion. An efficient gradient descent method solves for the landing position* (x,y) *by optimizing* $E(x,y) = (\mathbf{p}_c - \mathbf{h}(\mathbf{q}(x,y,t_c)))^2$.

ideal for editing such a motion because of the way it intrinsically models the interdependencies between different stages of a motion. Our approach inherits the same key benefit from spacetime, but allow us generate motions in realtime.

In this demonstration we wish to control the horizontal translation vectors of the first and second jumps, $\mathbf{d_1}$ and $\mathbf{d_2}$, respectively. First we generate a motion family by varying both the first and last ground stages along a 3x3 grid. The entire motion family then consists of 81 optimal motions resulting from permuting the 9 possible starting positions with 9 possible ending positions. This is necessary in order to sample the entire range of possible ground stage transitions between the two jumps. We are then able to populate the space between sampled motions by linearly interpolating the nearest neighbor optimal solutions. In this case, we have 4 dimensions in our sample space corresponding to the values of $\mathbf{d_1}$ and $\mathbf{d_2}$, making for a total of 2^4 (or 16) nearest neighbor motions. Therefore, we can express the output motion as vector function $\mathbf{q}(\mathbf{d_1},\mathbf{d_2})$, whenever $\mathbf{d_1}$ and $\mathbf{d_2}$ are within the bounds of the sample space. To make our demonstration even more interesting, we chain our jumping motion end to end, such that it continuously loops upon itself. This is done by blending the second flight stage of the first motion, $\mathbf{q_a}(\mathbf{d_{a1}},\mathbf{d_{a2}})$, into the first flight stage of the second motion,$\mathbf{q_b}(\mathbf{d_{b1}},\mathbf{d_{b2}})$. In order to make the blending work, we simply require that $\mathbf{d_{a2}} = \mathbf{d_{b1}}$. In order words, we require the length and direction of the blended jumps be the same.

The end result is an interactive jumping simulation where the user controls the direction that the character jumps and then sees the motion carried out in a physically plausible manner. Due to the foresight discussed earlier, the character must always have prior knowledge of the next two directions it will jump. This causes some lag time between when the user specifies a direction and when that motion will occur, but this is only natural given the deterministic nature of the ballistic motion.

Motion	Sequences	Frames	Time
Forward jumps	1	46	2 min
Two-step hop	1	49	3.5 min
360 degree spin	1	79	3.5 min
Volleyball slam	9	44	17 min
Interactive controller	81	56	4.5 h

Table 1: *Computation time for optimizations*

5. Results

The motion sequences in our demonstration were captured at 120 frames per second using an optical motion capture system. The character is composed of 18 rigid links and 43 degrees of freedom. $SO(3)$ rotations are expressed in exponential map representation. The mass distribution of the model is an appropriately scaled version of the population average as obtained from [dL96]. We used SNOPT [GSM96], a nonlinearly-constrained optimization package, for solving spacetime optimization, as well as for pre-fitting the momentum curves. Most edits shown in the accompanying video clips were done in less than 1 minute. The optimization process for each motion took on the order of 2 to 4 minutes to fully converge on a 2Ghz Pentium 4 machine (see Table 1).

5.1. Motion editing

Our system provides a set of UI tools to help the user rapidly specify modifications to existing motions. In a hopping example, the animator interactively manipulates the position, height, and orientation of each ground stage. The character must cover a longer distance, reach a greater height and assume a new orientation in the modified hopping motion, so she must lower her center of mass, lean farther to the right, and pivot slightly in preparation for the take-off. Despite these changes, the resultant motion remains stylistically similar to the original. To show that our system is capable of making drastic changes from the original motion, we edited the same hopping motion to exhibit a 360° spin followed by a 180° spin in the opposite direction(see Figure 5).

5.2. Real-time interpolation

In order to demonstrate real-time motion interpolation we modified a motion with two consecutive leaps. We let the user control the landing and take-off positions along an evenly spaced grid to generate a set of parameterized motions. Since the interpolation can be performed in real-time, we are able to generate a jumping motion with arbitrary take-off and landing positions within the parameterized space in an interactive fashion. Another example shows a soccer header motion observed to miss its target. First, we correct the motion by increasing the height of the jump to meet the

ball at the point of contact. Next, we use our editing algorithm to generate a motion family parameterized over the space of the landing position of the motion. By interpolating between the optimal motions, we are able to generate arbitrary intermediary motions where the character contacts the ball at any location within the sampled space, in real-time.

5.3. Inverse control

A more intuitive way to edit motion capture data with arbitrary positional constraints is to use our real-time inverse control mechanism. In the volleyball slam example, the user interactively specifies the position of the character's hand in mid-flight. Our system immediately determined the correct linear interpolation of 4 nearest neighbor samples to meet the positional constraint on the hand. The brightness of the sample motions on the floor indicates the weights associated with each sample. We used 9 sampled motions which are all edits of the same input sequence. The demonstration shows various slam motions being generated in real-time by using the trajectory of the volleyball to guide the character's motion. (see Figure 6).

5.4. Non-realistic motion

Our system can also be used to create a class of non-realistic motions that allow the character to exhibit super-human strength and to defy the laws of physics. Consider an example where we wish to edit a jumping motion to reach a higher mid-point in the same time span as the the original motion. The first observation to make is that this is physically impossible without altering the gravitational constant, which dictates the maximum rate at which the character returns to the ground from the height of the jump. In our system it is easy to alter the gravitational constant in one or more ground stages. Still, the character must gain the momentum required to achieve the specified height on takeoff and, subsequently, absorb the same amount of momentum on landing. This requires a super-human muscle strength, but since we do not directly model muscle forces, and we place no limits on their magnitude, our system can easily handle these imaginary circumstances. From the animators perspective, editing non-realistic motion is the same as editing any other motion. To increase the height of a flight stage, the animator simply manipulates a visualization of the trajectory of the motion in the flight stage to the required height, and then specifies whether the system should change gravity or, alternatively, the total time in the flight stage. If the animator chooses to leave the gravity unaltered, the system increases the length of the time-step in each frame of the flight stage and then continues the editing process as normal. In one example, we edited a forward jump into a 2-meter-long backward jump (see Figure 7).

6. Conclusion

This work builds on the research in both physics-based motion synthesis and interpolation-based motion editing approaches. In this paper we suggest that using physics-based adaptation to create motion samples for the purpose of data interpolation is perhaps a "sweetspot" between these two approaches. Once the dataset is created, this paradigm allows animators to interactively edit the realistic dynamic motion.

The primary contribution of this work is a new momentum-based method for adaptation of ballistic character movement. In contrast to previous dynamic-based adaptation methods, our framework can produce an wide range of motions that are significantly different from the original motion. Our method does not require model reduction, or a reduced motion space. Because we do not solve for the generalized forces for each joint angle, our method is also significantly faster than other physics-based transformation methods. This speed allows us to create a large number of motions within a reasonable time.

Once the family of parameterized motion samples has been generated, we describe an interactive framework where the animator can explore the space of realistic motions. We also show how the same framework can be adapted for inverse control. Finally, we show how real-time data-driven controllers for realistic human motion can be constructed from a single motion capture sequence.

Naturally, our framework does not handle all realistic character motions. It specifically applies to highly-dynamic motions with ballistic stages. We suspect that momentum-based approach would not be well suited for less energetic motions such as walking. Furthermore, the number of samples required is exponentially proportional to the number of dimensions, thus the current framework is hindered by the offline computation of a large dataset. There are several ways to facilitate the computation by taking advantage of the fact that we are solving a sequence of very similar problems. A more intelligent sampling strategy is essential for generalizing our approach to a multi-dimensional dynamic space. Because our model does not account for realistic muscle strength, and friction during ground contact there are some extreme cases which do not produce realistic motion. Adding heuristics such as balance during contact can to a large extent eliminate these problems.

7. Acknowledgments

Special thanks go to Mira Doncheva for her assistance with creating videos. We also thank Keith Grochow for his help with processing motion capture data. This work was supported by the UW Animation Research Labs, NSF grants CCR-0092970, ITR grants IIS-0113007, EIA-0121326, NSF REU grant, Alfred P. Sloan Fellowship, Electronic Arts Corporation, Sony and Microsoft Research.

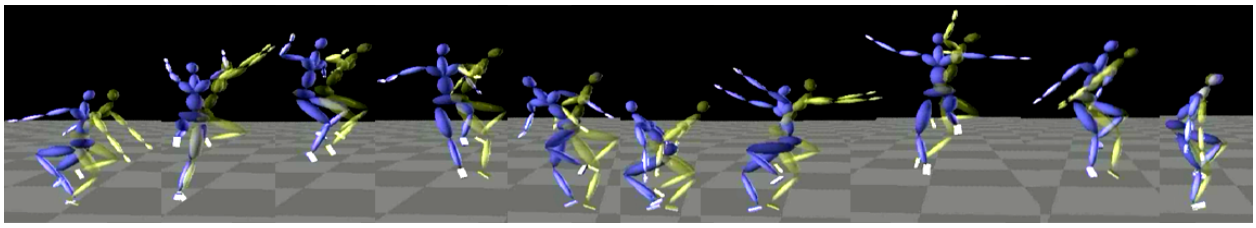

Figure 5: *A forward hopping motion (shown in yellow) is modified to make a 360 degree spin in the clockwise direction followed by a 180 degree spin in the opposite direction (shown in blue).*

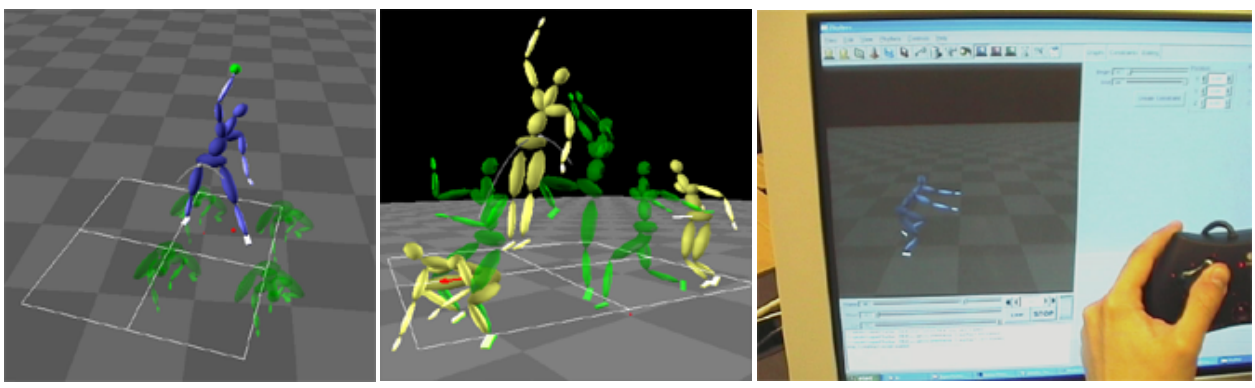

Figure 6: *Left: For a volleyball slam motion, the user interactively specifies the position of the character's hand in mid-flight. The system then determines the correct linear interpolation of the sampled motions to meet the positional constraint on the hand. Middle: The volleyball motion in profile. Right: The user interactively controls the direction the character jumps with a multi-directional control pad.*

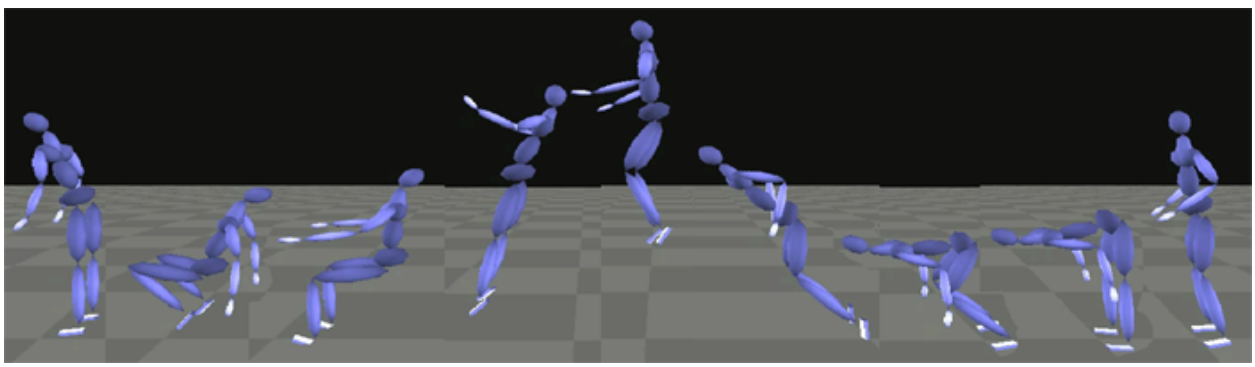

Figure 7: *The timeline of this animation goes from left to right. To demonstrate a motion that is impossible to achieve in the real world, the animator altered a forward jump to a 2-meter-long backward jump.*

References

[Buc] BUCHANAN J.: Personal communication. *Electronic Arts*.

[BW95] BRUDERLIN A., WILLIAMS L.: Motion signal processing. In *Computer Graphics (SIGGRAPH 95 Proceedings)* (Aug. 1995), pp. 97–104.

[dL96] DE LEVA P.: Adjustments to Zatsiorsky-Seluyanov's segment inertia parameters. *J. of Biomechanics 29*, 9 (1996), 1223–1230.

[FvDFH92] FOLEY J., VAN DAM A., FEINER S. K., HUGHES J.: *Computer Graphics: Principles and Practice*. Adidison Wesley, 1992.

[Gle98] GLEICHER M.: Retargeting motion to new characters. In *Computer Graphics (SIGGRAPH 98 Proceedings)* (July 1998), pp. 33–42.

[GSM96] GILL P., SAUNDERS M., MURRAY W.: *SNOPT: An SQP algorithm for large-scale constrained optimization*. Tech. Rep. NA 96-2, University of California, San Diego, 1996.

[LP02] LIU C. K., POPOVIĆ Z.: Synthesis of complex dynamic character motion from simple animations. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 408–416.

[LS99] LEE J., SHIN S. Y.: A hierarchical approach to interactive motion editing for human-like figures. In *Computer Graphics (SIGGRAPH 99 Proceedings)* (Aug. 1999).

[PR01] POLLARD N. S., REITSMA P. S. A.: Animation of humanlike characters: Dynamic motion filtering with a physically plausible contact model. In *Yale Workshop on Adaptive and Learning Systems* (2001).

[PW99] POPOVIĆ Z., WITKIN A. P.: Physically based motion transformation. In *Computer Graphics (SIGGRAPH 99 Proceedings)* (Aug. 1999), pp. 11–20.

[RCB98] ROSE C., COHEN M. F., BODENHEIMER B.: Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics & Applications 18*, 5 (Sept. – Oct. 1998).

[RGBC96] ROSE C., GUENTER B., BODENHEIMER B., COHEN M.: Efficient generation of motion transitions using spacetime constraints. In *Computer Graphics (SIGGRAPH 96 Proceedings)* (1996), pp. 147–154.

[RSC01] ROSE C. F., SLOAN P.-P. J., COHEN M. F.: Artist-directed inverse-kinematics using radial basis function interpolation. In *EG 2001 Proceedings*, Chalmers A., Rhyne T.-M., (Eds.), vol. 20(3) of *Computer Graphics Forum*. Blackwell Publishing, 2001, pp. 239–250.

[SHP04] SAFONOVA A., HODGINS J., POLLARD N.: Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. In *Proceedings of the 31st annual conference on Computer graphics and interactive techniques* (2004), ACM Press.

[SKG03] SHIN H. J., KOVAR L., GLEICHER M.: Physical touch-up of human motions. In *Pacific Graphics 2003* (Oct. 2003).

[SP04] SULEJMANPASIC A., POPOVIĆ J.: Adaptation of performed ballistic motion. *ACM Transactions on Graphics* (2004).

[TSK02] TAK S., SONG O.-Y., KO H.-S.: Spacetime sweeping: An interactive dynamic constraints solver. In *Proceedings of the Computer Animation 2002* (2002).

[WH97] WILEY D. J., HAHN J. K.: Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications 17*, 6 (Nov./ Dec. 1997), 39–45.

[WK88] WITKIN A., KASS M.: Spacetime constraints. In *Computer Graphics (SIGGRAPH 88 Proceedings)* (1988), pp. 159–168.

[WP95] WITKIN A., POPOVIĆ Z.: Motion warping. In *Computer Graphics (SIGGRAPH 95 Proceedings)* (Aug. 1995), pp. 105–108.

[ZH99] ZORDAN V. B., HODGINS J. K.: Tracking and modifying upper-body human motion data with dynamic simulation. In *Computer Animation and Simulation '99* (Milano, Italy, September 1999), Eurographics. ISBN 3-211-83392-7.