To appear in the Proceedings of the 1997 Symposium on Interactive 3D Graphics

Motion Editing with Spacetime Constraints

Michael Gleicher *

Apple Research Laboratories

Abstract

In this paper, we present a method for editing a pre-existing motion such that it meets new needs yet preserves as much of the original quality as possible. Our approach enables the user to interactively position characters using direct manipulation. A spacetime constraints solver finds these positions while considering the entire motion. This paper discusses the three central challenges of creating such an approach: defining a constraint formulation that is rich enough to be effective, yet simple enough to afford fast solution; providing a solver that is fast enough to solve the constraint problems at interactive rates; and creating an interface that allows users to specify and visualize changes to entire motions. We present examples with a prototype system that permits interactive motion editing for articulated 3D characters on personal computers.

CR Categories and Subject Descriptors: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Animation; I.3.6 [Computer Graphics]: Methodology and Techniques - Interaction Techniques; G.1.6 [Numerical Analysis]: Optimization.

Additional Keywords: Spacetime Constraints, Motion Displacement Mapping.

1 Introduction

Advances in techniques such as motion capture, keyframe editing, and physical simulation make it possible to create high quality motion for animation. Editing these motions is often useful, whether to make some unforeseen adjustment or to reuse the motion for an entirely new purpose. Unfortunately, adjusting a good motion can lose something – the motion may no longer be physically correct, or may lose some nuance given by the original animator. We therefore must look at motion editing as a creative process where decisions are made as to how best to keep the original quality of the motion while meeting new needs.

Traditional motion editing tools can be inconvenient for making alterations to existing motion. Current systems provide a range of techniques for editing poses at given times and use keyframe interpolation to reduce the tedium of specifying every frame. While recent methods, especially motion-displacement mapping[2][21], enable large scale changes while preserving fine detail, they offer little assistance at maintaining specific aspects of the original motion, except to tediously check each frame. The user gets no explicit control over non-key frames.

In contrast, spacetime constraint methods[20] consider the entire motion simultaneously. These methods enable the user to specify constraints over the whole motion and use a solver to compute the "best" motion that meets these requirements. Since a single large mathematical problem must be solved to determine a motion, spacetime constraint solutions have been computed as batch processes. While the methods have been successful for synthesis of physically realistic motions, to date, they have offered little help for interactive editing.

In this paper, we present a new method for motion editing. Like more traditional keyframe and inverse kinematics methods, the user makes adjustments to an animated character with direct manipulation, for example pulling on a character's hand to reposition it. But, to achieve these new positions, the animation system makes adjustments that attempt to preserve the original motion. Like other spacetime constraint methods, our system considers the entire motion in making changes. Unlike the previous spacetime systems, we solve the numerical constraint problems fast enough to provide direct manipulation dragging. To achieve this new style of motion editing, we must tackle three sets of issues: first, we need a constraint formulation that is rich enough to be effective, yet simple enough to permit rapid solution; second, we need fast methods for solving these constraint problems; third, we need new interaction techniques for specifying and visualizing changes to an entire motion. This paper addresses these three topics.

We emphasize that we do not have new solution techniques that allow us to solve the same spacetime problems orders of magnitude faster than previous systems. Instead, we rely on simplifications and approximations that would be unacceptable for the tasks of synthesizing physically-correct motions. Because our goal is interactive editing, we can make many tradeoffs to achieve performance. The quality of the motion can come from the talent of the animator, not just the solution process. While our methods do not afford automatic synthesis of novel motions, they do enable interesting interactions for editing existing motion.

2 A Model for Motion Editing

Our task is to take a motion that is basically good but is in need of adjustments, i.e. our initial motion has the basic form that we want. Consider one adjustment: at a particular instant of the animation, we have a specific, unmet desire. To use a simple example, consider Figure 1. We have a good jumping motion, but we would like to have our character jump a little higher. We would like to add the constraint that the character's hand is at a particular, higher location at frame 20 (which is the time of the apex of the jump). We expect to keep a similar jump – raising the hand will not cause a dramatically different motion like obtaining a ladder and climbing it.

The traditional method that is most applicable to this task is inverse kinematics. Traditional inverse kinematics (IK) enable the user to edit motion by dragging an end-effector of a character. The IK solver computes the best pose of the character that achieves this new positioning. Good IK methods can prevent the violation of other constraints, such as keeping joint angles from going past their

*Apple Computer, 1 Infinite Loop M/S 301-3J, Cupertino, CA 95014.
gleicher@apple.com
http://www.research.apple.com/People/gleicher

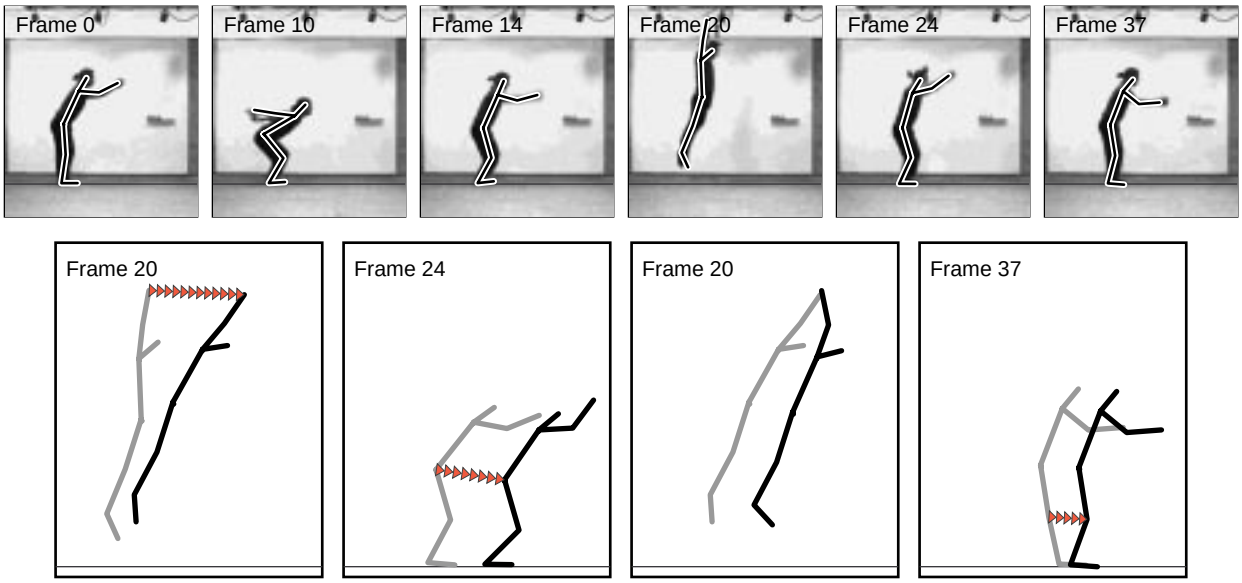

Figure 1: Editing a rotoscoped jumping motion. Frames from the initial motion are shown in the top row. The lower row shows the three editing drag operations, first pulling the hand at frame 20, pulling the pelvis at frame 24 (frame 20 is shown after this edit, notice that the hand constraint is maintained), and pulling the knee at frame 37. The grey figure shows the position before each dragging operation.

legal ranges.

A traditional IK solver considers only one pose in its computation – for editing, it controls only the current frame. If the frame is a key frame of an interpolated motion, other neighboring frames will be altered, although the number of frames and the way that they are affected will be determined by the key spacing, an artifact of how the original motion was created. Motion displacement techniques improve this by allowing the key spacing for editing to be set independently of the original motion. However, they provide no control over what happens between the key frames.

In contrast, our extended solver computes the best motion that achieves the specified positioning, considering more than just the current frame. This, of course, leaves the open question of how do we define what the "best" motion is. There is an obvious candidate for editing a good motion: the best motion is the one that best preserves the important qualities of the original motion. This is still open to a realm of interpretation as we cannot know what about the original motion the user wants to keep. We address this with a threefold strategy that we will detail in Section 4:

1. We augment the motion and the character to include constraints that must be maintained to keep the integrity of the motion.

2. We minimize some metric comparing the original and resulting motion over all the possible solutions that satisfy the constraints.

3. We diminish the importance of getting the "right" answer by providing interactive performance. The user can see undesired solutions and add additional constraints to achieve desired results.

For each dragging operation, there may be many constraints that are maintained on the motion, but there will be only a small number (often, but not always, one) of constraints that drive the changes to the motion. This constraint may be created specifically for a dragging operation, for example, when we grab the character's hand to adjust the jump, a constraint on the hand position is created if one does not already exist. As the mouse is moved, the constraint is adjusted, and the motion is reconfigured, hopefully rapidly enough to give the illusion of continuous motion for direct manipulation feel. Typically, we leave the constraint in place after the dragging operation is completed to make the alteration persistent. The model of one constraint continuously changing while others are maintained has implications of this on usability and performance [6].

When making an edit, the user can also specify a time range for the change. The system tries to create a change that is somewhat smooth at this scale, keeping higher frequency details. This parallels Cohen's concept of "windows" [3] for focusing the numerical solution and preventing unwanted lookahead.

3 Extensions to Previous Approaches

Our work combines two powerful techniques in computer animation, motion displacement maps and spacetime constraints. Spacetime constraints approach was introduced by [20]. [3] extended this with a more complete system and the concept of windows that focus the solution process. Subsequent work has addressed solver performance [13] and provided alternative formulations [11] [15] [19]. Recently, [12] applied the approach to the problem of generating transitions between motion segments, and [8] showed how the methods can be used for repurposing motions.

Previous spacetime work has not been applicable for interactive editing, due to a number of factors, including the computational expense of solving, difficulty in controlling the computations, and the failure to identify objectives other than energy consumption. To achieve interactive editing, we use a different formulation that places less emphasis on the minimization of the objective function and instead relies on the initial motion, the motion representation, and kinematic constraints to create the new motion. This simplification does sacrifice spacetime's ability to generate physically correct motions from sparse user input, but it affords fast approximate solvers, and makes the user interface easier: rather than having to define objective functions that mathematically characterize motion qualities, we instead provide initial motions and kinematic constraints.

Motion displacement maps (also known as motion warps) were introduced by [2] and [21]. The methods, which we will review in Section 4.1, allow motions to be edited independently of the scale of the detail in the original motion by adding a new curve with properties that make it convenient for editing. By using an interpolating curve for the displacement map, the method can be a simple extension to existing keyframe animation systems. Our approach gives up this simplicity, but provides additional user control by allowing simultaneous constraints at non-key frames. Constraint-based control of motion displacement maps was introduced in [8]. This work focused on providing detailed descriptions of motion changes using many constraints across the entire motion which made interactive performance infeasible.

4 Constraints, Parameters, Objectives

Like all spacetime approaches, we have turned our problem into a numerical constrained optimization. Subject to a set of constraints, including both those that maintain the existing motion and those that specify desired changes, minimize some objective function. In mathematical terms, we write this problem as

$$\text{minimize}\, g(\mathbf{x}) \text{ subject to } \mathbf{f}(\mathbf{x}) = \mathbf{c}. \tag{1}$$

Where $\mathbf{x}$ is a vector that represents the parameters of the motion, g is the objective (a scalar function of $\mathbf{x}$), and $\mathbf{f}$ is a vector function of the constraints. For each cycle of a dragging operation, a few elements of the vector $\mathbf{c}$ will be updated, and the equations re-solved. We now explore the three pieces of the optimization problem, keeping in mind that we need to choose them in a way that will lead to Equation 1 being solvable in a timely fashion.

4.1 Representation

The mathematical formulation of the animation problem encodes the motion into a vector of parameters. We assume that the animated figure's configuration can be represented by some vector of parameters, for example, an articulated figure by the position of its root and the joint angles. An animated motion $\mathbf{m}(t, \mathbf{x})$ is then a function that provides this vector given a time t, based on a vector of parameters that define the motion. For example, if the motion was created by a set of key values that are interpolated, $\mathbf{x}$ would be the concatenation of all of the keys. Typically we include only parameters that are to be adjusted in $\mathbf{x}$.

Typically, for keyframe animation systems the motion function is some interpolation of the keys since this allows the keys to be manipulated individually. Spacetime approaches are more flexible since the algorithms need not look at individual keys. Previous spacetime systems have chosen representations that are more well behaved mathematically, such as B-Splines and Wavelet B-Splines [13].

For motion editing, we are given an initial motion $\mathbf{m_0}(t, \mathbf{x_0})$ to adjust. Presumably, this representation was chosen to be convenient for the creation of the motion, and may not be adequate for the task of editing. Motion displacement techniques offer an alternative. Rather than use these parameters, we can treat the initial motion as a constant and add in a new curve with convenient parameters, that is, we let our edited motion be

$$\mathbf{m}(t, \mathbf{x}) = \mathbf{m_0}(t) + \mathbf{d}(t, \mathbf{x}), \tag{2}$$

where $\mathbf{d}$ is a new motion curve. This provides the freedom to choose a representation that is convenient for editing. For example, a straightforward implementation could use an interpolating curve with its keys placed at the times that the user specifies new poses.

We use the motion displacement approach for our spacetime computations. This allows us to choose our parameterization based on our computational needs, as well as the scope of the changes we would like to make. Our system allows a variety of displacement curve types, such as linear and cubic interpolation, cubic B-Splines, and wavelet B-Splines. In previous work [8], we experimented with using contrived displacement functions as a mechanism for controlling the changes. In an interactive setting, we prefer (for both numerics and user interface) to use standard curves, typically endpoint interpolating cubic B-Splines.

The key spacing has an important impact on the results of the motion edits. If the keys are spaced far apart, the displacement curve will be smoother and will add only low frequency (typically subtler) changes to the motion. Having distant keys also means having fewer keys, which can reduce the time for solving the optimization problems. On the other hand, with distant keys, it may be impossible to meet all of the constraints. In such cases, we aim for a least-squares norm solution, solving the constraints as best as possible, distributing the error amongst them. In cases where more accuracy is needed, we can run a cleanup process later where additional degrees of freedom can be added and resolved. There is no guarantee that the coarser solution is a correct approximation to the finer one.

We generally use two strategies for picking the key spacing and positioning for our displacement curves. The simplest is to evenly space a number of keys along the duration of the motion. Alternatively, when making an edit, the user can specify a time range for the change. In response to the user's range specification, a new motion displacement curve is created with fixed zero interpolating control points at its ends, and a small number of control points evenly spaced along its duration. It is these latter points that the solver configures. The use of a motion displacement map causes the alteration to be done at the user specified scale, while preserving finer details in the motion. However, since the solver can configure all the controls along the displacement, constraints can be placed at any time, freeing the user to drag at times other than key times, and place multiple constraints.

4.2 Constraints

Constraints in our approach serve two purposes: they encode specific aspects of the motion that should be maintained during subsequent edits and they serve as handles to drive changes to the motion. There is little distinction in our system, in fact, constraints are often moved between the categories.

Most constraints that we consider in our approach are kinematic, that is that they place a restriction on the configuration of the character at a given instant. These constraints have the form

$$f(\mathbf{m}(t_c, \mathbf{x})) \,\square\, c$$

where $\square$ is one of $\leq$, $\geq$, or $=$, t_c is the time at which the constraint exists, c is some scalar, and f is the "constraint function." Typically, a conceptual constraint consists of multiple scalar constraints, for example a point position uses one per axis. For notational convenience, we group all constraint functions into a single vector function.

We do not explicitly provide for dynamic constraints, e.g. constraints on accelerations or velocities. We do permit constraints that relate multiple instants, such as requiring that a point is in the same place as it is in another frame. This provides an approximation to dynamic constraints.

Often, we prefer to place constraints over a range of times, rather than at a specific instant. We call such constraints *variational* constraints because ideally they describe a relationship over a continuous interval. In practice, we implement variational constraints by sampling: placing individual kinematic constraints at specific instances within the time interval. The most obvious sampling is to place a constraint on each frame.

There are many constraints that can be useful in motion editing. Some that we have found useful in our prototype system include specifying: the position of a point at a specific time; the position of a point over a range of times, for example a hand-hold or footplant; a path that a point must follow; a region a point must stay inside or outside of; limits on the range of joint angles; and fixed distances between points. By point we include any position along the kinematic chain of a character.

The most obvious constraint for interactive dragging is one that positions a point in a specific frame. This position can be tied to the location of the input device during dragging. Other constraints also serve as interesting interactive handles; for example, footplant and handhold locations or obstacles to avoid, may also be dragged.

Obtaining the large numbers of constraints that seem to be required to produce high-quality adaptations is, in practice, far less work for the user than it might seem. Constraints tend to fall into three general categories: constraints on the character, such as joint angles; constraints that provide information about the initial motion, such as footplants and handholds; and constraints used to make adjustments. Given the amount of work usually required to obtain a model for a character, the extra effort is minor. Similarly, augmenting the motion takes an insignificant amount of time compared to obtaining the motion, especially since many of these constraints can be obtained semi-automatically (such as detecting footplants by examining foot heights).

4.3 Objectives

Usually, there will a space of possible solutions that meet the constraints, the solver chooses the solution that minimizes an objective function. Even in cases where there is no solution to the constraints, there is typically a space of equally close solutions. To select among the solutions, we choose one that minimizes an objective function. Previous spacetime approaches have chosen to minimize the energy consumption of the character. For our approach, we choose to minimize the difference between the motion before and after the edit. This still provides a range of choices, for example, we might choose to match the positions of the end-effectors, their velocities, their accelerations, or the joint angles. Our initial experiments suggest that there is no right answer – it is easy to construct an example where one metric is either clearly right or clearly wrong.

Rather than seek the perfect objective, or devise a good way of presenting the choices to the user, we have chosen an alternative approach: not worry about it too much. Instead of using the objective function for control, we instead use constraints. We pick a simple objective function that is selected to make solution of the optimization problem as fast as possible. The interactive performance that this allows enables the user to make adjustments by adding more constraints, guiding the system closer and closer to a satisfactory answer.

A simple objective function is suggested from the form of Equation 2. The difference between $\mathbf{m}(t)$ and $\mathbf{m_0}(t)$ is the displacement curve $\mathbf{d}(t)$. Minimizing the displacement curve would require minimizing an integral over the duration. Because of the types of curves we employ, we can approximate this by minimizing the magnitudes of the controls.

Simply minimizing the magnitude of the parameter vector does have some drawbacks. One particular problem is that different parameters often have vastly different effects, for example if one is measured in millimeters and another in miles, if one is an angle that affects the pinky and another is the orientation of the entire human figure, or if a key is close to other keys verses one that affects a very large number of frames. To combat these problems we use a weighted sum-of-squares of the parameters, where the weights are chosen to unify the parameter sensitivities as discussed in [7]. The sensitivities are computed by summing the amount each point on the character ($j \in pts$) in each frame ($k \in t$) is moved by a change in a variable i,

$$\mathbf{s}_i = \sum_{k \in t} \sum_{j \in pt} \frac{\partial \mathbf{p}_j(\mathbf{m}(k,\mathbf{x}))}{\partial \mathbf{x_i}} \cdot \frac{\partial \mathbf{p}_j(\mathbf{m}(k,\mathbf{x}))}{\partial \mathbf{x_i}}. \qquad (3)$$

For best interactive performance, we compute the weightings once for each dragging operation, and sample the sums of Equation 3 sparsely. The objective function is then

$$g(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{S}\mathbf{x}, \qquad (4)$$

where $\mathbf{S}$ is a diagonal matrix with the sensitivities as its elements.

Our objective can be seen as an approximation to the function that minimizes the amount the points on the character are displaced over the course of the motion. Equation 4 is a quadratic approximation to this function, the matrix $\mathbf{S}$ is a diagonal approximation to the Hessian. We do not argue for the correctness of this objective, nor the validity of the approximation, but rather, admit that these were chosen because they will lead to systems of equations that can be solved efficiently in Section 5.2.

5 Solving

We have posed the motion editing problem as a numerical constrained optimization, in the form of Equation 1. This is a standard, well-studied problem, albeit a difficult one. A good introduction to solution methods is [17], and a more thorough one can be found in [4] or [5]. Our usage is non-standard because of our performance demands: to provide for interactive dragging, we must solve the constrained optimization problems fast enough to provide the illusion of continuous motion. The nature of our problem makes this all the more challenging: we have nonlinear equations with potentially large numbers of constraints and variables since we create a single problem for an entire motion. We provide an overview of our solver here primarily to show how standard methods can be adapted to our unique needs.

The performance of general purpose, nonlinear solving algorithms is difficult to characterize. Available algorithms are iterative, taking a series of steps that refine an estimate of the solution and are extremely sensitive to the problem, slightly different equations or starting conditions can cause different solution times. Performance considers two things: the overall solution time on real examples, which is important since it indicates what can be achieved in practice; and the cost per iteration, which typically does have fixed computational costs and because we have the opportunity to display the intermediate results between steps if the solver does not run to conclusion.

The most common general class of nonlinear solving algorithms by building approximations of the nonlinear problem. At each iteration, an approximation with a form that has a known solution method is created. The algorithms we use belong to a class of algorithms known as *sequential quadratic programming* (SQP) because they use are quadratic programs as the approximations. Quadratic programs have linear constraints and quadratic objective functions.

For our discussion, we consider only equality constraints because they are simpler. In practice, we implement inequality constraints using *active set* techniques [4] that operate by switching equality constraints on and off. Our solver employs a simple, approximate active set method presented by [7].

An iteration of an SQP solver begins with the current estimate for the solution $\mathbf{x_i}$ and compute an updated solution $\mathbf{x_{i+1}}$ by:

1. building a quadratic program using Taylor expansion of the non-linear functions. The constraint function is approximated

by

$$\mathbf{f}(\mathbf{x_i} + \mathbf{\Delta}) \approx \mathbf{f}(\mathbf{x_i}) + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\mathbf{\Delta},$$

where $\partial \mathbf{f}/\partial \mathbf{x}\mathbf{\Delta}$ is the Jacobian of the constraint function at the value $\mathbf{x_i}$ which we denote by $\mathbf{J}$. The constraint equation is therefore

$$\mathbf{J}\mathbf{\Delta} = \mathbf{c} - \mathbf{f}(\mathbf{x_i}) = \mathbf{c_i}, \tag{5}$$

where we denote the residual as $\mathbf{c_i}$.

A second order Taylor expansion to provides a quadratic version of the objective function,

$$g(\mathbf{x_i} + \mathbf{\Delta}) = g(\mathbf{x_i}) + \frac{\partial g}{\partial \mathbf{x}}\mathbf{\Delta} + \frac{1}{2}\mathbf{\Delta}^\mathbf{T}\frac{\partial^2 g}{\partial \mathbf{x}^2}\mathbf{\Delta}.$$

We denote the gradient of g as $\mathbf{g}$ and the Hessian matrix as $\mathbf{G}$. Note that when g is a quadratic function $g(\mathbf{x}) = 1/2\mathbf{x}^\mathbf{T}\mathbf{G}\mathbf{x} + \mathbf{b}\mathbf{x}$,, the Hessian is $\mathbf{G}$, but the gradients is not $\mathbf{b}$, but rather $\mathbf{G}\mathbf{x} + \mathbf{b}$.

2. solve this quadratic program for $\mathbf{\Delta}$, the step direction vector. If the original constrained optimization problem is a quadratic program (or if this approximation is very good), then $\mathbf{x_i} + \mathbf{\Delta}$ would be the answer.

3. performing a line search to find how far to move in the step direction. This computes a value κ for which $\mathbf{x_i} + \kappa\mathbf{\Delta}$ provides the best answer to the problem. This search minimizes a merit function that accounts for both the constraints and the objective.

This 3 step process is repeated until the algorithm decides to terminate either because the solution $\mathbf{x_i}$ is sufficiently good or because no progress is being made. The stopping tolerances can be set based on the problem's precision requirements. The following sections describe how we have implemented each of the 3 tasks of the step process in our system.

Our system solves the non-linear optimization problems between each refresh of the screen. This is in contrast to previous differential approaches [7] that emphasize constant frame rate, refreshing the screen after each update of the configuration.

5.1 Computing Functions and Derivatives

Solving the optimization problem requires evaluating the objective and constraint functions and their derivatives. These are large and complicated: they involve the blending of keys, the kinematics of the character, and the relation applied to these points. Viewed as a monolithic whole, these functions are daunting. However, viewed as smaller pieces composed together, the task is much more manageable. The derivatives of these composed functions can be built by the chain rule. Rather than symbolically performing the process, the elements are computed and multiplied together numerically, a process called *automatic differentiation* [10]. For example, a constraint of the position of the hand of an articulated figure at time t might have the form

$$\mathbf{f}(\mathbf{x}) = \mathbf{p_h}(\mathbf{m}(t, \mathbf{x}))$$

where $\mathbf{p_h}$ is the kinematic function that computes the position of the hand given the figure's parameters, and $\mathbf{m}$ is the function that computes the parameters at a given time from the key configuration. The chain rule permits us to compute

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \frac{\partial \mathbf{p_h}}{\partial \mathbf{m}}\frac{\partial \mathbf{m}}{\partial \mathbf{x}}.$$

Each of the two matrices can be computed independently and multiplied together. We use a general purpose implementation of automatic differentiation [7] [9] designed specifically for the demands of interactive systems, allowing functions to be defined dynamically and evaluated efficiently.

To compute the Jacobian of the kinematic function $\partial \mathbf{p_h}/\partial \mathbf{m}$, we also use a mixed symbolic-numeric approach. The kinematics is defined by a chain of matrix multiplications. We compute the Jacobian by recursively applying the product rule for differentiation, symbolically computing the derivatives of each transformation but combining them numerically. Special case techniques exist for performing these evaluations for rigid kinematic chains (for example one is described in [12]), but we have not yet explored such techniques.

The form of the objective function (Equation 3) does not require us to evaluate the Hessians of complicated functions. Instead, our objective Hessian is built by computing the gradient of many functions, performing dot products on them, and summing them together.

Sparsity is an important consideration in implementing our approach. Avoiding excess storage by using representations that exploit the large numbers of zeros in the gradients, Jacobians and Hessians is vital for performance, in evaluation and especially in linear system solving in the next section.

5.2 Solving the Quadratic Program

At each step of the SQP, we must solve an optimization problem that has a quadratic optimization objective and linear constraints,

$$\begin{aligned} \text{minimize } g(\mathbf{\Delta}) &= \frac{1}{2}\mathbf{\Delta}^\mathbf{T}\mathbf{G}\mathbf{\Delta} + \mathbf{g}\mathbf{\Delta} \\ \text{subject to } \mathbf{f}(\mathbf{\Delta}) &= \mathbf{J}\mathbf{\Delta} = \mathbf{c_i} \end{aligned}$$

There are many methods available to solve these problems (see [4] or [5]). Most methods involve posing the problem as a linear system. The method that we use exploits the fact that the Hessian matrix is easy to invert. The method is detailed in [7]. Briefly, the extrema of a function is the point where the gradient vanishes. However, this point may not be feasible given the constraints. The constrained extrema, therefore, is a point where the gradient points in a direction that is prohibited by the constraints. The gradient of g must therefore be a linear combination of the constraints

$$\mathbf{G}\mathbf{\Delta} + \mathbf{g} = \mathbf{J}^\mathbf{T}\boldsymbol{\lambda}, \tag{6}$$

where $\boldsymbol{\lambda}$ is a vector called the *Lagrange Multipliers* that denote the linear combination. Solving this equation for $\mathbf{\Delta}$ gives

$$\mathbf{\Delta} = \mathbf{G}^{-1}\mathbf{J}^\mathbf{T}\boldsymbol{\lambda} - \mathbf{G}^{-1}\mathbf{g} \tag{7}$$

which we plug into Equation 5 to get

$$\mathbf{J}\mathbf{G}^{-1}\mathbf{J}^\mathbf{T}\boldsymbol{\lambda} = \mathbf{c_i} + \mathbf{J}\mathbf{G}^{-1}\mathbf{g}, \tag{8}$$

a linear system that we solve for $\boldsymbol{\lambda}$, since $\mathbf{c_i}$, $\mathbf{g}$, $\mathbf{G}$, and $\mathbf{J}$ are known. This is then substituted back into Equation 7 to compute $\mathbf{\Delta}$.

If the constraints are over-determined, meaning that there is no exact solution so that we are looking for one with least-squared error, the multipliers are underdetermined. To handle this case, we use a method known as damping or multiplier penalties [7] [14]. While this method has some numerical disadvantages, it has the advantage that it merely replaces Equation 8 with

$$(\mathbf{J}\mathbf{G}^{-1}\mathbf{J}^\mathbf{T} + \epsilon\mathbf{I})\boldsymbol{\lambda} = \mathbf{c_i} + \mathbf{G}^{-1}\mathbf{g}, \tag{9}$$

where ϵ is a small constant, and $\mathbf{I}$ is the identity matrix. We prefer this approach because it allows us to use any method we like to

solve the positive-definite-symmetric linear system. We have chosen a conjugate-gradient method [1]. This iterative algorithm allows us to trade accuracy for performance and exploits the sparsity in the matrix.

5.3 Line Search

If our optimization problem were a quadratic program, computing $\mathbf{\Delta}$ would provide the answer. For the non-linear problem, $\mathbf{\Delta}$ is only a suggestion as to the best direction to search from the current estimate. Because of the expense of computing the direction, it is usually worthwhile to maximize its utility, by searching along the direction. Because we are attempting to minimize the objective and satisfy the constraints simultaneously, we select the step length based on both of these. We define a merit function

$$m(\mathbf{x}) = \alpha_1 g(\mathbf{x}) + \alpha_2 \mathbf{f}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x}) \tag{10}$$

that evaluates the quality of a solution. We find a value of the step length κ that for which the new value $\mathbf{x_i} + \kappa\mathbf{\Delta}$ minimizes the merit function. The line search process tries various values of κ, evaluating the merit function for each. We use a variant of Brent's method [17] to perform the line search.

There is a tradeoff between spending effort on computing step directions and on making evaluations in the line search. For problems such as ours, some line searching is useful because evaluations are much less costly than computing new step directions. Another tradeoff in the line search process is to prefer solving the constraints (reducing the magnitude of the residual) or minimizing the objective function. This tradeoff is controlled by choosing the values of the constants α_1 and α_2 in Equation 10. We typically emphasize the constraints by choosing a very small value for α_1 because their error is much more noticeable than non-minimal objective values.

5.4 Performance

In order to provide interactive performance, we must be able to solve the constrained optimization quickly. We must perform the solve/redraw cycle many times per second, each solution cannot take very long. Our entire approach is designed around these performance needs. Some of these considerations:

- the motion displacement maps enable the use of sparse keys, which lead to fewer variables to solve for;
- the user defined time range limits the scope of the computations;
- the maintain/drag model leads to good starting points for the numerical solutions;
- the inexact nature of typical animation tasks means that imprecise answers are acceptable. This allows the use of large tolerances for stopping criteria and coarse sampling of the variational constraints. We use iterative algorithms so that we can trade accuracy for performance;
- the simple objective function has a Hessian that is trivial to evaluate and invert. While computing the sensitivities at the beginning of each dragging operation can be expensive, it's impact on the condition of the linear system and quality of the results are worthwhile, and it can be sampled sparsely to cut computation cost.

The utility of good starting points creates a bit of a performance paradox. If solutions are fast, the distance that the user can move the mouse is smaller, so the required adjustments are smaller, so the solution times are shorter. As solutions take longer, the user has more time to move the mouse, creating larger adjustments, which in turn require more time for solution.

During dragging, we have the greatest performance demands, so we cut corners, using higher tolerances on the iterative solvers and sparse sampling of the variational constraints. At the end of the dragging operation, we perform a cleanup solution at the standard settings. While this sometimes leads to an undesirable pause and jump, the differences between the fast and normal solutions are typically small enough that it is not a problem.

6 Interface

Our spacetime constraints approach to motion editing has a number of challenges in the design of the interface for motion editing. The user must be able to specify and adjust constraints over the entire motion and see the results. Like most animation systems, our prototype allows direct manipulation in multiple 2D and 3D views.

While our prototype system does not have a commercial quality user interface, and has not yet been tested on animators, we have tried to restrict ourselves to an interface design that could conceivably be used by someone other than a programmer. In particular, our interface never presents equations to the user – all constraints can be specified by direct manipulation (or through scripting), and constraint are displayed graphically.

6.1 Visualization

Providing feedback for motion editing is a tough task before we add the additional complexities of constraints – there is a lot of data that has meaningful temporal information. In our spacetime constraints editing approach, the visualization problem becomes even more challenging. The constraints are yet another bit of information that must be conveyed. Because a duration of the motion may be affected by a dragging operation, it is even more important that the user can see the changes over this entire duration as editing occurs. Our system provides a number of simple tactics that the user can employ:

- the use of multiple windows, for example having a second window showing the animation cycling while motion is edited;
- drawing multiple frames simultaneously to create a strobe effect, often using simplified drawing or transparency to reduce clutter;
- drawing streamers that trace the paths of points.

Representation of constraints is a challenging problem even in a static scene. We prefer visual representations that display constraint symbols in the locations that they act, as opposed to representations that are separate. Designing a visual representation for constraints requires trading off visibility of the constraints and their function against clutter and visibility of the objects. When constraints from multiple frames are shown simultaneously, the clutter problem becomes even worse, especially since different frames often have similar constraints. Presently, we use extremely simple representations for constraints and rely on selective display to manage interframe clutter problems. Figure 2 and the color plate shows an example that uses many of the visualization methods together.

7 Examples

In this section we provide some representative examples that we have used with our system. All run in real time using our prototype software on a Power Macintosh 8500/180 with a 180MHZ

Figure	d.o.f.s	keys	frames	consts total	consts drag	Time (msec.) eval	jac	solve	step	SQP	draw	Rate (fps)
1 2D Jump	9	7*	37	1025	514	2	2	7	19	57	23	10
3 2D Walk	14	12	81	2195	1125	4	6	34	115	360	63	2.5
2D Walk (range)	14	5*	81 (45)	2195	671	2	3	18	50	142	61	5
5 2D Slide	14	6*	55	1546	799	3	4	27	69	65	70	5
2 3D Walk	57	10*	88	2850	1440	7	17	12	139	284	58	3
6 3D Walk	35	10*	88	2811	1422	7	11	4	94	182	59	4
3D Slide	37	5	54	4176	2123	6	12	10	73	127	56	5
7 3D Thief	49	10	100	5921	1587	9	38	24	221	261	94	3

Table 1: Examples discussed in this paper. *Range* examples limit to a smaller number of frames. *D.o.f.s* are the number of degrees of freedom used for the character in the example. *Keys* are the number of B-Spline control points used in the displacement curve, an asterisk denotes that one of the end frames is frozen. *Frames* is the length of the motion. Both the *total* number of constraints and the reduced number of constraints used during dragging (*drag*) are given and include the constraint used for manipulation. Timings are averages over all cycles during a long dragging operation: *eval* is the time required to evaluate the constraint and objective functions, *jac* is the time taken to compute the Jacobian of the constraints and the gradient of the objective, *solve* is the time to solve the linear system, *step* is the amount of time for one step of the SQP solver, *SQP* is the time for solving the nonlinear programming problem, and *draw* is the time for the redraw required at each cycle. For the 2D slide example, the solve times are misleading because the mouse sometimes did not move enough to require the solver to take a step. The last column is the average frame rate during dragging.

Figure 2: (see color plate section for color version) Visualizing a spacetime constraint-based motion edit of a walking motion. Strobing (with color alternation and transparency to help contend with clutter) and streamers (the thin lines following the feet and hand) are used to convey the motion. The striped streamer shows the initial (pre-edit) motion. Dark symbols represent constraints.

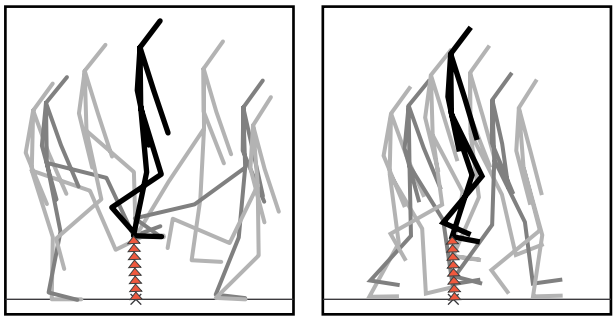

Figure 4: The example of Figure 3 is detailed. On the left, we see frames near the edited step. Notice that the footplant constraint is maintained as its position is moved. On the right, standard motion displacement mapping techniques are applied to adjust the edited frame. The method does permit the user to reposition the foot and provides a smooth transition, however, because it only considers constraints on one frame, the techniques achieve the edit by making the character float into the new place.

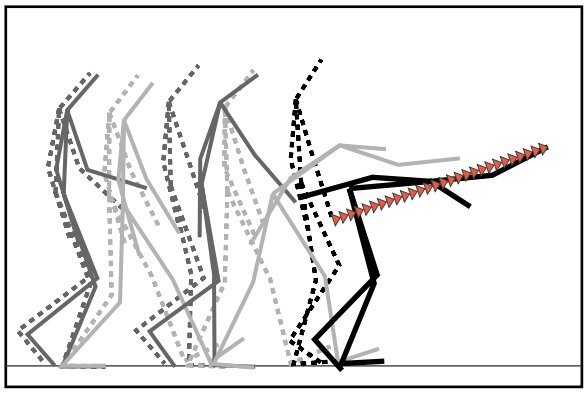

Figure 5: The first 55 frames of the walking motion of Figure 3 are edited to have the character reach for a spot in the last frame. In this example, the solver is permitted to adjust the footplant positions. The dotted figures represent the original motion, the arrows indicate the single constraint dragging operation. Notice that positioning the characters hands not only causes him to lean forward but to take larger steps as well.

PowerPC 604e microprocessor and a graphics accelerator for 3D rendering. The constraint counts and timings for all examples discussed in this section are presented in Table 1. Timings are averages over the many cycles during a long dragging operation. The time for many parts of the computation, especially the iterative portions, are highly variable and problem sensitive. We provide timings primarily to compare the costs of various parts of our algorithm.

The images in the figures are taken from our prototype system, but are not exactly what the user would see. The 2D images were written out as Postscript with arrows added in a drawing program. The 3D images are drawn with extra polygons to enhance their legibility, for example drawing tubes instead of lines.

7.1 2D Examples

Our first examples use a 2D human figure with 14 degrees of freedom (translation, orientation and 11 joint angles). The motions were rotoscoped by manually tracking points and using a version of our solver to compute an inverse kinematic solution that fills in missing data and maintains continuity.

The jumping motion in Figure 1 has 37 frames (recorded at 15fps), and has the stick figures right and left arms tied together (leading to a reduce number of degrees of freedom). Footplant constraints pin the foot position for the first 12 frames and force the foot to be on the ground and not skidding for frames 24 through 37. This latter "floating footplant" constraint does not specify the x location of the footplant, but does insure that it is constant over the 13 frames. Joint angles have upper and lower limits on each frame. This adds up to 1025 constraints including the 2 used to connect the hand to the mouse. The displacement curve is a cubic B-Spline with 7 control points, although the first control point is not permitted to change. For example 1, we alter the jump by grabbing the character's hand at the apex of the jump and pulling it to a desired position. The inexpensive solution methods provide an unrealistic solution that has the character snapping back to the original position. By adding additional constraints, a better motion can be achieved.

The walking example of Figures 3 and 4 have 81 frames, augmented with footplant constraints and joint limits. The footplant constraints have fixed positions. The example shows one dragging operation where the user drags the position of one of the footplants. To emphasize the difference between our approach and motion displacement maps, consider their use on this problem, shown in Figure 4. Both approaches allow the footplant to be dragged at the given frame, but only ours can relocate the entire footplant with one dragging operation.

In Figure 5, floating footplants are used. When the character's final destination is changed by dragging his hand in the final frame, the lengths of the steps are made longer.

7.2 3D Examples

Our 3D examples use motion capture data[1]. The model is taken directly from the the motion capture format [18] and uses ball joints throughout, leading to 57 degrees of freedom. This hierarchical model is rooted at the pelvis and uses Euler angles for each joint.

For some examples, we treat the elbows and knees as planar joints by preprocessing the motion to remove these degrees of freedom while maintaining end-effector positions. Some examples omit the feet and/or hands from the figure. These examples constrain the ankle position rather than the heel and toe and the wrist position rather than the fingertips. Although it detracts from the realism of the motion even before editing, we use this simplification because the reduced model is commonly used (for example by [12]), data for the end-effectors is sometimes incomplete, and because it speeds the constraint solution.

As in the 2D examples, the 3D examples use footplant constraints to prevent skidding. Again, specific footfall positions can be dragged, or floating footfalls adjust in response to other constraints. Figure 2 shows a walking motion. The footplant constraints are attached to the character's heels, and additional constraints are used to position the toe in a reasonable manner. It is these latter constraints that make this example difficult. When the feet are removed, as in Figure 6, solver performance improves. The 3D slide example, mentioned in Table 1 but not pictured, is analogous to the 2D example of Figure 5.

The thief of Figure 7 provides a more challenging example. The motion captured data has the thief approach, pick up, and run away with a treasure. We add constraints that ensure that the thief's hands are at the location of the treasure at the moment he picks it up, that the hands stay the correct distance apart while he carries the treasure, that his feet do not skid while he is on the ground, and his joints do not go beyond their limits. With these constraints, we are

[1]We thank Biovision Motion Capture Studios for providing us with this sample data.

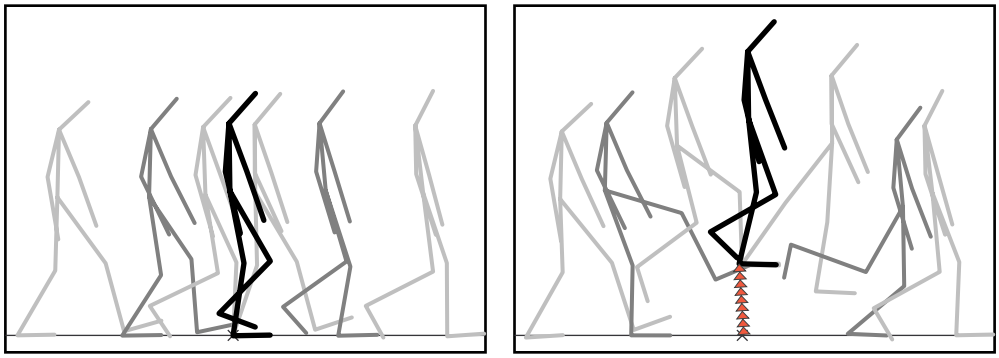

Figure 3: A walking motion is shown on the left by "strobing." The character is made to step over an obstacle by dragging the location of the footplant constraint.

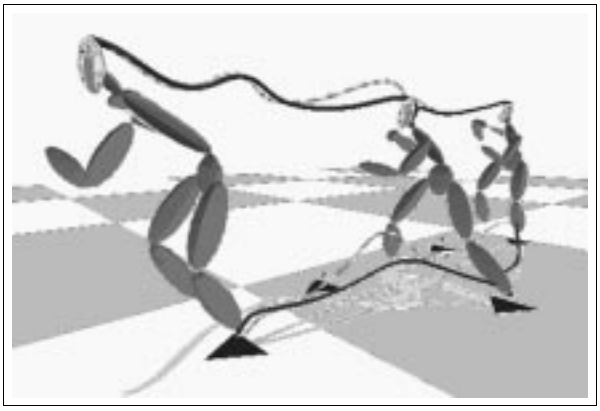

Figure 6: Adapting a walk without feet. The motion of Figure 2 is used with the character's feet removed. All constraints are applied to the ankles, which float above the floor. The footplant in the lower right of the image has been dragged. The dark traces show the motion of the head and foot, the striped traces show the original motion.

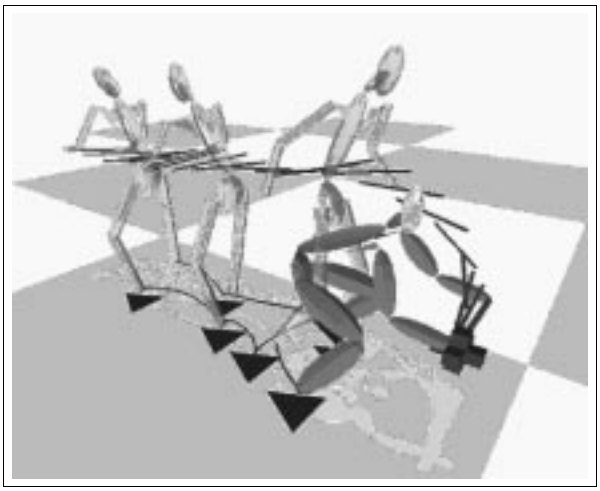

Figure 7: The thief bends down, picks up the treasure, and carries it away. Constraints ensure that the thief's feet do not slide, that the thief picks up the treasure at the correct spot, and that his hands stay the correct distance apart. These constraints are maintained if the treasure is moved.

able to reposition the treasure and have the motion adapted automatically. For the thief example, we use even coarser sampling of the variational constraints and larger tolerances to achieve interactive performance.

8 Discussion: Is this Spacetime?

At this point, it should be clear that our formulation is very different that previous spacetime approaches. In particular, we have given up the seemingly most prominent feature of previous spacetime work: physics and energy minimization. In our view, energy minimization is just one possible objective, in fact, one that is hard to justify for real characters. Ideally, we would like our objective functions to provide the animator with "director level controls" – specifying meaningful attributes such as "gracefully," "as if you were scared," or even "like Charlie Chaplin." While such objectives were anticipated by Witkin and Kass' original paper [20], these meaningful quantities seem hard to encode mathematically. In a sense, we provide the user with the ability to describe any of these objective functions by example. If the user desires a graceful walk, they can begin with a graceful walk.

A more serious omission is the lack of Newton's laws in our system. It is all too easy to make a character appear to fly through space, or make some other violation of physical law. At one level, these are simply constraints and could be added in. In fact, using finite different approximations for the time derivatives, adding constraints that $f = ma$, and for gravity is a simple addition. Part of our reluctance to include these constraints is computational: not only will they add to our already large numbers of constraints, but the physics constraints have structure that appears to make solution of the optimization problems much more difficult. Also, these constraints take control away from the animator – maybe the character is supposed to fly through space. In practice, we aim to find constraints that allow the animator to include elements of physicalness as desired, for example to keep the character from passing through the floor. In the future, we hope to find more ways to help prevent the animator from accidentally destroying the physicalness, or other desirable characteristics, of the motion.

There are many potential improvements to our implementation. Some obvious ones are the use of automatic adaptive subdivision to add a sufficient number of control points (as in [13]), the use of special purpose methods for evaluating the derivatives of kinematic chains more rapidly (like those employed by [12]), addition of constraints for balance (as in [16]) and better use of transparency and texture for constraint display. However, when considering richer optimization objectives and constraints, we must be careful not to violate the simplifications that make interactive performance possible on spacetime problems.

There is a tradeoff between having constraints and objectives that are better able to compute better edits and having a simple enough problem to solve at interactive rates. In this paper, we have chosen the latter approach. Our system allows editing motions such that they meet new constraints but maintain essential features of the original. While the solutions may not always be perfect, they are computed rapidly enough to permit the user to interactive guide the system to an acceptable solution by adding more constraints or adjusting existing ones. Our examples demonstrate that even on a personal computer we can achieve interactive rates with our current prototype.

Acknowledgments

I would like to thank Pete Litwinowicz for getting me interested in motion reuse, suffering with early versions of the software, rotoscoping the 2D motions, and jumping in front of the camera (for Figure 1). The people of BioVision Motion Capture Studios generously provided the sample motion and file format assistance. Pablo Fernicola of Apple's Quickdraw 3D team helped with my use of the graphics library. Sebastian Grassia at CMU offered helpful advice on numerics and articulated figures. Gavin Miller, Dulce Ponceleon, and Yalin Xiong offered advice on the writing.

References

[1] Richard Barrett, Michael Berry, Tony Chan, James Demmel, June Donato, Jack Dongarra, Victor Eikhout, Roldan Pozo, Charles Romine, and Henk van der Vorst. *Templates for the solution of linear systems: Building Blocks for Iterative Methods*. SIAM, 1994.

[2] Armin Bruderlin and Lance Williams. Motion signal processing. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 97–104, August 1995.

[3] Michael F. Cohen. Interactive spacetime control for animation. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 293–302, July 1992.

[4] Roger Fletcher. *Practical Methods of Optimization*. John Wiley and Sons, 1987.

[5] Phillip Gill, Walter Murray, and Margaret Wright. *Practical Optimization*. Academic Press, New York, NY, 1981.

[6] Michael Gleicher. Integrating constraints and direct manipulation. In David Zeltzer, editor, *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 25, pages 171–174, March 1992.

[7] Michael Gleicher. *A Differential Approach to Graphical Interaction*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1994.

[8] Michael Gleicher and Peter Litwinowicz. Constraint-based motion adaptation. Technical Report 153, Apple Computer, June 1996.

[9] Michael Gleicher and Andrew Witkin. Supporting numerical computations in interactive contexts. In Tom Calvert, editor, *Proceedings of Graphics Interface '93*, pages 138–145, May 1993.

[10] Andreas Griewank. On automatic differentiation. In M. Iri and K. Tanabe, editors, *Mathematical Programming: Recent Developments and Applications*, pages 83–108. Kluwer Academic, 1989.

[11] Radek Grzeszczuk and Demetri Terzopoulos. Automated learning of Muscle-Actuated locomotion through control abstraction. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 63–70, August 1995. held in Los Angeles, California, 06-11 August 1995.

[12] Brian Guenter, Charles F. Rose, Bobby Bodenheimer, and Michael F. Cohen. Efficient generation of motion transitions using spacetime constraints. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 147–154, August 1996.

[13] Zicheng Liu, Steven J. Gortler, and Michael F. Cohen. Hierarchical spacetime control. In Andrew Glassner, editor, *SIGGRAPH 94 Conference Proceedings*, Annual Conference Series, pages 35–42, July 1994.

[14] Yoshiko Nakamura. *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley, 1991.

[15] J. Thomas Ngo and Joe Marks. Spacetime constraints revisited. In James Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 343–350, August 1993.

[16] Cary Phillips and Norman Badler. Interactive behaviors for bipedal articulated figures. In Thomas W. Sederberg, editor, *Computer Graphics (Proceedings SIGGRAPH 91)*, volume 25, pages 359–362, July 1991.

[17] William Press, Brian Flannery, Saul Teukolsky, and William Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, England, 1986.

[18] BioVision Motion Capture Studios. BVH motion file format specification, 1996.

[19] Michiel van de Panne and Eugene Fiume. Sensor-actuator networks. In James Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 335–342, August 1993.

[20] Andrew Witkin and Michael Kass. Spacetime constraints. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 159–168, August 1988.

[21] Andrew Witkin and Zoran Popović. Motion warping. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 105–108, August 1995.