Constraint-based Motion Adaptation

Michael Gleicher [*] Peter Litwinowicz [†]
Apple Research Laboratories
Apple Computer, Inc.
Cupertino, CA 95014, USA

November 2, 1998

To appear in *The Journal of Visualization and Computer Animation*.
Edited draft from June, 1996
Edited draft from May, 1997
Edited draft from August 12, 1997
Edited draft from September 1, 1997
This version October 16, 1997, processed November 2, 1998

Abstract

Today's computer animators have access to many systems and techniques to author high quality motion. Unfortunately, available techniques typically produce a particular motion for a specific character. In this paper, we present a constraint-based approach to adapt previously created motions to new situations and characters. We combine constraint methods that compute changes to motion to meet specified needs with motion-signal processing methods that modify signals yet preserve desired properties of the original motion. The combination allows the adaptation of motions to meet new goals while retaining much of the motion's original quality.
Keywords: computer animation, motion adaptation, constraint solving, motion displacement and warping.

[*]current address: Vision Technology Center, Autodesk, Inc., 2465 Latham St. Suite 101, Mountain View, CA 94040, `gleichercs.cmu.edu`

[†]current address: 1059 Noe St., San Francisco, CA 94114 `litwinow@best.com`

1 Introduction

Creating high quality motion for animation is a tedious and difficult task. Even with advanced computer animation tools, creating motion that is purposeful, expressive and attractive requires considerable effort, typically requiring skilled animators or actors and engineers using motion capture equipment. The cost, difficulty, and talent required puts motion generation out of the reach of many potential users.

Despite its high cost, generated motion is not commonly reusable. More often than not, motion will only be valuable for a particular model and is almost certainly unusable for more than a particular scenario. For instance, the motion of a woman reaching for a doorknob will be precisely that — the motion will most probably be useless for having the character pick up something from the ground, or for a different character, or even for a different doorknob.

Computer animation research has evolved 4 general strategies to the problem of producing motion. The first is to improve the tools used for keyframing; for example, by adding inverse kinematics to control models. While such methods can make it much less tedious to produce motion, they are predominantly a tool for helping skilled animators produce single-use motions. Another strategy uses algorithmic or simulation methods to generate motions based on descriptions of goals. While such methods have the promise of generating motions for non-experts by allowing them to simply specify their needs, they are, at present, of limited use. A third approach tracks motion of real world actors or objects. This approach requires having real motion to capture and typically requires sophisticated sensors and processing.

A fourth, more recent, approach to the motion problem attempts to adapt existing motion generated by one of the other methods. Such adaptive methods could put animation capabilities in the hands of inexperienced animators, allowing the use of motion created by others in new scenarios and with other characters. It can also enable "on-the-fly" adaptation of motions, creating motion as needed in an interactive context.

One promising approach to motion adaptation, presented by Bruderlin and Williams[8], treats motions as signals and applies some traditional signal processing methods to adapt them, while preserving aspects of their character. A variant of one of their most interesting methods, motion-displacement mapping, was simultaneously introduced as "motion warping" by Witkin and Popovic [46]. Unfortunately, as initially presented the methods fall short of being able to realize our goals. In this paper, we present an approach to motion adaptation that extends the originally published methods to better address their shortcomings.

This paper presents constraint-based motion adaptation, an approach that uses numerical constraint solving techniques to alter motions so that they meet desired goals while preserving as much of the original motion as possible. We combine elements of two prior approaches: motion signal processing alters motion in ways that preserve desired qualities; and constraint-based direct manipulation (a generalization of inverse kinematics) provides a flexible mechanism for specifying goals. Together, these provide an approach whereby a user can create new motions that have the character of the original, but meet a set of new requirements.

The basic idea of our approach is to treat motion adaptation as a constrained optimization problem: what is the smallest change that can be made to a motion signal in order to meet a set of specified goals? Like the spacetime constraint methods of Witkin and Kass [45] and Cohen [9], we solve a constraint problem for the entire motion. In constrast, most other constraint methods that process each frame individually. The reward for solving these large numerical problems is a method that affords flexibility in the types of controls we can provide and the types of alterations performed on the motions. We can have the best of both worlds: easy to use inverse-kinematic controls with smoothness-preserving motion warps. For example, we can alter a character's walk by simply specifying new foot plant positions — the system keeps as much of the character of the original as possible and can insure that the feet never pass through the floor.

We begin this paper with a brief review of related work on motion for computer animation. Discussion of our methods follow, first by describing how some previous motion adaptation methods can be viewed under a common framework and controlled using constraint techniques. Extensions to these methods provide additional flexibility in the types of control. We discuss a variety of applicable constraints. Following a discussion of implementation details, we provide a number of examples created with our approach. We conclude with a discussion of these results and ideas for future exploration.

2 Previous Approaches

The simplest motion creation technique is the manual input of poses (keyframing). Input parameters such as positions, scale, and rotation angles are interpolated to generate poses between key poses.

Forcing the user to specify values for parameters is often inconvenient, especially for tasks like controlling the position of the hand of an articulated figure.

Inverse kinematics methods provide the user control over end-effectors (like the hand) by computing the configuration of the character required to achieve it. Robotics texts, such as Craig [10], or Nakamura [31] present methods for solving inverse kinematic problems, although they do not provide methods that address the concerns of computer animation. Methods that better address animation needs include those presented by Zhao and Badler [47], or surveyed by Welman [42]. Many commercially available animation systems, such as provided by Kinetix, Alias, and SoftImage, include inverse kinematics capabilities.

Inverse kinematics are a specific type of constraint-based method. Constraint-based methods use a solver to compute configurations that meet specified requirements, typically allowing many constraints to be satisfied simultaneously. Constraint solving has a long history in computer graphics, dating back to Sutherland's Sketchpad [39]. The utility of solving multiple constraints for positioning figures in animations was first shown by Badler et al [2]. The usefulness of a more general class of constraints was first examined by Witkin et al [44].

Physically-based methods attempt to create realistic motion by simulating the laws of physics. An early example was Reeves work on particle systems [36]. As the field has progressed, better and better simulation methods, such as those discussed by Baraff [3], permit more accurate modeling of more complicated effects, such as collision and friction. Physically-based approaches suffer from the drawback of having to always be "physical." They also suffer from being hard to control, which has lead many to study how to determine what forces and torques must be applied to achieve specified requirements. Methods for this include inverse dynamics, as presented by Armstrong and Green [1], Wilhelms [43], and Issacs and Cohen [24], and the dynamic constraints presented by Barzel and Barr [5] and Platt [34].

Almost all constraint-based approaches apply constraints to individual instants in time, either computing configurations that meet specified constraints (e.g. inverse kinematics) or required forces to apply at the current instant to meet constraints sometime in the future (inverse dynamics). Spacetime constraints are a very different variety that were first introduced by Witkin and Kass [45] and Brotman and Netravali [6]. By specifying constraints on the motion like "jump from here to there, clearing a hurdle in between" and "don't waste energy" (quotes taken from [45]), the method uses physical laws to produce the motion from first principles. To find the optimal motions, constraints over the entire motion must be considered simultaneously. Placement of a constraint at the end of a motion can affect the behavior of a character at the beginning.

While Spacetime Constraint methods have produced some exciting results, they have been limited to the creation of simple motions for simple characters. One limitation is the size and complexity of the numerical calculations required to solve the optimal control problems (although Liu et al [29] shows methods to improve the tractability). A potentially more pressing issue is that an optimally efficient, physically-correct motion is not always most desirable for animation. Other desires have been difficult to describe within the framework. As we will discuss in Section 3.6, our work has much similarity to spacetime constraints, yet avoids these restrictions.

An approach related to spacetime constraints attempts to design controllers for models, rather than their motions. Although some authors, particularly Ngo and Marks [32], also used the term "spacetime constraints" for the controller based approach, we prefer to reserve the term for methods like those presented by Witkin and Kass [45] that use constrained optimization to compute motions. Ngo and Marks [32] and Van Panne and Fiume [40] present methods that compute controllers for different characters, but do not necessarily have these characters produce controlled motions. Grzeszczuk and Terzopoulos [20] use methods to design controllers for more complex creatures that are more capable of meeting desired goals. Sims [38] extends spacetime to design the creature as well as the controller.

For some specific cases, parametric methods have been developed to generate motions that meet high-level goals. Some of these include motion planning such as Lee et al [28] and Koga et al [27], human walking and running such as Girard et al [15], Bruderlin et al [7], and Hodgins et al [22], Miller's snake and worm locomotion [30], and Reynolds' flocking [37]. While many of these methods meet our desires to produce high-quality, goal-directed motion without the need for expertise, each of these has very specific, hence limited, utility. Also, these methods are useful for generating new motions, not editing existing motions.

The focus of all of the above techniques is the creation of motion, mainly from scratch. Some recent work deals more directly with the problem of altering existing motions. Bruderlin and Williams [8] describe how motion editing can be viewed as a signal processing problem and provides a menagerie of methods for manipulating motions. Witkin and Popovic [46] describe a method for adding displacements, scaling and blending motions as well. While these methods are quite powerful, they have limitations that may make them hard to apply. In the following discussion, we will review these methods, some of their shortcomings, and show how these limitations can be addressed.

Guenter et al [21] present an approach that uses spacetime constraint methods to generate transitions between motion segments. Like this paper, spacetime methods are used in conjunction with previously obtained motion, however, unlike our approach, they do not change the previous motion, only generate new motions that provide transitions between existing motions. Because the generated transitions are quite short, they need not consider efficiency, complexity, and specification issues like those presented this in paper.

Gleicher [17] uses a variant of the methods described in this paper to provide interactive editing of motions. The method presented places limitations on the problem that allow for interactive performance. In particular, the interactive editing requires the constraints to be solved initially, allows only a small number of constraints to be changed at any time, and uses uniform representations for the motion displacement curves.

3 Basics

In this section, we discuss the basic methods used for our approach. We begin by considering a single motion signal and describe how we combine prior techniques for direct manipulation curve editing and motion signal processing. Then we discuss controlling multiple signals using our method, using a 2D particle and a 2D human figure as examples. In the next sections we discuss our objective function, describe some useful constraints, and compare our method with other constraint-based methods.

Throughout this paper, we use the notion of a graphical model that is to be animated. The configuration of this model is determined by a vector of parameters, for example consisting of the positions and joint angles of an articulated figure. We denote this vector as $\mathbf{p}$ and the individual, scalar parameters as p_i, or simply as p if there is only one. To animate the model, we vary the parameters over time, denoting the function of time that defines the model's configuration by $\mathbf{p}(t)$, or $p(t)$ for a single signal. We will also refer to these signals as motion curves.

3.1 A Motion Signal

We assume that motion for a model consists of signals (a signal being the time-varying data for a single parameter) that are represented by a set of samples that are potentially sparse. We will call the samples *keys,* although we use the term to describe sampled data from motion capture or generated algorithmically, as well as those manually adjusted using keyframe tools.

To begin, we consider a single parameter of a model. To use a set of sparse samples as motion, we must interpolate them to have a continuous signal which will then be resampled at a target frame rate to produce the animation. Our motion signal is therefore defined by

$$p(t_i) = interp(t_i, keys[]),$$

where $keys[]$ is an array of samples. In this paper we use linear and cubic interpolation, although the discussion is independent of how the $interp$ function works. In fact, $interp$ could be a curve such as a B-Spline that does not interpolate its keys.

3.1.1 Direct Manipulation curve editing

We now consider the problem of altering a motion curve. Suppose that we know what value we would like the parameter to have at a particular time, e.g. we would like to impose the constraint

$$p(t_i) = v,$$

where t_i is a value of time and v is a valid value for the signal. To alter the motion curve we must adjust the keys. If the time of the constraint happens to coincide with a key, making this alteration is easy. Otherwise, we must adjust nearby keys to achieve the desired effect. This has been termed "direct manipulation curve editing" because the methods allow the user to alter a curve directly, not just by its "controls." The methods, such as have been introduced by Fowler and Bartels [13], Welch and Witkin [41], and Hsu et al [23], solve

$$v = p(t_i) = interp(t_i, keys[]) \tag{1}$$

numerically for the values of the keys. For the types of interpolation typically used in computer graphics, $interp$ is evaluated at any given t_i by a linear function of a small number of keys (1 or 2 for linear interpolation, up to 4 for cubics), and Equation 1 will be easy to solve, except for one small complication: there may be many possible solutions. Methods must define a way of choosing which of these is best, typically defining some measure which is optimized. What seems to be most effective for editing is to choose a solution that minimizes the change from the original state. Such an approach is attractive for editing because we prefer to alter curves as little as possible to preserve as much of their original nuances as possible.

The decision of how we measure change is important. A wide range of choices is conceivable. At one extreme of the complexity scale are methods which effectively minimize the amount of work required to find a solution (such as used by

Fowler and Bartels [13]). For more control, we might prefer metrics that measure properties of the curve. Such methods are termed variational methods because they minimize quantities that are integrals over the curve. Variational methods are often approximated by minimizing simple functions of the control points (keys). One simple approach is to minimize the amount the control points move in a least-squares sense. For simplicity, we consider this sum-of-squares objective and we will return to the question in Section 3.4.

The use of a numerical solver allows multiple constraints to be solved simultaneously. This is different than solving for each constraint independently because the solver can find the best solution that satisfies all the constraints. However, if we specify too many constraints, there might not be any solution. In such a case, we prefer to choose a solution that comes as close as possible (again in a least-squares sense) to meeting the constraints.

Direct manipulation curve editing with the simple objective gives us a constrained optimization problem: subject to the curve meeting the constraints, minimize the amount the keys are moved from their initial configuration, e.g.

$$\begin{aligned} \text{minimize } g(keys) &= \frac{1}{2}\sum_{i\in keys}(keys[i]-oldkeys[i])^2 \\ \text{subject to } \mathbf{f}(keys) &= \bigcup_{j\in constraints} v_j = interp(t_j, keys[]) \end{aligned} \tag{2}$$

Because this problem has linear constraints and a least-squares objective function, it can be easily solved, for example using a bi-conjugate gradient iterative solver like the one that appears in Press et al [35].

3.1.2 Additive Motion Editing

Direct manipulation curve editing provides a way to adjust a motion curve at arbitrary points. However, simply adjusting the keys has a severe drawback: the nature of the alteration is an artifact of the key times. Consider an example of a signal that has a constant value over a time interval for which we would like to meet 2 constraints. Figure 1 shows 2 possible problematic outcomes. In the case on the left, the motion is has just two keys (just at the beginning and end), and in the case on the right, the motion is described with a very dense set of keys. The constraints are shown by x's in the figure.

The left case may arise when an animator manually creates the motion and does the obvious process of creating a key at the beginning and end of the sequence. The right case can arise when constant motion has been generated by a

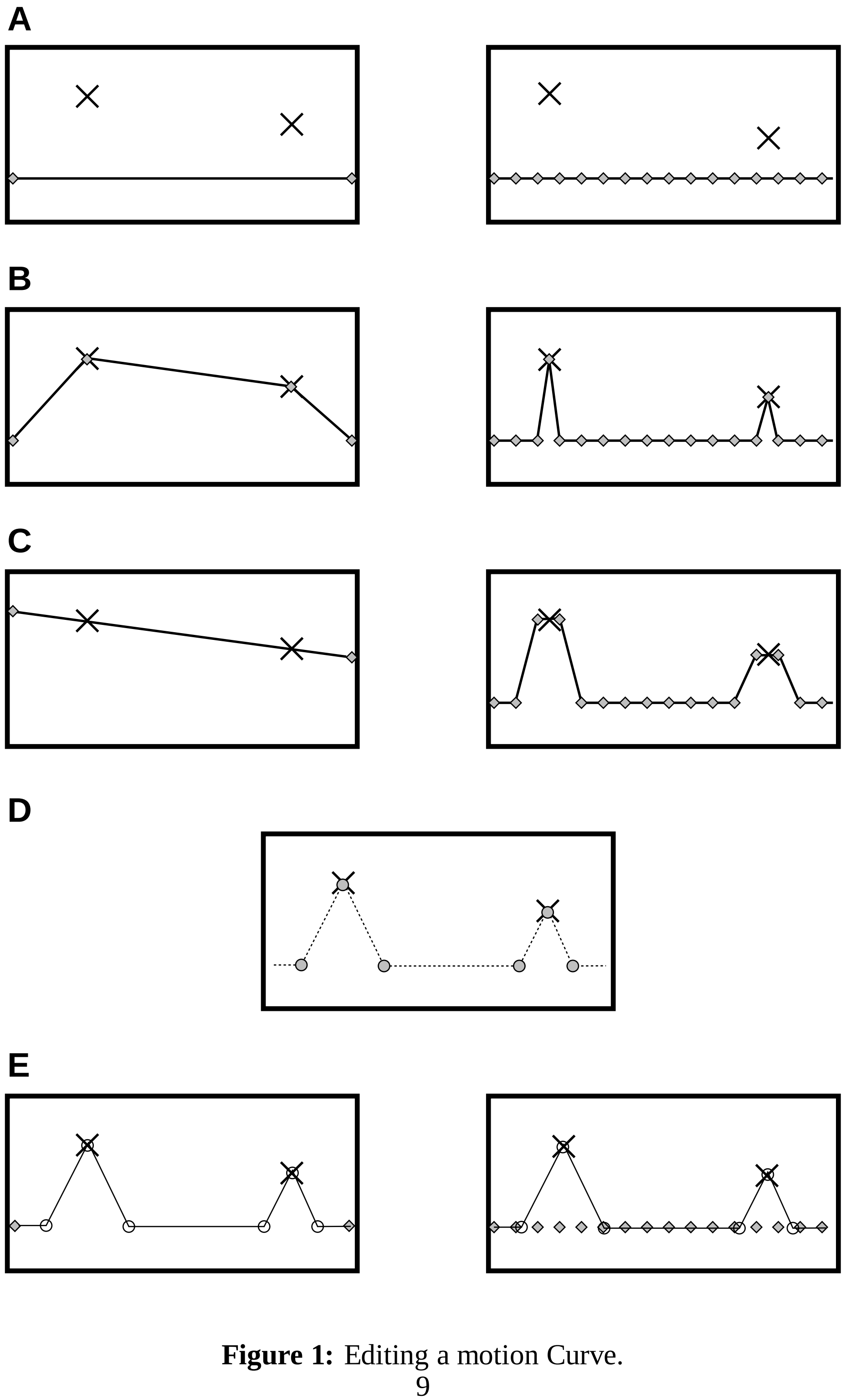


Figure 1: Editing a motion Curve.

A Two straight line curves are to be adapted to pass through goals (denoted by X). The curve on the left has two keys, the curve on the right has many.

B New keys are inserted into the curves so that the new goals are interpolated.

motion tracking system (providing a key at every frame). Such differences arrise because key spacing was determined for some reason other than later adaptation. Because editing the curve, by either inserting new keys or adjusting existing keys, depends on the key spacing, different results occur (Figure 1b and 1c).

Moving or inserting new keys edits the motion to meet new constraints, but often does not provide the desired control. The actual change that occurs is dependent on how the curve was originally constructed, not on the users needs. Ideally, the user should be able to control the scope of the changes.

Motion displacement curves provide a solution to this problem. Rather than editing the keys of the motion curve, which may be inconveniently placed, we modify the curve by adding a new curve, $d(t)$ to the motion so

$$p(t) = interp(t, keys[]) + d(t),$$

and alter the parameters of this new curve instead. This provides freedom to choose functions which alter the motion in desirable ways that do not depend on the representation of the original signal.

Motion displacement maps are a successful variety of displacement curve. As introduced by Bruderlin and Williams [8] and Witkin and Popovic [46], the method uses an interpolating spline for the displacement curve. The keys of this spline can be placed at convenient locations to specify where changes are to occur. By choosing the key spacing for the displacement curve appropriately, the animator can have frequency control over the changes made to the motion. The bottom of Figure 1 shows such a displacement curve, d, that allows us to get the desired shape for both case A and case B.

Many other motion editing techniques can be viewed as variants of the displacement curve approach. For example, motion blending introduced by Bruderlin and Williams [8] and Perlin [33] uses another motion signal as the displacement map. Common to all these approaches is the editing of a signal by adjusting the parameters of a secondary signal.

3.1.3 Additive Editing with Constraints

So far, we have reviewed two previous approaches to motion editing: constraint-based direct manipulation approaches, that provide freedom in specifying goals to be met, and motion displacement mapping, that provides control over how motions are affected. We introduce a novel technique that combine these two to provide the controls of the former with the effects of the latter.

The problem of motion editing with a displacement method involves finding values for the parameters of the displacement map. If we have some requirements for the resulting signal, this may be easy or difficult depending on the type of function representing the displacement curve. For a displacement curve that is a spline with keys at the times of the constraints (as in [46]), determining the required changes is easy. In other cases, it might be more difficult.

We could choose our displacement curve representation based on how easy the curve is to control, but we also must choose it based how the curve affects the motion. These can be conflicting goals: we might need a displacement map curve with distant keys to prevent adding high frequencies, but also need to place a number of constraints at nearby times, or we might want to choose a curve other than an interpolating spline for its smoothness properties. It is our premise that flexibility in what types of displacement curves we choose is important, and their design must be decoupled from our need for an effective interface.

Fortunately, we can provide convenient controls for any displacement curves using the direct manipulation curve techniques of Section 3.1.1. Rather than necessarily solving for new key values, we solve for the parameters of the displacement curve. As before, our solver provides minimum change solutions on underdetermined problems, and least-squares-error solutions solutions for cases where too many constraints have been specified so that no solution can meet them all. This provides a uniform interface to whatever type of displacement curve we desired, on the condition that the solver is capable of handling the equations.

The core of our method is to create a curve that is the summation of the original motion and displacement curves, specify constraints on these summed curves, and then solve for the parameters of the displacement curve to achieve the goals. We pose this as a single constraint problem over the whole motion. It is "global" in the sense that the solution considers all constraints simultaneously, as opposed to considering each frame independently[1].

We denote the vector of parameters that are computed by the solver by $\mathbf{q}$. For editing a motion displacement curve, $\mathbf{q}$ would be the parameters of the curve, for example, the concatenation of its control points. To illustrate this, we give the trivial example shown in Figure 2. Here, we add a signal that is the linear interpolation of 3 key values to an initial signal, and require that the 1st and last key have a zero value, which ensures that the motion connects at the ends. The

[1]In the constrained optimization literature, the term "global" sometimes has a different meaning, refering to algorithms that guarantee the solution is not a local minimum. We do not use such an algorithm.

parameter vector $\mathbf{q}$ for this motion consists of the single value of the middle key. We denote the intitial motion as $p_0(t)$. An example constraint might specify the value of the resulting curve at time 4 to be v_4, which leads to the constraint

$$p_0(4) + .75 * q = v_4,$$

where the .75 factor comes from the fact that frame 4 is 75% of the way between a zero key and the key whose value is q. This constraint is independent of the representation for p_0.

3.2 A Particle

Most graphical models have more than one parameter. We consider the simplest, a particle in 2D. To animate the particle, we need 2 motion signals (which we can view as a vector signal). All motion editing approaches in section 3.1 treated these signals independently.

With a constraint-based approach, we are able to handle all signals (or all the elements of a vector signal) simultaneously. This lets us place constraints on signals that do not just depend on a single parameter. For example, we might want our particle to be a specified distance away from a center point at a particular time. This doesn't necessarily specify the particle's location, it just places a constraint on it — for the example, the particle might lie anywhere on a circle surrounding the centerpoint. Specifying the coordinates of the particle, which is all independent control of its parameters affords, may not be sufficient. For example the distance constraint does not specify where on the circle the particle lies, the exact position may be determined by other things, such as other constraints or the location of the particle at other times.

Multi-parameter models require the ability to use a richer variety of constraints on the motion signals. Rather than simply being able to specify

$$\mathbf{p}(t_i) = \mathbf{v},$$

we add the freedom to specify

$$f(\mathbf{p}(t_i)) = v.$$

The set of variables we must solve over, the state vector, is the concatenation of all of the parameters of the displacement curves for each parameter. The machinery of the previous sections requires little alteration, except that the function

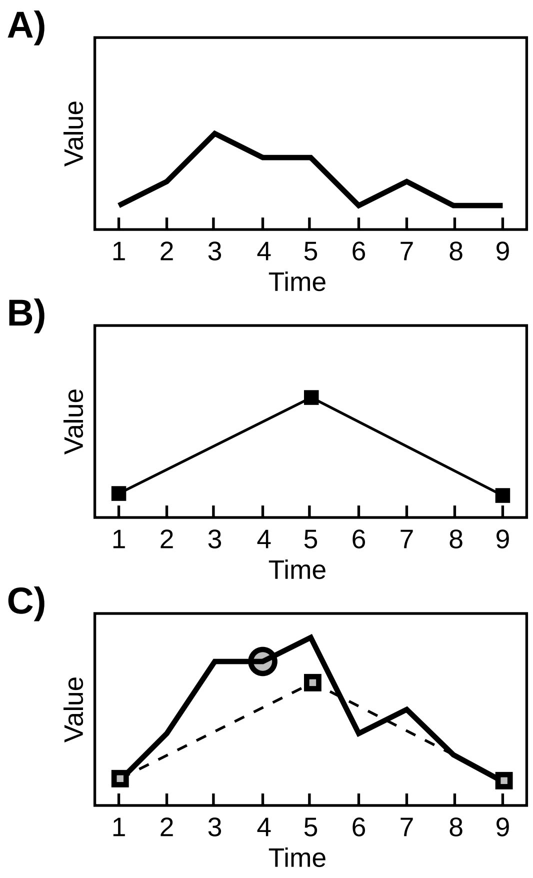


Figure 2: A trivial example of additive editing.

A An initial signal, p_0.

B A displacement curve that linearly interpolates 3 controls, denoted by squares.

C The final curve is the summation of the initial and displacement curves. The value of the key of the displacement curve is computed such that the final curve passes through the constraint, denoted by the circle.

f is likely to be non-linear, which means we need a solver capable of handling such equations. Another useful extension to the solver is to add the facility for inequality constraints such as

$$f(\mathbf{p}(t_i)) \geq v.$$

It is also desirable to be able to add constraints that are enforced over an interval of time

$$f(\mathbf{p}[t \in t_0 - t_1]) = v.$$

To solve such constraints in the continuous time domain requires solving variational calculus problems. To approximate these using the machinery presented, we add an individual constraint for each frame in the time interval. For animationsm this approximation is reasonable because the motion is only sampled at discrete frame times.

3.3 A Character

Most interesting animations require models that are more complicated than a particle. For these problems the methods of the previous section still work, but the number of input variables and the set of useful constraints (and their complexity) grows.

Consider editing the motion of an articulated figure. The parameters of this figure are most likely to include a set of joint angles and a position for the root of the hierarchy. We prefer not to specify these parameters directly. Instead, we prefer to control quantities of interest, such as the position of a hand. Since this position is a function of the model parameters, we can place a constraint on the hand by placing a constraint on the function which calculates the hand position. These constraints are identical to the constraints used with inverse kinematics, the difference being that rather than simply being used to compute the character's pose, the constraints serve to determine the entire motion.

Mathematically, an inverse kinematics constraint specifies

$$\mathbf{e_t} = \mathbf{f}(\mathbf{p_t}),$$

where $\mathbf{e_t}$ is the desired position of the end-effector at time t, $\mathbf{f}$ is the function that computes the position of the end-effector from the characters parameters (e.g. joint angles), and $\mathbf{p_t}$ is the parameters for time t. For inverse kinematics, $\mathbf{p_t}$ would be a key value. We alter the problem by instead solving for the parameters of the displacement curve,

$$\mathbf{e_t} = \mathbf{f}(\mathbf{p_0}(t) + \mathbf{d}(t, \mathbf{q})),$$

where $\mathbf{p_0}(t)$ is the original motion sampled at time t, $\mathbf{d}(t, \mathbf{q})$ is the displacement curve sampled at time time t which is computed from $\mathbf{q}$, the parameters of the displacement curve.

There is a difference between traditional inverse kinematics and the placement of positional constraints on motion signals. Traditional methods use a solver to adjust the parameters at a particular instant in time, while our proposed scheme uses the solver to adjust any parameters of the motion, which may be keys at nearby times, scalings for blended motions, or any other parameters of the motion signals. It is important to see that the two approaches are not equivalent: our approach allows constraints at different times to affect one another.

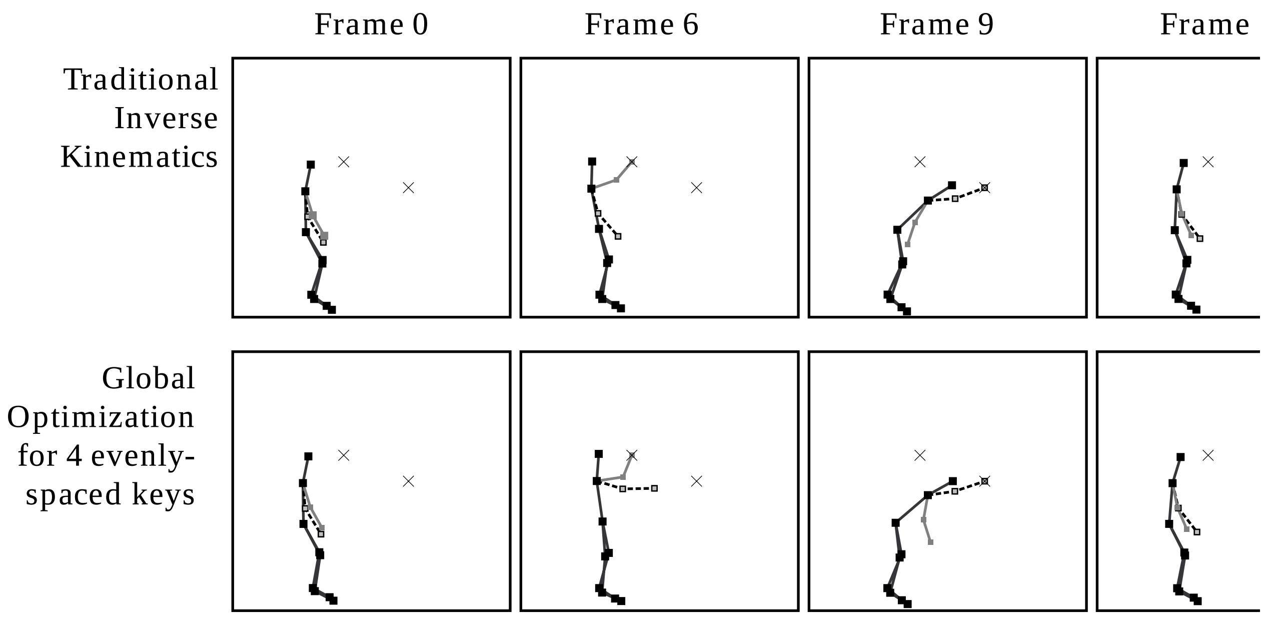
Figure 3 is a simple illustration of the difference. A human figure must reach for two different goals at nearby times. In an interactive system with inverse kinematics (IK), the user would first use IK to position the figure to meet the first constraint, then similarly interactively pose the figure at the second (later) key time. Because the figure doesn't know it will be reaching with the second hand until it has reached with the first hand, the second arm doesn't start moving forward until after the first arm meets its goal. With our technique the figure will more smoothly move its arms to satisfy the second constraint because we solve for the motion over the entire time range. This results in smoother motion that can be more realistic, if the character knows about its future goals. For cases where we prefer the character to be surprised, the windowing methods of [9] prevent the solution for time t_1 to "see" the later constraint.

3.4 Sensitivity Scaling

We now return to the question of what is the "smallest" difference. For our global optimization of motion signal parameters, there are two places where we "choose" among solutions, requiring us to have a notion of what is the best solution:

1. to adjust a motion curve at a non-key time, it may be possible to have the curve meet a desired value by adjusting the set of keys in more than one way.

2. at a particular time, there may be many configurations of the parameters that meet the constraints.



Frame 0
Frame 6
Frame 9
Frame
Traditional Inverse Kinematics
Global Optimization for 4 evenly-spaced keys

Figure 3: An articulated figure reaches for two points.
An articulated figure is instructed to grab the first dot at time 6 and the second dot at time 9 of a 16 frame animation. Frames 0, 6, 9 and 15 are shown for two different methods. In the upper sequence, traditional inverse kinematics are used to position the figure to meet the goals. Because the constraints are solved indepently, they do not take each other into consideration. In the lower sequence, our method is used with a displacement curve that uses cubic interpolation on 4 evenly spaced keys. Notice how the character "plans ahead" by starting to move towards its later goal (look at the position of the non-grabbing hand in frame 6).

As we mentioned in Section 3.1.1, the simplest approach to constraint-based curve editing is to minimize the amount of change in the keys. Such an approach is undesirable because each parameter may affect the resulting animation by a different amount. For example,

1. if a signal is created by linearly interpolating between keys at time 0, 5, 10 and 100, an adjustment of key 5 will cause a much smaller change to the resulting animation than a similar change at key 10, because changing key 5 will affect a much smaller interval of time;

2. an adjustment to the character's hand angle will make much less of a difference than a similar adjustment to the torso, because changing the torso will affect a large part of the body (including the hand);

3. an adjustment of equal numerical magnitude in parameters that are measured in different units, for example millimeters and miles, will have wildly different effects. Comparison between units can be especially difficult when the two parameters do not measure similar quantities, such as meters and radians.

All of these issues can be addressed by choosing an optimization criterion that minimizes a measure of how much the resultant animation changes, rather than just how much the parameters change. This criterion offers a mechanism for defining the kinds of effects various manipulations will have. Currently, we use a weighted least squares metric, as described in [16]. Rather than minimizing the magnitude of the parmeter changes (as in Equation 2), we weight the least squares, giving each individual scalar variable a different weight. That is, we minimize

$$\sum_i \sum_j \frac{1}{w_{ij}} (keys[i][j] - oldkeys[i][j])^2,$$

where w_{ij} is a weighting factor for variable j of key i. We pick the weights such that each of the variables that we control have the same effect. For example, by computing

$$w_{ij} = \sum_t \sum_p \frac{\partial f_p(t)}{\partial keys[i][j]} \cdot \frac{\partial f_p(t)}{\partial keys[i][j]},$$

which computes a weight for each variable that computes the sum (over all times t in the animation and points p on the character) of the magnitude of the change on the animation due to the variable.

The least-squares weighting approximates the objective function that measures the difference between the original and resulting animation, measured by the positions of points on the characters. Because of the numerical methods that we will use in Section 4.1, we would need to create a quadratic approximations to the non-linear objective function. The weights are the diagonal elements of a quadratic approximation to this objective. The quality of the approximation is unimportant, because it is unclear how useful the real objective would be. We propose other objective functions in Section 6, and typically rely on using additional constraints and well designed motion representations, rather than on the objective funtion, to control our results,

3.5 A Menagerie of Constraints

Although we have not experimented extensively with varying optimization objectives to control the resulting motion, we have employed a variety of constraints both to specify our requirements for the resulting motion, and to guide the solver to motions that we prefer. With our general non-linear solver, many constraints are possible (see [44] or [16] for some examples of the utility of the generality of non-linear constraints). Some constraints that we have used in our examples include:

- constrain a point to be a particular place at a particular time;
- constrain a point to be above the floor;
- constrain a point to be at another point's location at a particular time;
- constrain a point to follow another point's motion path;
- constrain two points to have a particular distance (at a particular time or over a range of times);
- constrain an angle between two vectors to be within a range of values (good for joint limits).

3.6 Comparison with other constraint-based methods

Like many other approaches in computer graphics and animation, we use numerical constraint solving to provide a convenient set of controls. As we mentioned in

Section 2, most constraint methods perform solving for individual times, with the notable exception of spacetime constraint methods.

Our method is a variant of the spacetime motion synthesis approach pioneered by Witkin and Kass [45] and Cohen [9]. We use a similar set of constraints, and similar implementation techniques. The fundamental difference between the previous methods and ours is that we do not necessarily generate physical motions, but instead adapt pre-existing motions. For this reason we do not need to include the constraints that insure physical motion, do not need to pose our problems as control instead of placement, and choose objective functions based on motion similarity rather than energy optimality. We believe that this eliminates many of the concerns that hinder the spacetime approach: it is simpler to implement (since we do not need to derive equations of motion), constraint solving is more tractable, it is applicable across a broader domain (not just physically correct motions), and it is potentially easier to use (picking a motion seems a more reasonable task than designing a physical control system).

4 Implementation

One issue that we share with the previous spacetime constraint approaches is that we must set up large, complex constrained optimization problems. How this is accomplished is not important to the method — in fact, we would prefer to hide it as much from the user as possible. What is important is that the methods are fast, robust and that the problems can be defined on the fly in response to the user's requests. We therefore only describe implementation strategies briefly.

4.1 Solver

Our methods rely on the use of a solver for non-linear constrained optimization problems. Good, general methods for such problems are an unsolved problem. Press et al [35] argues that not only does no reliable, general, non-linear solver exist, but that one cannot exist. The difficulty of the general problem has lead to an extensive literature and a wide variety of methods. (we suggest textbooks by Fletcher [12] or Gill et al [14], or the general numerical analysis text by Press et al [35] for a practical introduction).

Non-linear constraint solving has been used in many computer graphics applications, such as inverse kinematics. Solvers used this way are applicable to the problems of our methods with some extensions:

- the equations are more complicated as the end-effector positions at a given time depend not just on a single configuration vector, but some function combining displacement curve controls;
- the problems are larger as we must solve for all time frames simultaneously;
- we must find reasonable solutions to overdetermined problems as they occur frequently.

Our solver must handle problems of the form

$$\begin{aligned} \text{minimize } g(\mathbf{q}) &= \frac{1}{2}\mathbf{q}\mathbf{M}\mathbf{q} \\ \text{subject to } \mathbf{f_e}(\mathbf{q}) &= \mathbf{c_e} \\ \mathbf{f_i}(\mathbf{q}) &\geq \mathbf{c_i}, \end{aligned} \tag{3}$$

where $\mathbf{M}$ is a diagonal matrix, $\mathbf{q}$ is the vector of parameters to the motion displacement curves, and $\mathbf{f}(\mathbf{q})$ is the vector constraint function formed by concatenating individual constraints. We divide this into equality and inequality constraints since the latter must be handled specially. Because of the simplified form of the objective function ($\mathbf{M}$ is a diagonal matrix), and because we are less concerned with accurately minimizing the objectives, we can use simplified methods. The solver must handle over-determined and conflicting constraints.

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 quadratic programs as the approximations. Quadratic programs have linear constraints and quadratic objective functions. We implement inequality constraints using *active set* techniques [12] that operate by switching equality constraints on and off.

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

1. selecting a set of *a*ctive constraints that will be used in the iteration. This set consists of all of the equality constraints, as well as any inequality constraints that might be violated. We use a simple heuristic scheme discussed by Gleicher [16] for selecting which constraints to deactivate. All active constraints are considered as equality constraints.

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

   $$\mathbf{f}(\mathbf{q_i} + \mathbf{\Delta}) \approx \mathbf{f}(\mathbf{q_i}) + \frac{\partial \mathbf{f}}{\partial \mathbf{q}}\mathbf{\Delta},$$

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

   $$\mathbf{J\Delta} = \mathbf{c} - \mathbf{f}(\mathbf{q_i}) = \mathbf{c_i}, \tag{4}$$

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

   A typical SQP solver uses a second order Taylor expansion provides a quadratic approximation to the objective function. Because our objective is a quadratic function, this is not an approximation, we simply re-write the objective in terms of $\mathbf{\Delta}$

   $$g(\mathbf{q_i} + \mathbf{\Delta}) = g(\mathbf{q_i}) + \mathbf{q_i M \Delta} + \frac{1}{2}\mathbf{\Delta^T M \Delta}$$

   In cases where we are uninterested in minimizing the objective, we sometimes ignore the $\mathbf{q_i M \Delta}$ term.

3. solving this quadratic program for $\mathbf{\Delta}$, the step direction vector. If the constraints truly were linear, (or if this approximation is very good), then $\mathbf{q_i}+\mathbf{\Delta}$ would be the answer.

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

This 4 step process is repeated until the algorithm decides to terminate either because the solution $\mathbf{q_i}$ is sufficiently good or because no progress is being made. The stopping tolerances can be set based on the problem's precision requirements.

4.2 Setting up systems

Solving the optimization problem requires evaluating the objective and constraint functions and their derivatives. Symbolic approaches that generate code and require recompilation are undesirable. We would rather generate these systems of equations on the fly in response to user interactions. Thus, we use an approach first introduced by Witkin and Kass [45] and further developed and encapsulated by Gleicher and Witkin [18]. This snap-together-math system provides data structures that represent functional elements that are "wired together" with composition. We extend this idea to wire together entire motions, allowing us to build constraint problems using primitives such as interpolation of keys and time warps in addition to more basic mathematical functions.

The constraint and objective functions for the motion adaptation problems can be 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* [19]. For example, a constraint of the position of the hand of an articulated figure at time t might have the form

$$\mathbf{f}(\mathbf{x}) = \mathbf{e_h}(\mathbf{p}(t, \mathbf{q}))$$

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

$$\frac{\partial \mathbf{f}}{\partial \mathbf{q}} = \frac{\partial \mathbf{e_h}}{\partial \mathbf{p}} \frac{\partial \mathbf{p}}{\partial \mathbf{q}}.$$

Each of the two matrices can be computed independently and multiplied together. We use a general purpose implementation of automatic differentiation [16] [18] 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 $\partial \mathbf{e_h} / \partial \mathbf{p}$, 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.

$$\mathbf{e_h}(\mathbf{x}) = \mathbf{M_0}(\mathbf{x})\mathbf{M_1}(\mathbf{x}) \cdots \mathbf{M_n}(\mathbf{x})\mathbf{h},$$

where each $\mathbf{M_i}$ is a function that computes a $4x4$ transformation matrix and $\mathbf{h}$ is the position of the end effector in its local coordinate frame. The Jacobian can be built by a recursive process based on the product rule for differentiation. Beginning with the end effector, we compute the position and Jacobian of the point in each coordinate frame. Given the position of the point in frame $i+1$ (e.g. $\mathbf{h_{i+1}} = \mathrm{M_{i+1}} \cdots \mathrm{M_n}\mathbf{h}$) and its Jacobian we can compute the next level "up" the hierarchy by the product rule

$$\frac{\partial \mathbf{m_i}}{\partial \mathbf{x}} = \frac{\partial \mathrm{M_i}\mathbf{h_{i+1}}}{\partial \mathbf{x}} = \frac{\partial \mathrm{M_i}}{\partial \mathbf{x}}\mathbf{h_{i+1}} + \mathrm{M_i}\frac{\partial \mathbf{h_{i+1}}}{\mathbf{x}}.$$

We compute each $\partial \mathrm{M_i}/\partial \mathbf{x}$ with a symbolic expession and perform the composition using numerical matrix multiplies. Special case techniques exist for performing these evaluations for rigid kinematic chains (for example one is described in [21]), but we have not yet explored such techniques.

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.

We should emphasize that the user of our system never need see an equation. Unlike the systems described by Witkin and Kass [45], Witkin et al[11], or Kass [25], our function blocks are only data structures inside the system — graphs are created or altered in response to direct manipulation graphical operations on the animation itself. We believe the availability of such an interface is crucial to meet the needs of our target audience.

4.3 Solving the Quadratic Program

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

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

There are many methods available to solve these problems (see texts such as Fletcher [12] or Gill et al [14]). Most methods involve posing the problem as a linear system. The method that we use exploits the fact that M is trivial to invert, and is detailed by Gleicher [16]. 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

$$\mathbf{M}\boldsymbol{\Delta} = \mathbf{J^T}\boldsymbol{\lambda}, \tag{5}$$

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

$$\boldsymbol{\Delta} = \mathbf{M^{-1}J^T}\boldsymbol{\lambda} \tag{6}$$

which we plug into Equation 4 to get

$$\mathbf{JM^{-1}J^T}\boldsymbol{\lambda} = \mathbf{c_i}, \tag{7}$$

a linear system that we solve for $\boldsymbol{\lambda}$, since $\mathbf{c_i}$, $\mathbf{G}$, and $\mathbf{J}$ are known. This is then substituted back into Equation 6 to compute $\boldsymbol{\Delta}$.

If the constraints are over-determined, there is no exact solution and the multipliers are underdetermined. To handle this case, we use a method known as damping or multiplier penalties, discussed by Nakamura [31] and Gleicher [16]. While this method has some numerical disadvantages, it has the advantage that it merely replaces Equation 7 with

$$(\mathbf{JM^{-1}J^T} + \epsilon\mathbf{I})\boldsymbol{\lambda} = \mathbf{c_i}, \tag{8}$$

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, adapted from the one presented by Barret et al [4]. This iterative algorithm allows us to trade accuracy for performance and exploits the sparsity in the matrix.

4.4 Line Search

If our optimization problem were a quadratic program, computing $\boldsymbol{\Delta}$ would provide the answer. For the non-linear problem, $\boldsymbol{\Delta}$ 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. If 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

$$e(\mathbf{x}) = \alpha_1 g(\mathbf{x}) + \alpha_2 \mathbf{f}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x}) \tag{9}$$

that evaluates the quality of a solution. In cases where we are not concerned with the objective, we often set α_1 to 0. 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 $\kappa,$ evaluating the merit function for each. We use a variant of Brent's method [35] 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.

5 Examples

We now consider a number of examples to illustrate our approach. All were created using our prototype system. In cases where prior approaches are shown for comparison, these approaches have been implemented in our system. For many of the examples, we include solution times on a Power Macintosh 8500. For most of the examples, the character being animated is a stick figure with 12 joint angles, a position for the pelvis, and 12 limb lengths which are not permitted to vary in any of the presented examples. Unless otherwise noted, we use cubic interpolation and the objective function using sensitivity scaling described in Section 3.4.

5.1 Jumping

For out first example, we consider altering a jumping motion to meet a desired goal. An animator provided us with a motion of a stick figure jumping. We would like to alter this motion such that the character touches a particular point (denoted by the small cross) at a particular time (frame 20 of the 32 frame sequence). The user can specify this single constraint of this example interactively. The only other thing which must be specified is the type of displacement curve. Figure 4 shows results for a variety of displacement curves, and Figure 5 shows a graph of the X coordinate of the pelvis (the root of the hierarchy).

Figure 4a shows an attempt to use traditional inverse kinematic keyframe editing on the sequence (e.g. no displacement curve). Dragging the position of the hand at time 20 is easy, and provides a good result for this frame. However, because the original motion has a key for every frame of the motion, the alteration only affected one frame, failing to create an acceptable motion.

Figures 4b and 4c show the use of the previous motion displacement technique [46] [8]. The displacement curve has a key at time 20 whose value is computed with inverse kinematics. Two additional zero valued keys limit the effect of the alteration. In Figure 3b, we simply chose to place the new keys 5 frames from the edited key (so there are two zero keys at times 15 and 25, and the constraint key at time 20). In Figure 3c, we place the keys at the takeoff and landing points of the jump so only the flight of the jump is affected (zero valued keys at times 13 and 29). This, in effect, broadened the scope of the change, and introduced uneven key spacing for the displacement curve.

Figure 4d shows the use of our method on our system's default displacement curve with 5 evenly spaced keys over the entire motion. Figure 4e shows a curve with 4 keys evenly spaced only over the time of the jump.

While the creation of Figure 4d and Figure 4e require the generality of our method, it admittedly provides little advantage in this case. The motions are all similar enough that none is definitively best. A subjective poll of some colleagues didn't result in a consistent preference. However, several people complained none seemed realistic because the character is pulled back to the original position, as if connected to the left edge of the frame by a rubber band. We could correct this problem by placing a constraint on the final position of the character, although this would require us to know the length of the jump. We can also correct this flaw by choosing a different displacement map.

Figure 4f and Figure 4g show the use of our method with a "contrived" displacement map. We wish to have the character move continuously in the x direction throughout its flight. To enforce this, we use a displacement curve for the x position of the pelvis that has only two keys — one at the beginning of the flight and one at the end. We permit only the end key to be altered. If the character moves forward while jumping, it must do so continuously across the flight. With our objective function, altering this last key of the x position curve causes more of a change to the animation than other variables because it effects a larger time interval. Due to this, the solver will prefer to change it less than the other variables, resulting in a shortened jump. Manually adjusting the weight on the variable for the x displacement key adjusts the length of the jump (Figure 4g).

Developing the contrived displacement curve of the last paragraph is beyond the skills of much of our potential audience, but so would manually tweaking the motion to generate a good jump. While it did take effort to devise the curve, it is reusable: we could place different constraints on the jump and solve again. In effect, we have created a procedure for creating goal-directed jumping motions. When the effort to devise the displacement map is compared to approaches for generating paramterized motion, the approach seems more reasonable.

We emphasize that for this example, we merely took an existing motion and added a single constraint. The only other thing specified was the type of the interpolating curve used for the displacement. Each variant described can be created by specifying a different displacement map. Solution times for all were less than a quarter of a second.

5.2 Hand Gestures

To show the advantages of solving the inverse kinematics problems on the displaced motion we consider a simple example with two initial motions. We begin with a stick figure standing still, and a point which moves in a square path (linear interpolation between the corner points). We would like the character to trace the path of the point with its hand. In Figure 6a the motion is generated by solving the inverse kinematics problem independently for each frame. This motion accurately tracks the square, but is not smooth — in fact, it contains "jigglies" where the elbow alternates between solutions where the elbow is either straight or bent. In Figure 6b, motion displacement maps [46] are used, placing a displacement key at the corners of the square and solving the inverse kinematics at these 5 keys to determine their values. This creates a motion that is smooth, but does not approximate the square well. In contrast, Figure 6c shows the use of our methods applying a pseudo-variational constraint that ties the figure to the moving point. Because we have chosen a cubic displacement curve with only 5 keys, the figure cannot track the square exactly. However, it gets as close as possible given the limitations. To better approximate the path, a displacement curve with more keys can be used.

To solve this problem we placed two constraints: one ties the hand to a given motion path, and one keeps the pelvis stationary. Our system then generates a positional constraint for the pelvis and hand at each of the 21 frames. Each positional constraint is 2 scalar constraints, so this results in 2 x 2 x 21 = 84 scalar constraints.

5.3 Walking

In this section, we consider adapting a 2D walking motion obtained by rotoscoping video of one of the author's walking. This latter point is significant for comparison with alternate approaches such as synthesis. Our goal in adapting this motion is to obtain new motions that maintain as much of this walk as possible:

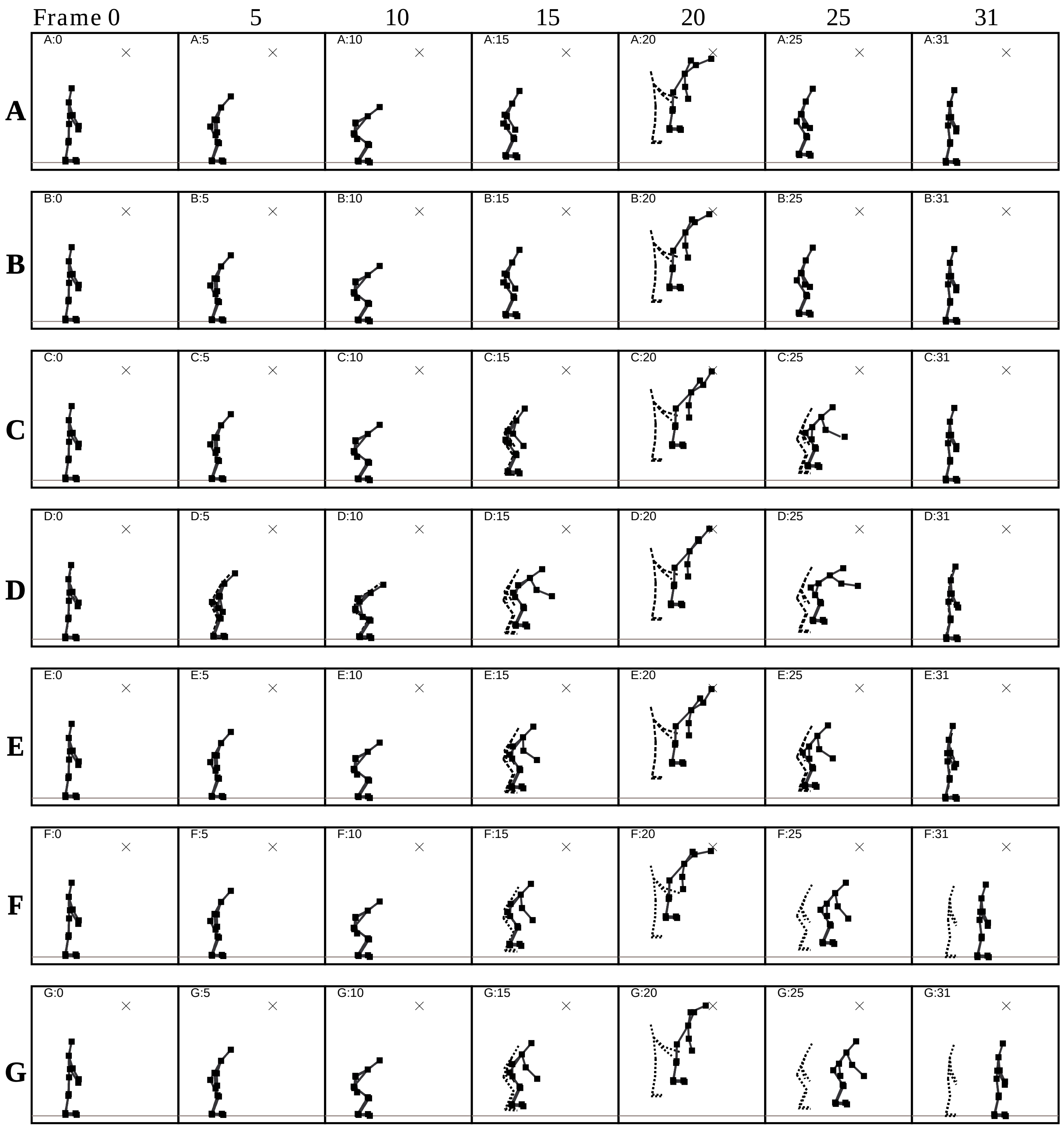


Figure 4: Adapting a Jumping motion

Adapting a jumping motion. A vertical jumping motion (shown in faint blue)is altered so that it touches a spot at frame 20.

A. Inverse kinematics is used to pose figure. Only 1 frame is affected because

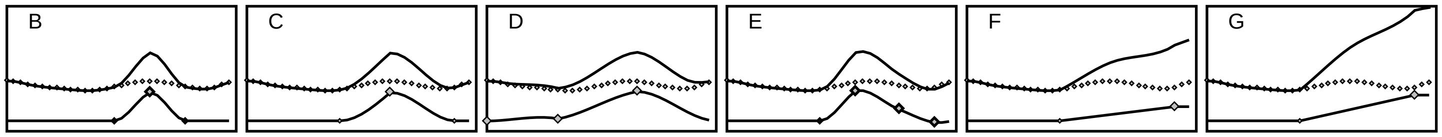


Figure 5: Timeline curves for the jumping motion. These curves graph the pelvis (root of the hierarchy) x-position parameter of the jumping motions shown in Figure 4 Sequences B through G. In each, the upper black curve represents the resulting motion, while the lower curve at the bottom shows the displacement map. The keys of the displacement curves are shown with diamonds: large diamonds represent keys that are permitted to change, while smaller symbols are locked to their zero values. The initial motion is shown as a sequence of locked keys.

the depressed mood of the actor, and the particular quirks of the initial motion. While synthesis tools, such as those available in a commercial tool such as Character Studio [26], would permit new walking motions that meet the constraints to be generated, they would be new motions. While it is realistic to believe that such systems will evolve to permit parameters like “excited” or “depressed” in the walk, it is difficult to imagine generating motion “like Mike would do it when he was unhappy.”

The first adaptation of the walking motion we consider is having the character walk over a step, shown in Figure 7. This example has have 81 frames. We have specified with footplant constraints that correspond with the real foot positions and joint limits. We alter this motion by specifying that one of the footplants is in another location.

To emphasize the difference between our approach and the use of motion displacement maps alone, consider their use on this problem, shown in Figure 8. Both approaches allow the footplant to be relocated at the given frame, but only ours can relocate the entire footplant with one dragging operation. With the previous approach, the interpolation of the changes cause the character to float up to the step.

For the example of Figure 9, we consider changing the initial walking motion into a motion of the character walking across the room to open a door. Rather than specifying the positions of the footplants, we allow the solver to alter them. When the foot is planted, it is not permitted to skid. We add a constraint that positions the character’s hand at the doorknob in the final frame. In addition to having the character reach forward to meet its goal, the solver lengthens the footsteps to create a more reasonable motion.

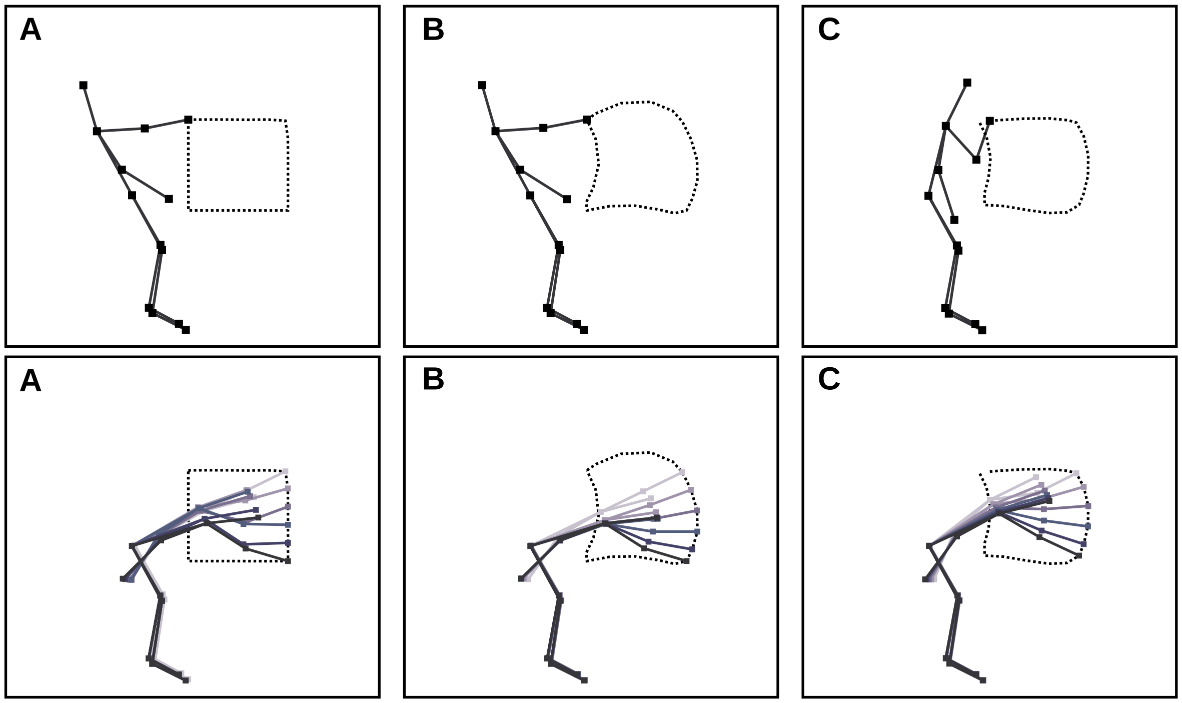


Figure 6: An articulated figure's hand is tied to a moving point that traces a square point. The dotted line shows the path actually traced by the character's hand, and the lower pictures show the character's position strobed from time 5 to 10. (a) Inverse kinematics applied to each frame individually. Although the character traces the path precisely, the motion is "jiggly."Note that the arm oscillates between being bent and straight. (b) prior motion displacement techniques are used, with displacement keys set with inverse kinematics at the four corners. This figure uses an interpolating cubic spline with 5 keys. (c) Our method with 5 keys for the displacement map and a variational constraint tying the hand to the square motion path. The resulting motion is smooth and approximates the square. Because the displacement has limited degrees of freedom, the square cannot be traced perfectly. This figure also uses an interpolating cubic spline with 5 keys. A displacement curve with more keys would produce a curve that better traced the path.

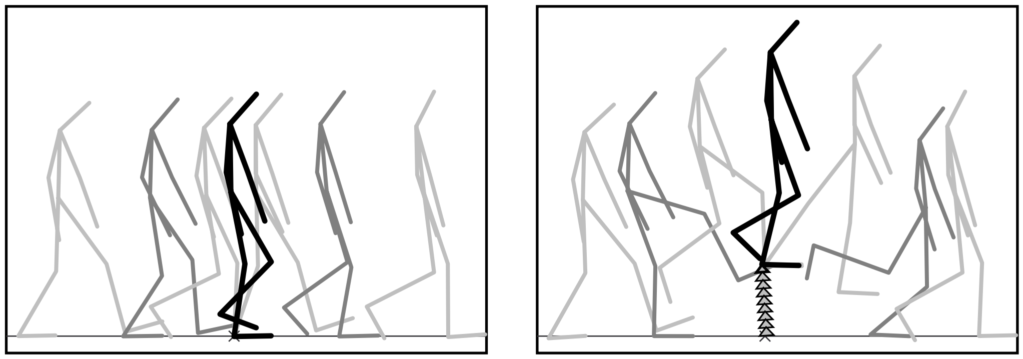


Figure 7: A walking motion is shown on the left by “strobing.” The initial walking motion is shown on the left. The character is made to step over an obstacle by changing the location of the footplant constraint. The arrowed path shows where the footplant has been dragged.

Figure 8 goes here

Figure 8: The example of Figure 7 is detailed. The left shows the use of our method to relocate the 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 footstep’s position at a 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.

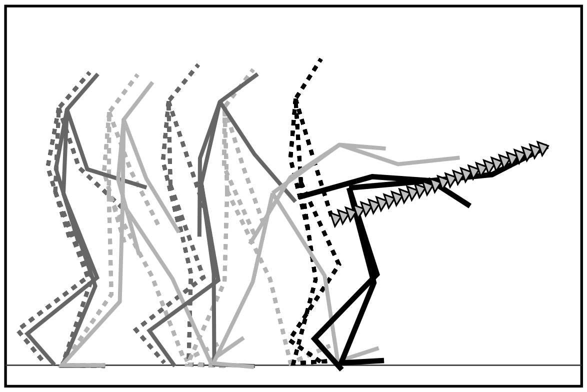


Figure 9: The first 55 frames of the walking motion of Figure 7 are adapted to have the character reach for a spot in the last frame. In this example, the solver is permitted to adjust the footplant positions. The dotted figures represent the original motion, the arrows indicate the single constraint dragging operation. Notice that positioning the characters hands not only causes him to lean forward but to take larger steps as well.

5.4 Switching characters

This example of Figure10 starts with two base motions: short and tall figures walking. We wish to make the short man walk in the tall man's footsteps. The period of the footfalls match by design. However, if they didn't, we could make them match by using the dynamic time-warping algorithm presented by Bruderlin and Williams [8]. The constraints provided to our method were:

1. tie the x-position of the short figure's torso to that of the tall figure;
2. make the short figure's footpaths (specifically, the ankles) follow the taller figure's foot paths;
3. maintain realistic joint angles (the arms, knees, feet and hands can't bend backwards);
4. maintain all body parts above the floor.

This example shows how the presented algorithm is used to modify an original motion so that end-effectors meet user-supplied goals, while retaining much of the original character of the motion. Figure 10d shows the resulting motion. Figure

10e shows a similar example, the only difference being that for our initial motion we have the short figure running. The footfalls of the tall character are large enough that it is a stretch for the small character to meet them.

5.5 3D Examples

Thus far, we have demonstrated our approach with 2D examples. We note, however, that there is nothing about our approach that precludes the more interesting case of 3D character animation. The basic elements of our approach have been demonstrated in 3 dimensions, both spacetime constraints [9] and motion displacement mapping[2] [8] [46]. The system described in this paper has been extended for 3D character animation [17]. An illustration of this is shown in Figure 11, where a 3D motion of a walking character obtained by optical motion capture is altered by repositioning footplant constraints.

The most unique challenges in employing our constraint-based approach to motion editing to 3D tasks are in the user interface. Specifying and visualizing the changes to the 3D motions are a difficult task, especially in the constraint-based approach where a given frame might be affected by constraints on other frames, and similarly, each constraint may affect a number of frames on the motion. There are some potential technical challenges as well, for example, the quaternion representation typically preferred for 3D rotations does not have the addition operator defined, therefore their use in a motion-displacement approach is non-trivial.

6 Discussion, Conclusions and Future Directions

In this paper, we have presented a constraint-based approach to motion adaptation. With these methods, we can specify a set of goals that a motion needs to meet, and have a pre-existing motion adapted to meet these needs in a way that preserves the initial motion. Such methods can empower users without animation skill, allowing them to create animations by selecting motions from libraries. The methods also enable scenarios where motions are created on the fly, without the intervention of an animator.

In the course of developing our prototype implementation, experimenting with it on a number of examples, and assessing the methods, we have identified a number of issues that might be better addressed:

[2]In fact, the technique has not been demonstrated in the simpler, 2D case.

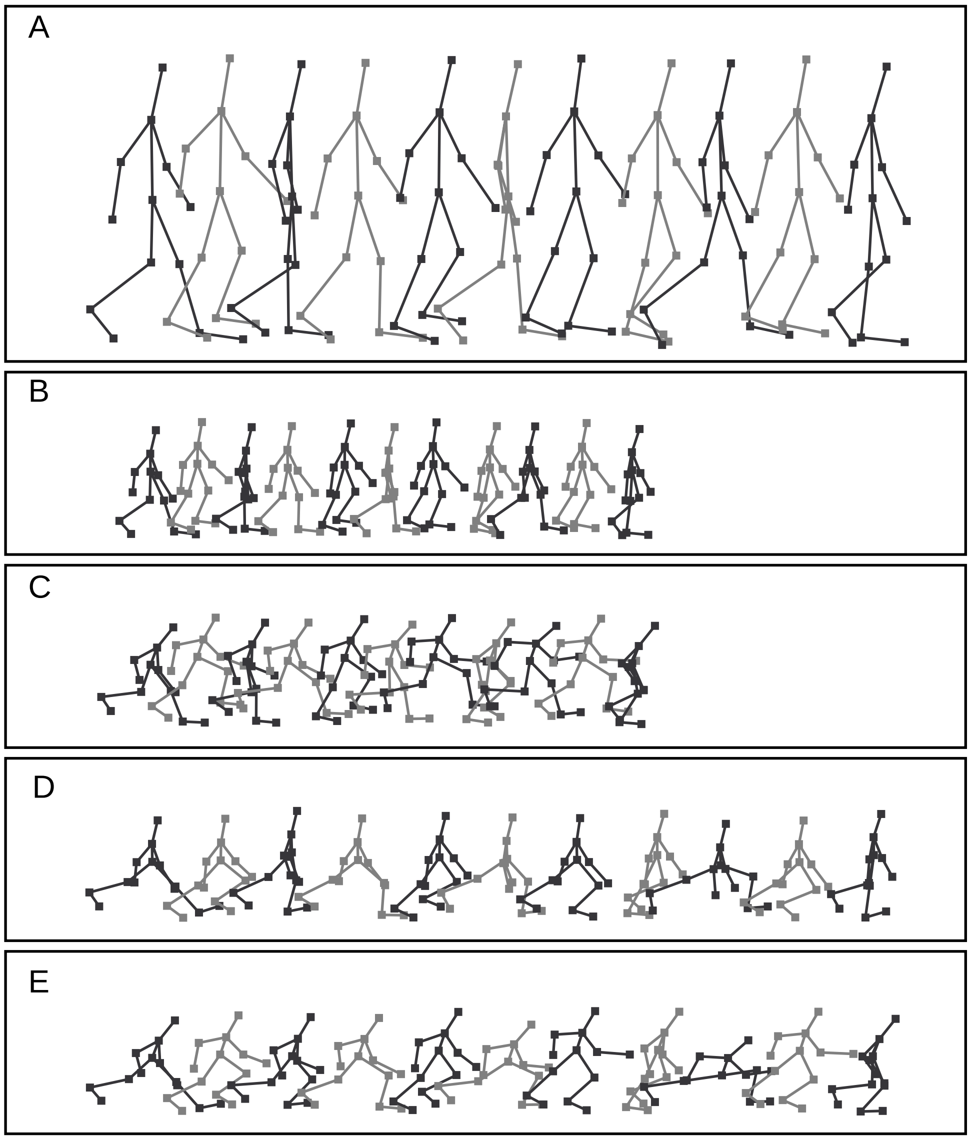


Figure 10: Switching characters by constraint-based motion adaptation. Walking motions are shown for a tall character and a short character by strobing every 5 frames in alternating shades of grey.

- **A.** The initial walking motion of a tall character.
- **B.** Initial walking motion of the small character.
- **C.** Initial running motion for the small character.
- **D.** Adapting the short character's walk (B) to meet the footplants of the taller characters walk (A).
- **E.** Adapting the short character's run (C) to meet the footplants of the taller characters walk (A).

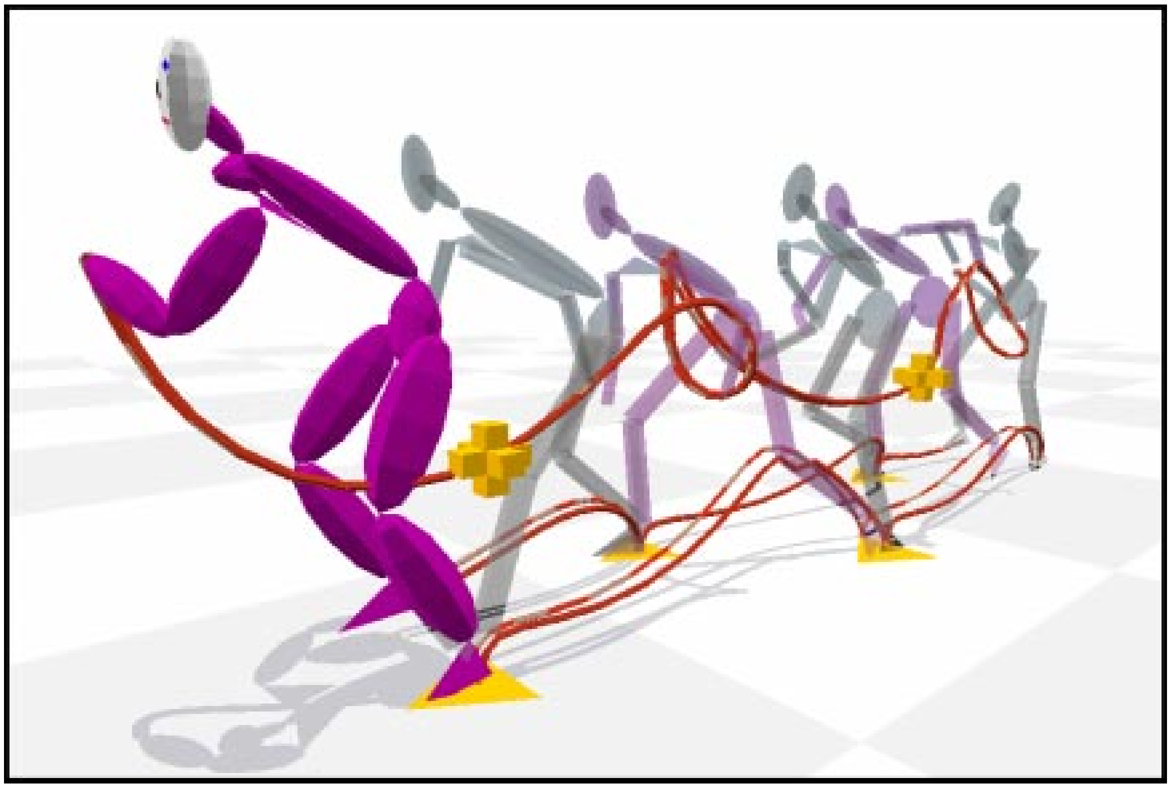


Figure 11 goes here

Figure 11: 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. Yellow symbols represent constraints.

Usability — Our approach offers advantages in that it allows the use of convenient constraint-based controls, like inverse kinematics. However, it also requires a variety of new control types to adjust the motions. To achieve a desired motion, one may need to change the key spacing or function type for the displacement curve, select an alternate initial motion, or adjust the parameters of the objective function. We selected displacement curve types empirically — we typically would try one or two of the default choices and pick the motion we liked best. A good interface which allows defining more complicated functions or key spacings for the displacement curves is an open question.

We believe that choosing an initial motion is an intuitive control over the resulting motion. As seen in the examples, we can pick initial motions with properties that we want to see in the final animation.

Objective Functions — The choice of objective function provides a way to control the types of motions the solver will find. To date, our emphasis has been on specifying better motions through increased constraints. However, Witkin and Kass [45] suggest how proper choice of objective functions might translate into high level goals. While objective like “as cautiously as possible” seem out of reach for current methods, basic hints such as making the solver prefer to keep the knees bent or the figure balanced, could be added to our system, although may prove difficult to present to users.

Better objective functions can also simplify the problem of choosing representations for the displacement curves. If the objective is well tuned to desireable motions, for example to prefer smooth displacements, there is less of a need for the displacement function to provide the preferences.

Performance — On simple examples, our prototype is capable of providing real-time, direct manipulation with feedback. The user can drag a single frame and continuously see the changes made to the entire motion. Faster hardware and more sophisticated implementation techniques might make such interactions possible for more complicated problems.

Robustness — Solving general non-linear constrained optimization problems is an intractable problem. Any method we use cannot guarantee solving every problem we pose. So far, we have had good results with a relatively simple solver. Better ones are both publicly and commercially available. We are less concerned, however, with cases where our solver cannot find any solution, and more concerned with cases where the solver finds a correct solution that is not the desired motion. Usually, this requires better specification of the problem, by adding more constraints or providing other hints to the solver.

Time-dependent constraints and objectives — We could add constraints and metrics that compute functions of the motion, rather than the configurations. For example, we might place a limit on the velocities of a point, or prefer that a point follow a smooth path. Such functions would depend on a number of adjacent frames. Requiring that a point follow a physically valid path seems to be a special case of this, which offers the potential of integrating our methods with traditional spacetime methods.

In conclusion, we have described a method which extends previous work on motion editing by combining motion warping with constraint satisfaction, resulting in a system that provides retention of the original qualities of a motion while mapping it into a new scene or onto a new character. This technique will empower the novice animator, and provide better starting motion for the experienced animator. While not quite yet real-time, we envision a system which will provide "on the fly" animation for interactive scenarios, as well as providing tools for animation authoring systems.

Acknowledgements

This research was conducted while the authors were members of the Graphics Research Group of Apple Research Laboratories. We thank our manager, Gavin Miller, for his support of this project. Sebastian Grassia and Will Welch provided advice on numerical issues. Mark Halstead provided wavelet B-Spline code. Kim Tempest animated the walking, running and jumping motions in Figures 4 and 10. The motion capture data used for Figure 11 was provided by BioVision Motion Capture Studios.

References

[1] William Armstrong and Mark Green. The dynamics of articulated rigid bodies for purposes of animation. *The Visual Computer*, 1:231–240, 1985.

[2] Norman Badler, Kamran Manoocherhri, and Graham Walters. Articulated figure positioning by multiple constraints. *IEEE Computer Graphics and Applications*, pages 28–38, June 1987.

[3] David Baraff. Coping with friction for non-penetrating rigid body simulation. In *Computer Graphics* (*Proc. SIGGRAPH*), volume 25, pages 31–40. ACM, July 1991.

[4] 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.

[5] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constaints. *Computer Graphics*, 22:179–188, 1988. Proceedings SIGGRAPH '88.

[6] Lynne Shapiro Brotman and Arun N. Netravali. Motion interpolation by optimal control. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 309–315, August 1988.

[7] Armin Bruderlin and Thomas W. Calvert. Goal-directed, dynamic animation of human walking. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 233–242, July 1989.

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

[9] 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.

[10] John Craig. *Robotics: Mechanics and Control*. Addison-Wesley, 1986.

[11] Kurt Fleischer and Andrew Witkin. A modeling testbed. In *Proc .Graphics Interface*, pages 127–137, 1988.

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

[13] Barry Fowler and Richard Bartels. Constraint-based curve manipulation. *IEEE Computer Graphics and Applications*, 13(5):43–49, September 1993.

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

[15] Michael Girard and Anthony A. Maciejewski. Computational modeling for the computer animation of legged figures. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 263–270, July 1985.

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

[17] Michael Gleicher. Motion editing with spacetime constraints. In Michael Cohen and David Zeltzer, editors, *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 139–148, apr 1997.

[18] 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.

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

[20] 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.

[21] 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.

[22] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 71–78. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[23] William M. Hsu, John F. Hughes, and Henry Kaufman. Direct manipulation of free-form deformations. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 177–184, July 1992.

[24] Paul Issacs and Michael Cohen. Controlling dynamics simulation with kinematic constraints, behavior functions and inverse dynamics. *Computer Graphics*, 21(4):215–224, 1987. Proceedings SIGGRAPH '87.

[25] Michael Kass. CONDOR: constraint-based data flow. *Computer Graphics*, 26:321–330, July 1992. Proceedings SIGGRAPH '92.

[26] Kinetix, a divison of Autodesk, Inc. Character studio 1.0. Computer Program, 1996.

[27] Yoshihito Koga, Koichi Kondo, James Kuffner, and Jean-Claude Latombe. Planning motions with intentions. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 395–408. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[28] Philip Lee, Susanna Wei, Jianmin Zhao, and Norman I. Badler. Strength guided motion. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 253–262, August 1990.

[29] 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.

[30] Gavin S. P. Miller. The motion dynamics of snakes and worms. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 169–178, August 1988.

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

[32] 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.

[33] Ken Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, March 1995. ISSN 1077-2626.

[34] John Platt. A generalization of dynamic constraints. *CGVIP: Graphical Models and Image Processing*, 54(6):516–525, November 1992.

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

[36] W. T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, 2:91–108, April 1983.

[37] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 25–34, July 1987.

[38] Karl Sims. Evolving virtual creatures. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 15–22. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[39] Ivan Sutherland. *Sketchpad: A Man Machine Graphical Communication System*. PhD thesis, Massachusetts Institute of Technology, January 1963.

[40] 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.

[41] William Welch and Andrew Witkin. Variational surface modeling. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 157–166, July 1992.

[42] Chris Welman. Inverse kinematics and geometric constraints for articulated figure manipulation. Master's thesis, Simon Frasier University, September 1993.

[43] Jane Wilhelms. Using dynamic analysis for realistic animation of articulated bodies. *IEEE Computer Graphics and Applications*, pages 12–27, June 1987.

[44] Andrew Witkin, Kurt Fleischer, and Alan Barr. Energy constraints on parameterized models. *Computer Graphics*, 21(4):225–232, July 1987.

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

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

[47] Jianmin Zhao and Norman Badler. Inverse kinemtics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics*, 13(4):313–336, Oct 1994.