Constraint-Based Motion Adaptation

Michael Gleicher and **Peter Litwinowicz**
Advanced Technology Group

June 14, 1996

Apple TR 96-153

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. This allows the adaptation of motions to meet new goals while retaining much of their original quality.

Constraint-based Motion Adaptation

Michael Gleicher and **Peter Litwinowicz**
ATG Graphics Group
Apple Computer
June, 1996

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. This allows the adaptation of motions to meet new goals while retaining much of their original quality.

Keywords: computer animation, motion adaptation, constraint solving, motion displacement and warping

1 Introduction

Creating high quality motion for animation is a tedious and difficult task [Las87]. 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 some other method. Such 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.

One promising approach to motion adaptation, presented by Bruderlin and Williams[BW95], treats motions as signals and applies some traditional signal processing 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[WP95]. Unfortunately, as initially presented the methods fall short of being able to realize our goals. In particular, they fail to provide the types of control we desire over the adaptations and provide no way to guarantee that desired properties are maintained in the adapted motion. In this paper, we present an approach to motion adaptation that extends the originally published methods to better address these shortcomings.

This paper presents *constraint-based motion adaptation*, an approach that uses numerical constraint techniques to alter motions so that they meet desired goals while preserving as much of the original character as possible. We combine elements of several prior approaches. We use motion signal processing methods to alter motion signals in ways that preserve desired qualities. We couple this with constraint-based direct manipulation (a generalization of inverse kinematics) that 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 methods of [Wk88][Coh92], we solve a constraint problem for the entire motion, unlike most 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. 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 follows, first by describing how some previous motion adaptation methods can be viewed under a common framework and controlled using constraint solving. Extensions to these methods provide additional flexibility in the types of control. We discuss a variety of applicable constraints. Following a brief discussion of implementation details, we provide a number of examples created with our approach, chosen both to illustrate the benefits of our approach and to suggest the range of applicability of our methods. 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. Basic robotics texts present methods for solving inverse kinematic problems [Cra86][Pau81]. Many commercially available animation systems, such as provided by Alias and SoftImage, include this capability.

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 to be satisfied simultaneously. Constraint solving has a long history in computer graphics, dating back to Sketchpad[Suth63]. The utility of solving multiple constraints for positioning figures in animations was first shown by Badler et al [BMB86]. The usefulness of a more general class of constraints was first examined by Witkin et al [WFB87].

Physically-based methods attempt to create realistic motion by simulating the laws of physics[AGL85][Ree83]. As the field has progressed, better and better simulation methods, such as those discussed in [Bar91] 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 [AGL85][Wil87][IC87] and dynamic constraints [BB88][Pla92].

Almost all constraint-based approaches apply constraints to individual instants in time to either compute the needed configurations to 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 [WK88] and [BN88]. 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 [WK88]), 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 they have produced some exciting results, spacetime methods 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 [LCG94] shows methods to improve the tractability). A potentially more pressing issue is that an optimally efficient, physically-correct motion is not always what is most desirable for animation. As we will discuss in Section 3.6, our work has much similarity to spacetime constraints, yet avoids these restrictions.

An alternative approach to spacetime attempts to design controllers for models, rather than their motions. [NM93] and [VF93] present methods that compute controllers for different characters, but do not necessarily have these characters produce controlled motions. [GT95] uses spacetime methods to design controllers for more complex creatures that are more capable of meeting desired goals. [Sims94] extends spacetime to design the creature as well as the controller. We will call these methods spacetime controller methods, to distinguish them from the spacetime constraint methods of [WK88] and [LCG94].

For some specific cases, parametric methods have been developed to generate motions that meet high-level goals. Some of these include motion planning [KKKl94] [LWZB90], human walking and running [GM85] [BC89][HWBO95], snake and worm locomotion [Mil88], and flocking [Rey87]. 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 (limited) utility.

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. [BW95] describes how motion editing can be viewed as a signal processing problem and provides a menagerie of methods for manipulating motions. In [WP95], the authors 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.

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 $\mathbf{p}_i$, 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.

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 constant value of time and v is a valid value for the signal. To alter the motion curve we can simply 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 [FB93][WW92][HHK92], solve

$$v = p(t_i) = interp(t_i, keys[\ \ ]) \qquad (1)$$

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 the equation 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 in concievable. At one extreme of the complexity scale are methods which effectively minimize the amount of work required to find a solution (such as [FB93]). 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.

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.

$$\text{minimize } \sum_i (keys[i] - oldkeys[i])^2 \qquad (2)$$

$$\text{subject to } \bigcup_{j \in constraints} v_j = interp(t_j, keys[\ \ ]).$$

Because this problem has linear constraints and a quadratic objective function, it can be easily solved, for example using an iterative solver like the one that appears in [PTVF92].

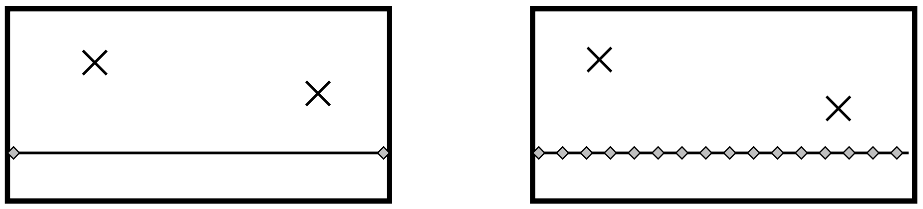
Because we have decided to solve the curve editing problem with a numerical solver, we can submit multiple constraints and solve them simultaneously. This is different than solving for each constraint independently because the solver can find the best solution that satisfies all the constraints. A complication here is that 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. The bi-conjugate-gradient solver from [PTVF92] does this for us as well.

3.1.2 Additive Motion Editing

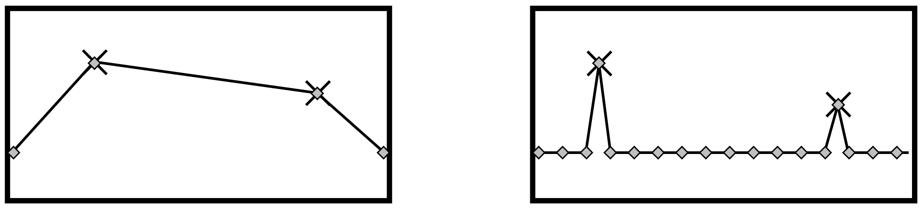
Direct manipulation curve editing provides a way to adjust a motion curve at arbitrary points. However, simply adjusting the keys has a severe drawback: the nature of the alteration is an artifact of the key times. Consider an example of a signal that has a constant value over a time interval for which 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 thing 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 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).

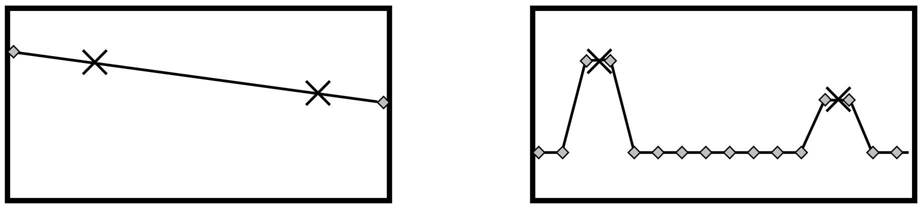
Moving or inserting new keys edits the motion to meet new constraints, but often does not provide the desired control. What we really want is to pass through the two points, while maintaining control over the spread of the changes. When adding or moving keys, the actual change that occurs is dependent on how the curve was originally constructed, not on the users needs.



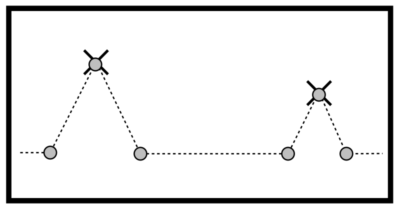
a) Two straight line curves are to be adapted to pass through goals (denoted by X). The curve on the left has two keys, the curve on the right has many.



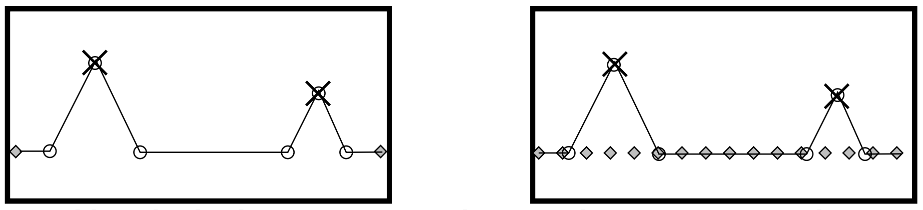
b) New keys are inserted into the curves so that the new goals are interpolated. The results vary depending on the initial key setup.



c) Existing keys are altered so that the goals are interpolated. The minimal change to the keys is used. Depending on the key setup, the results vary.



d) A displacement curve is chosen with keys that produce the desired effects, and zero values elsewhere.



e) Adding the displacement curve to the original curves gives a result independent of the original key setup.

Figure 1: Editing a motion signal

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 in [BW95] and [WP95], 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 [BW95][Per95] 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. The control of this secondary curve might be a scaling factor, in the case of blending, or the displacement curve's control points. Note that the interpolating spline displacement curves are a special case of blending where the blended functions are the basis functions of the spline.

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 [WP95]), determining the required changes is easy. In other cases, it might be more difficult (how much running motion do we mix into a walk to make a character clear a hurdle?).

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. Evidence for this is provided in the examples of Section 5.

Fortunately, we can provide convenient controls for any displacement curves using the direct manipulation curve techniques of section 3.1. The problem is only slightly different: rather than necessarily solving for new key values, we solve for the parameters of the displacement curve, whatever they may be. All of the extensions discussed previously still apply: we can use minimization to provide smallest change solutions when our problems are underdetermined, and use least-squares-error to provide best-fit solutions for cases where too many constraints have been specified so that no solution can meet them all. Notice that 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, not just solutions that only consider each frame independently. [1]

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 particular distance away from a particular place at a particular time. This doesn't necessarily specify the particle's location, it just places a constraint on it. Specifying the coordinates of the particle, which is all

1. In the constrained-optimization literature, the term "global" sometimes has a different meaning. In the sense other than ours it denotes a class of solving algorithms that exhaustively search for extrema.

independent control of its parameters affords, may not be sufficient. For example, at time t_1, we might desire that the particle meets a distance constraint, but where exactly on the circle it lies may be determined by other things, such as other constraints or the location of the particle at other times.

All that we have added is the ability to place a richer variety of constraints on the motion signals. Rather than simply being able to specify

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

we add the freedom to specify

$$f(\boldsymbol{p}(t)) = v.$$

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 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

$$f((\boldsymbol{p}(t)) > v).$$

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

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

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 animations we only sample the resulting signal at discrete frame times, so this approximation seems reasonable.

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.

Mathematically, the inverse kinematics problem solves

$$\boldsymbol{p}_t = \boldsymbol{f}(q_t),$$

where $\boldsymbol{p}_t$ is the desired position of the end-effector at time t, f is the function that computes the position of the end-effector from the characters parameters (e.g. joint angles), and $\boldsymbol{q}_t$ is the parameters for time t (e.g. a key). We alter the problem by instead solving for the parameters of the displacement curve, replacing $\boldsymbol{q}_t$ with the function that computes the value of the displaced motion curve,

$$\boldsymbol{p} = \boldsymbol{f}(\boldsymbol{s}_t + \boldsymbol{d}_t(\boldsymbol{q})),$$

where $\boldsymbol{s}_t$ is the original motion sampled at time *t*, $\boldsymbol{d}_t(\boldsymbol{q})$ is the displacement curve sampled at time time *t*, and $\boldsymbol{q}$ is now 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 2 illustrates 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 [Coh92] prevent the solution for time t1 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.

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 anapproach 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 100, 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 criteria 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 [Glei94]. 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 $i \times j$ 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. This weighting approximates the objective function that minimizes the difference between the original and resulting animation by measuring points on the characters, using only the diagonal elements of the matrix to speed computation. We propose other objective functions in Section 6.

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 [WFB87] or [Glei94] 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 2D 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 like spacetime constraints in this regard. We solve a single, potentially large, constraint problem to compute the entire motion.

Our method is a variant of the spacetime motion synthesis approach [WK88] [Coh92]. 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 motion (as opposed to controller) 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. [PTVF92] 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 [Fle87][Gil81] or even [PTVF92] 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 are a little bit more demanding about finding reasonable solutions to overdetermined problems.

We been using a solver that was originally designed for generalized inverse kinematics [Gle94] with trivial modification. The algorithm linearizes the constraints, solves a linear least-squares problem to define a search direction, and does a line search. The linear subproblems are solved with a damped Lagrange multiplier method [Wam86][Nak91]. A more traditional sequential-quadratic programming solver (SQP)[Fle87] is likely to be more appropriate.

The numerical problems that must be solved for our approach would appear large enough to be beyond the capabilities of desktop computers. However, the problems have many properties that allow efficient solutions, such as sparsity and partitioning. Many of these properties, and methods to exploit them, are discussed in [Gle95]. By paying careful attention to these implementation details we can build a system that is interactive, is capable of solving interesting problems in reasonable amounts of time, and that runs on modest computers. Timings for our prototype implementation are given in the examples.

4.2 Setting up systems

The constraint solver must be given functions that compute the errors of the constraints and the magnitude of the objective function. For most solvers, we must not only have the facility to evaluate these functions, but also 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 [WK88] and further developed and encapsulated in[GW93]. 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.

We should emphasize that the user of our system never need see an equation. Unlike [WK88], [FW88], or [Kass92], 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.

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 8500/120. 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 that minimizes the sum of the squares of the endpoints of each line segment of the figure.

5.1 Jumping Example

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 3 shows results for a variety of displacement curves, and Figure 4 shows a graph of the X coordinate of the pelvis (the root of the hierarchy).

Figure 3a 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.

Figure 3b and 3c show the use of [WP95] and [BW95]'s motion displacement technique. 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 3d shows the use of our method on our system's default displacement curve with 5 evenly spaced keys over the entire motion. Figure 3e shows a curve with 4 keys evenly spaced only over the time of the jump.

While the creation of Figure 3d and Figure 3e 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 3f and Figure 3g 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 3g).

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 resolve. 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, allowing for real-time, direct manipulation dragging.

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 5a 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 5b, motion displacement maps from [WP95] 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 5c 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. Our method solves this problem in under a second.

5.3 Switching characters

This example of Figure 6 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 in [BW95]. 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 6d shows the resulting motion

Figure 6e shows a similar example, the only difference being that for our initial motion we have the short figure running (figure 6c). The final motion is noticeably different, appearing much more like a running motion than the motion of figure 6a. Computation times are shown in Table 1.

	Walk adaptation	Run Adaptation
Number of Frames	101	101
Total Scalar Constraints	2929	2929
Displacement Keys	20	50
Time to solve (seconds)	17	21

Table 1:

5.4 Turning a human into a kangaroo

In this example we map motion from animals onto human stick figures. In Figure 7 we generate motion solely from the motion of a kangaroo... there is no starting motion for the human stick figure. This shows that the technique presented here can map motion from one character to another even when there is no original motion for the target figure.

We also demonstrate that animators are allowed to pick and choose what motion they want to have mapped to their character. In this example we ignore the knees and elbows of the kangaroo and let the constraint solver fill in that data for the human figure. At each frame we have tied the ankles of the human figure to the feet of the kangaroo, the hands to the hands, pelvis to pelvis, shoulder to shoulder, and head to head. The constraints cannot all be met, but the solver does its best, and produces reasonable motion.

Note that the kangaroo video is not have a true cycle: the first and last frames are not identical. To create a cyclical motion, we have ignored the position of the hands in the first and last frames of the video and placed them in a reasonable position (this explains why the hands of the stick figures do not accurately track the video).

Figure 8 uses the same set of constraints to constrain the human figure to the motion of the feet, pelvis, shoulder and head of a horse. The result is a human that crawls like a horse gallops. For the 33 frame cycle there are 1551 scalar constraints. Our solver took 11 seconds on this example.

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:

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. Changing displacement curve type is often easy to determine 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. For the kangaroo of Section 5.3 we originally started with a figure with straight legs which gave unsatisfactory results. It was fairly easy for us to decide what to change, since the resulting motion never really had bent knees. Starting with the figure with bent knees solved the problem, producing the motion shown.

Objective Functions – The choice of objective function provides a way to control the types of motions the solver will find. [WK88] 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.

3D – We see no major difficulties in extending our prototype to handle 3D animations, with the exception of user interface – it will be considerably harder to construct a direct manipulation interface for adjusting characters, specifying constraints, and providing feedback to the user. All of the methods that are combined to create our approach have been demonstrated by others to be useful for 3D animation. Our solver has already been used to solve generalized inverse kinematics problems in 3D.

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

Sebastian Grassia and Will Welch provided advise on numerical issues. Mark Halstead provided wavelet B-Spline code. Kim Tempest animated the walking, running and jumping motions and designed the character used in Figures 7 and 8. Allen Cypher and John Maloney provided comments on drafts of this paper. This work was supported in the Core Technologies Laboratory of the Advanced Technology Group of Apple Computer. We thank the Addison Gallery at the Phillips Academy for making the Muybridge images available to us.

References

[AGL85] William Armstrong, Mark Green,. The dynamics of articulated rigid bodies for purposes of animation. *The Visual Computer,* pp. 231-240, December 1985.

[Bar91] David Baraff. Coping with Friction for Non-penetrating Rigid Body Simulation. *Computer Graphics,* 25(4). Proceedings of SIGGRAPH 91.

[BB88] Ronen Barzel and Alan Barr. A Modeling System Based on Dynamic Constraints. *Computer Graphics,* 22 (4). Proceedings of SIGGRAPH 88.

[BC89] Armin Bruderlin and Thomas Calvert. Goal-Directed, Dynamic Animation of Human Walking. *Computer Graphics,* 23(3). Proceedings of SIGGRAPH 89.

[BMB86] Norman Badler, Karman Manoochehri and David Baraff. Proceedings of the 1986 Workshop on Interactive 3D Graphics. ACM Press.

[BN88] Lynn Shapiro Brotman and Arun Netravali. Motion Interpolation by Optimal Control. *Computer Graphics,* 22 (4). Proceedings of SIGGRAPH 88.

[BW95] Armin Bruderlin and Lance Williams. Motion Signal Processing. Proceedings of SIGGRAPH 95. In *Computer Graphics* annual conference series.

[Cra86] John Craig. Robotics: Mechanics and Control. Addison-Wesley, 1986.

[Coh92] Michael Cohen. Interactive Spacetime Control for Animation. *Computer Graphics,* 26 (2). Proceedings of SIGGRAPH 92.

[DZ95] Steven Drucker and David Zeltzer. CAMDROID: A System for Implementing Intelligent Camera Control. Proceedings of the 1995 Symposium on Interactive 3D Graphics.

[Fle87] Roger Fletcher. *Practical Methods of Optimization,* John Wiley and Sons, 1987.

[FB93] Barry Fowler and Richard Bartels, Constraint-based curve manipulation, IEEE Computer Graphics and Applications 13 (5), September 1993.

[FW88] Kurt Fleischer and Andrew Witkin. A modeling testbed. *Proceedings of Graphics Interface '88,.*

[Gil81] Philip E. Gil, Walter Murray, and Margaret Wright. *Practical Optimization,* Academic Press, 1981.

[GM85] Michael Girard and A. A. Maciejewski. Computational Modeling for the Computer Animation of Legged Figures. *Computer Graphics,* 19(3). Proceedings of SIGGRAPH 85.

[GW93] Michael Gleicher and Andrew Witkin. Supporting Numerical Computations in Interactive Contexts. Proceedings of Graphics Interface 93.

[Gle94] Michael Gleicher. A Differential Approach to Graphical Interaction. PhD Thesis, School of Computer Science, Carnegie Mellon University, 1994. Also appears as CMU School of

Computer Science technical report CMU-CS-94-217.

[Gle95] Michael Gleicher. Practical Issues in Graphical Constraints. In Vijay Saraswat and Pascal Van Hentenryck, editors, *Principles and Practice of Constraint Programming,* MIT Press, 1995.

[GT95] Radek Grzeszczuk and Demetri Terzopoulos. Automated Muscle-Actuated Locomotion Through Control Abstraction. Proceedings of SIGGRAPH 95. In *Computer Graphics* annual conference series.

[HWBO95] Jessica Hodgins, Wayne Wooten, David Brogan, and James O'Brien. Animating Human Athletics. Proceedings of SIGGRAPH 95. In *Computer Graphics* annual conference series.

[HHK92] William Hsu, John Hughes, and Henry Kaufman. Direct Manipulation of Free-form Deformations. *Computer Graphics* , 26 (2). Proceedings of SIGGRAPH 92.

[IC87] Paul Isaacs and Michael Cohen. Controlling Dynamic Simulation with Kinematic Constraints, Behavior Functions, and Inverse Dynamics. *Computer Graphics,* 21 (4). Proceedings of SIGGRAPH 87.

[Kass92] Michael Kass. CONDOR: constraint-based data flow. *Computer Graphics*, 21(4). Proceedings of SIGGRAPH 92.

[KKKL94] Yoshihito Koga, Koichi Kondo, James Kuffner, and Jean-Claude Latombe. Planning Motions with Intensions. Proceedings of SIGGRAPH 94. In *Computer Graphics* annual conference series.

[Las87] John Lasseter. Principles of Traditional Animation Applied to 3D Computer Animation. *Computer Graphics,* 21 (4). Proceedings of SIGGRAPH 87.

[LGC94] Zicheng Liu, Steven Gortler, and Michael Cohen. Hierarchical Spacetime Control. Proceedings of SIGGRAPH 94. In *Computer Graphics* annual conference series.

[LWZB90] Philip Lee, Susanna Wei, Jianmin Zhao and Norman Badler. Strength Guided Motion. *Computer Graphics,* 24 (f). Proceedings of SIGGRAPH 90.

[Mil88] Gavin Miller. The Motion Dynamics of Snakes and Worms. *Computer Graphics,* 22 (4). Proceedings of SIGGRAPH 88.

[Nak91] Yoshiko Nakamura. *Advanced Robotics: Redundancy and Optimization.* Addison Wesley, 1991.

[NM93] J. Thomas Ngo and Joe Marks. Spacetime Constraints Revisited. *Computer Graphics.* Proceedings of SIGGRAPH 93.

[Pau81] Richard Paul. Robot Manipulators: Mathematics, Programming and Control. MIT Press, 1981.

[Per95] Ken Perlin. Real Time Responsive Animation with Personality. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, No.1, March 1995.

[PTVF92] William Press, Saul Teukolsky, William Vetterling, and Brian Flannery. *Numerical Recipes in C, second edition*. Cambridge University Press, Cambridge, England, 1992.

[Pla92] John Platt. A generalization of dynamic constraints. *CGVIP: Graphical Models and Image Processing*, 54(6), pp. 516-525, November 1992.

[Ree83] Bill Reeves. Particle Systems -- a Technique for Modeling a Class of Fuzzy Objects. *ACM Transactions on Computer Graphics*, pp. 91-108, April 1983.

[Rey87] Craig Reynolds. Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics,* 21 (4). Proceedings of SIGGRAPH 87.

[Sims94] Karl Sims. Evolving Virtual Creatures. *Computer Graphics.* Proceedings of SIGGRAPH 1994.

[Suth63] Ivan Sutherland. "Sketchpad: A Man-Machine Graphical Communication System," MIT Lincoln Laboratory Technical Report, Number 296, January 1963.

[VF93] Michiel van de Panne and Eugene Fiume. Sensor-Actuator Networks. *Computer Graphics.* Proceedings of SIGGRAPH 93.

[Wam86] Charles Wampler. Manipulator-based inverse kinematic solutions based on vector formulations and damped least squares. *IEEE Transactions on Systems, Man, and*

Cybernetics, 16 (1) January 1986.

[Wil87] Jane Wilhelms. Using dynamic analysis for realistic animation of articulared bodies. *IEEE Computer Graphics and Applications,* June 1987.

[WFB87] Andrew Witkin, Kurt Fleischer and Alan Barr. Energy Constraints on Parameterized Models. *Computer Graphics,* 21 (4). Proceedings of SIGGRAPH 87.

[WK88] Andrew Witkin and Michael Kass. Spacetime Constraints. *Computer Graphics,* 22 (4). Proceedings of SIGGRAPH 88.

[WP95] Andrew Witkin and Zoran Popovic. Motion Warping. Proceedings of SIGGRAPH 95. In *Computer Graphics* annual conference series.

[WW92] William Welch and Andrew Witkin. Variational Surface Modelling. *Computer Graphics,* 26 (2). Proceedings of SIGGRAPH 92.

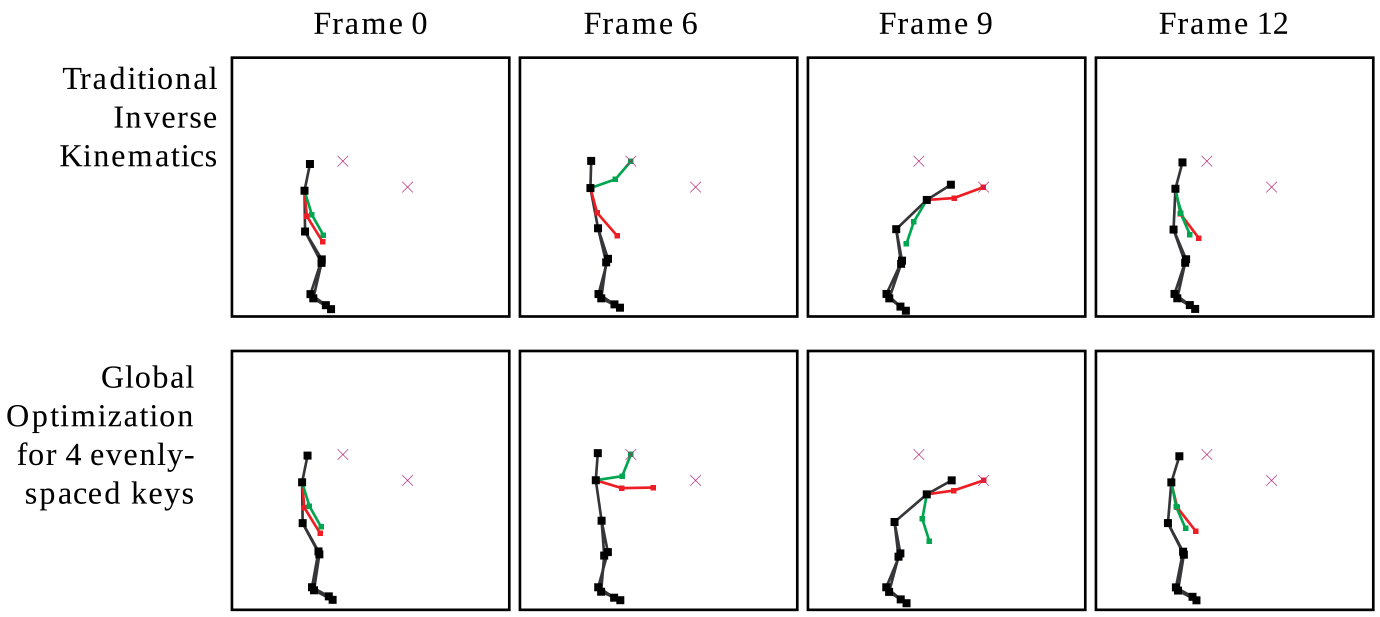

Frame 0
Frame 6
Frame 9
Frame 12
Traditional Inverse Kinematics
Global Optimization for 4 evenly-spaced keys

Figure 2: 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 a16 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)

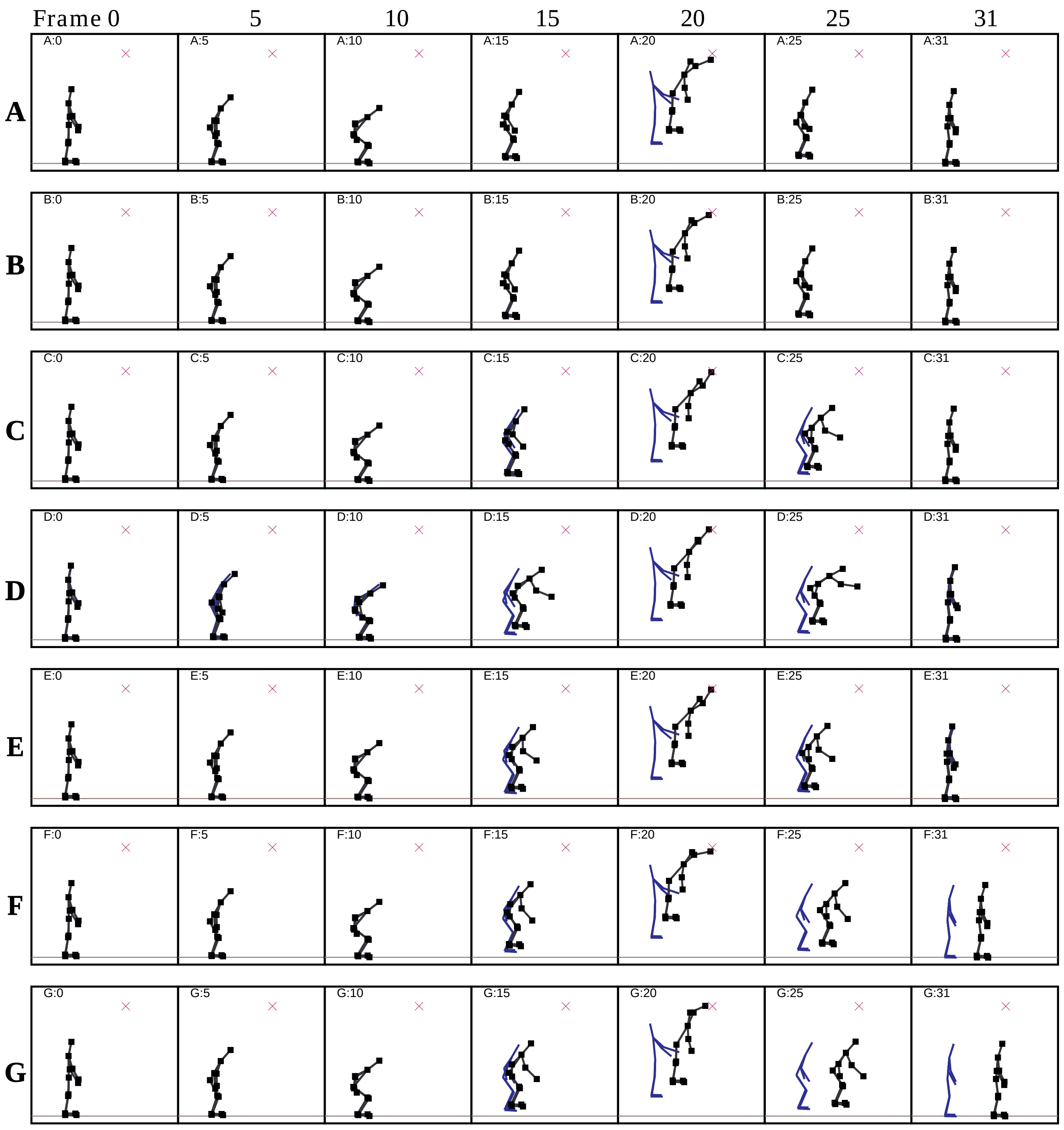


Figure 3: Adapting a jumping motion.

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 of the dense samples.

b) displacement mapping with constraint at time 20 and zeros at times 15 and 25.

c) same as b, but with zeros at beginning and end of flight, at times 13 and 29.

d) our method with 4 evenly spaced keys for the displacement curve. All keys may move.

e) our method with 4 evenly spaced keys for the flight, the first and last are held to be zero-valued.

f) x-displacement has two keys, one at the beginning and end of the flight, but only the last key is allowed to move. Displacement curves for other parameters are as in (e).

g) same as (f), but optimization function is modified allow longer flight.

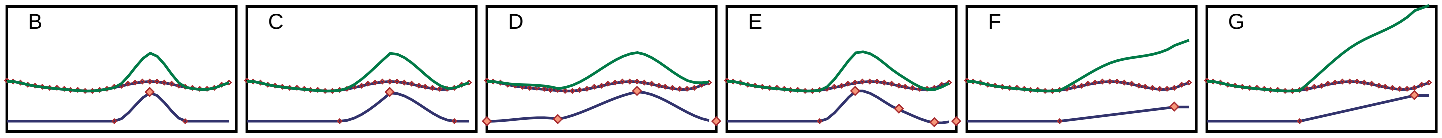


Figure 4: Timeline curves for the jumping motion.

These curves graph the pelvis-x-position parameter of the jumping motions shown in Figure 3 Sequences B through G. In each, the blue curve represents the resulting motion, the red curve the initial motion, and the displacement is shown in green. The keys of the displacement curves are shown with diamonds: green represents keys that are permitted to change, while red keys are locked at their zero values.

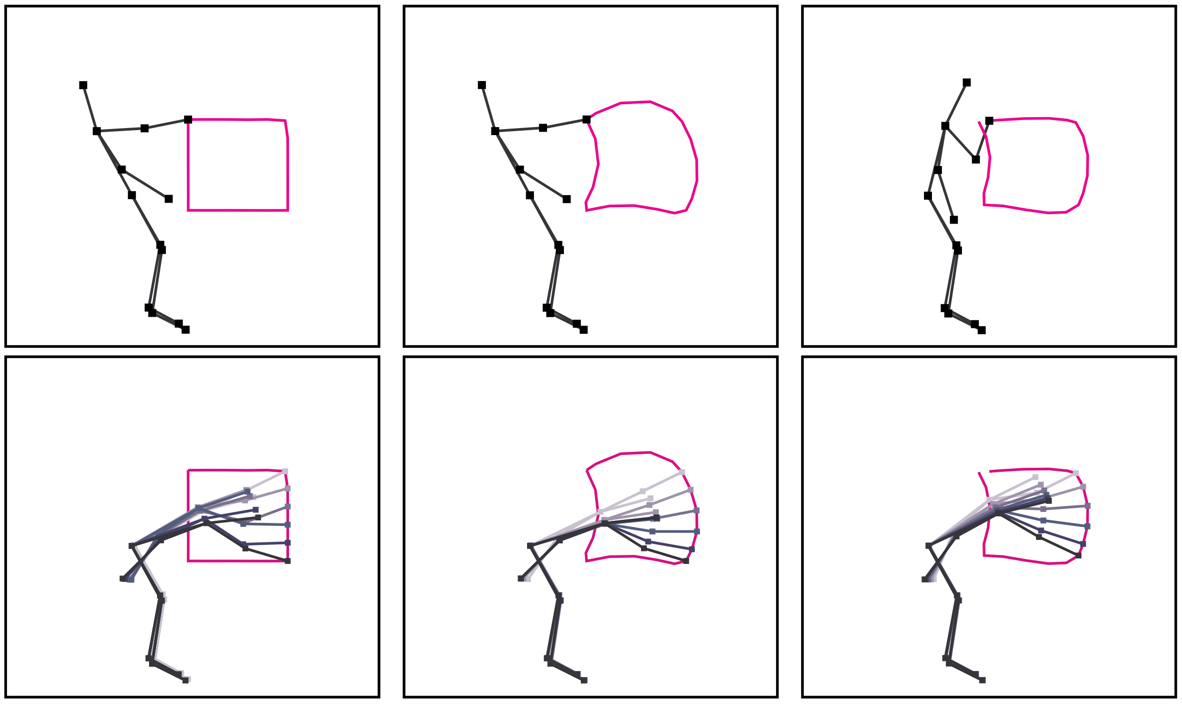


Figure 5: Tracing a square.

An articulated figure's hand is tied to a moving point that traces a square point. The magenta line shows the path actually traced by the character's hand, and the characters position is shown 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.

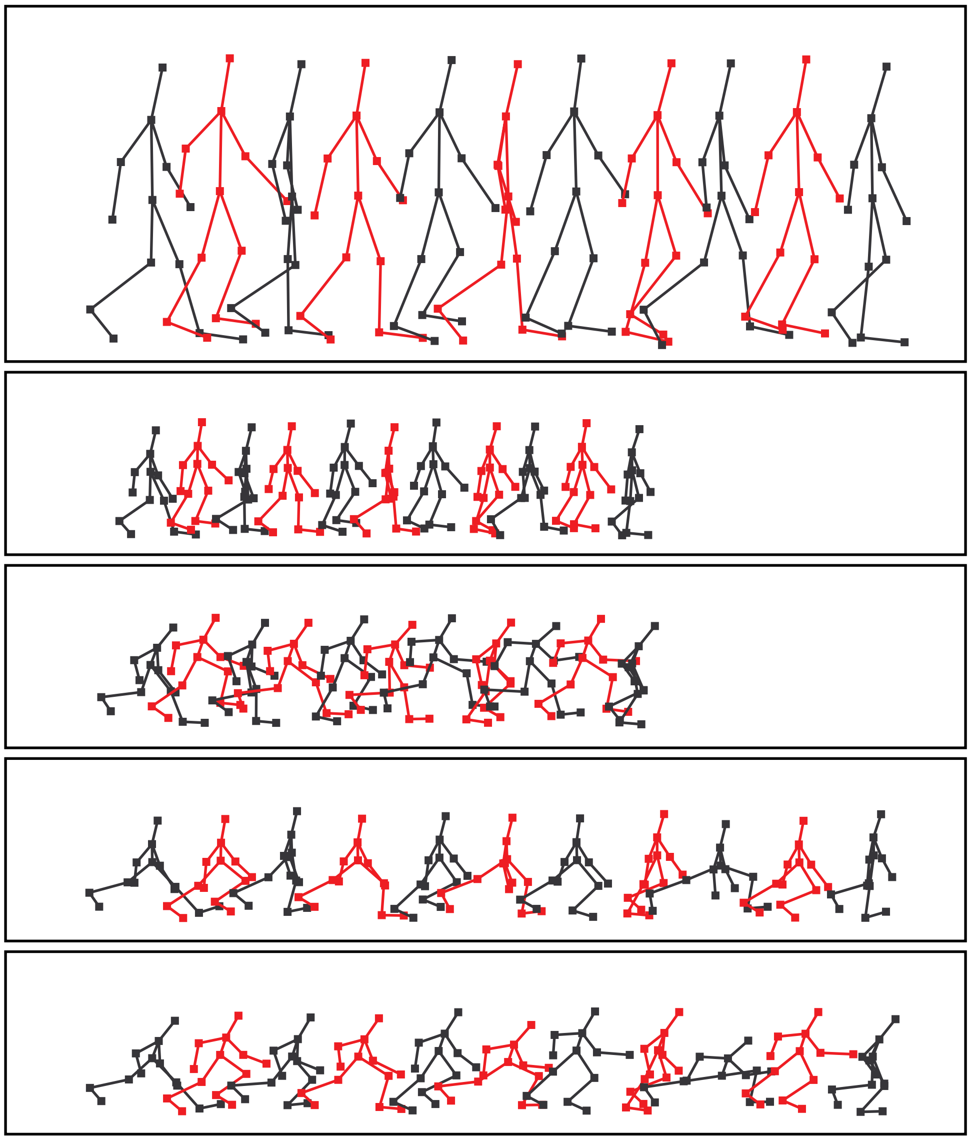


Figure 6:Tracing footpaths

The walking motion of the tall character (a) is used to adapt the motion of the smaller character (b and c). Figure (d) shows the result of adapting the initial walking motion (b) to have the same footfalls of the taller figure's walk (a). Figure (e) shows the same process applied to the running motion (e). Notice how the two adapted motions retain much of the character of their original motions, despite being heavily altered to meet the constraints.

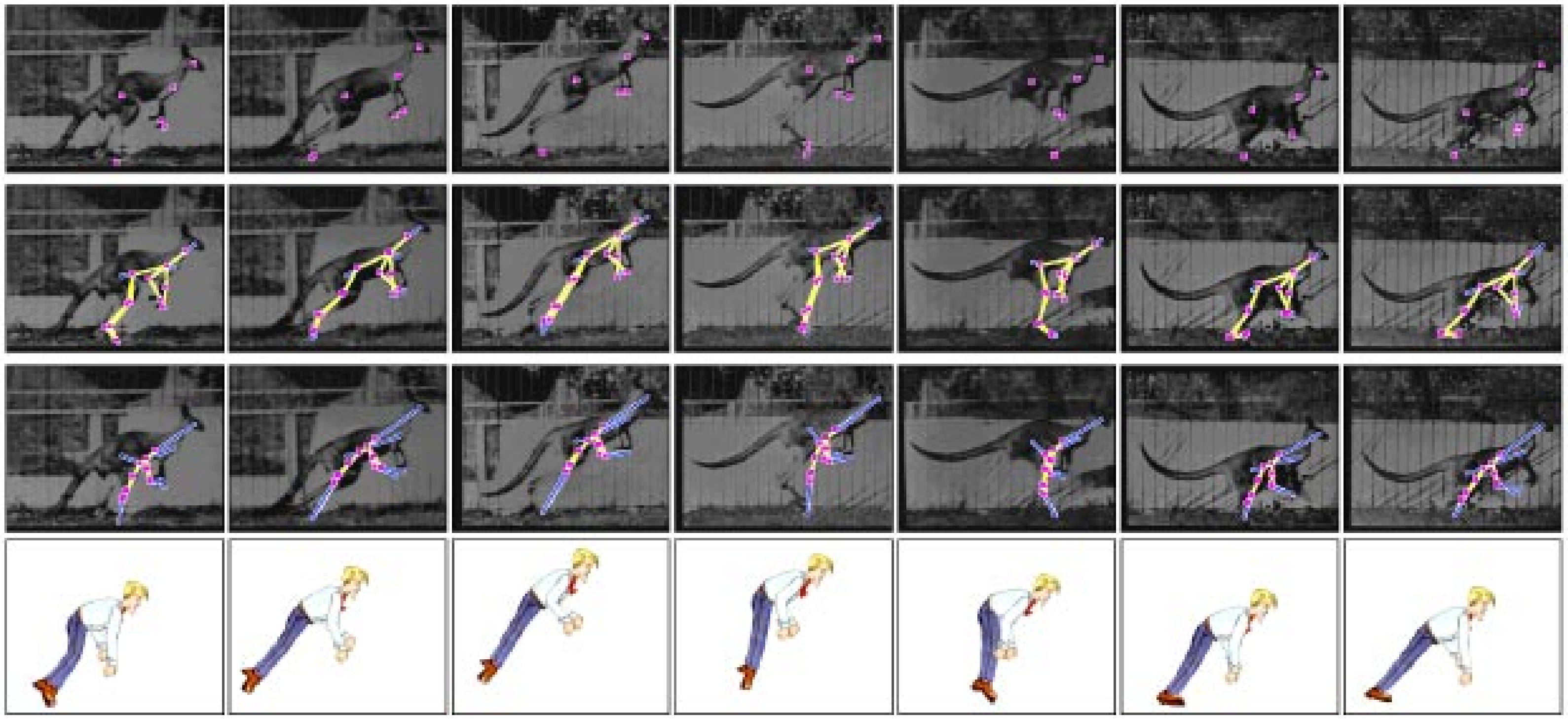


Figure 7: Mapping the motion of a kangaroo

We manual trace the locations of key points on the kangaroo (a).
The end effectors of the human figure (shown in yellow) are connected to the corresponding features of the kangaroo (e.g. we connect each of the person’s hands to the kangaroo’s paws).
Figure (b) shows the results of solving these constraints for a human figure approximately as tall as the kangaroo.
Figure (c) shows the results with a much shorter human figure. In this case, the figure is unable to meet theconstraints so the least-squares-error solution’s deviation (shown in blue) are much more pronounced.
Figure (d) shows the motion mapped onto a cartoon drawing.

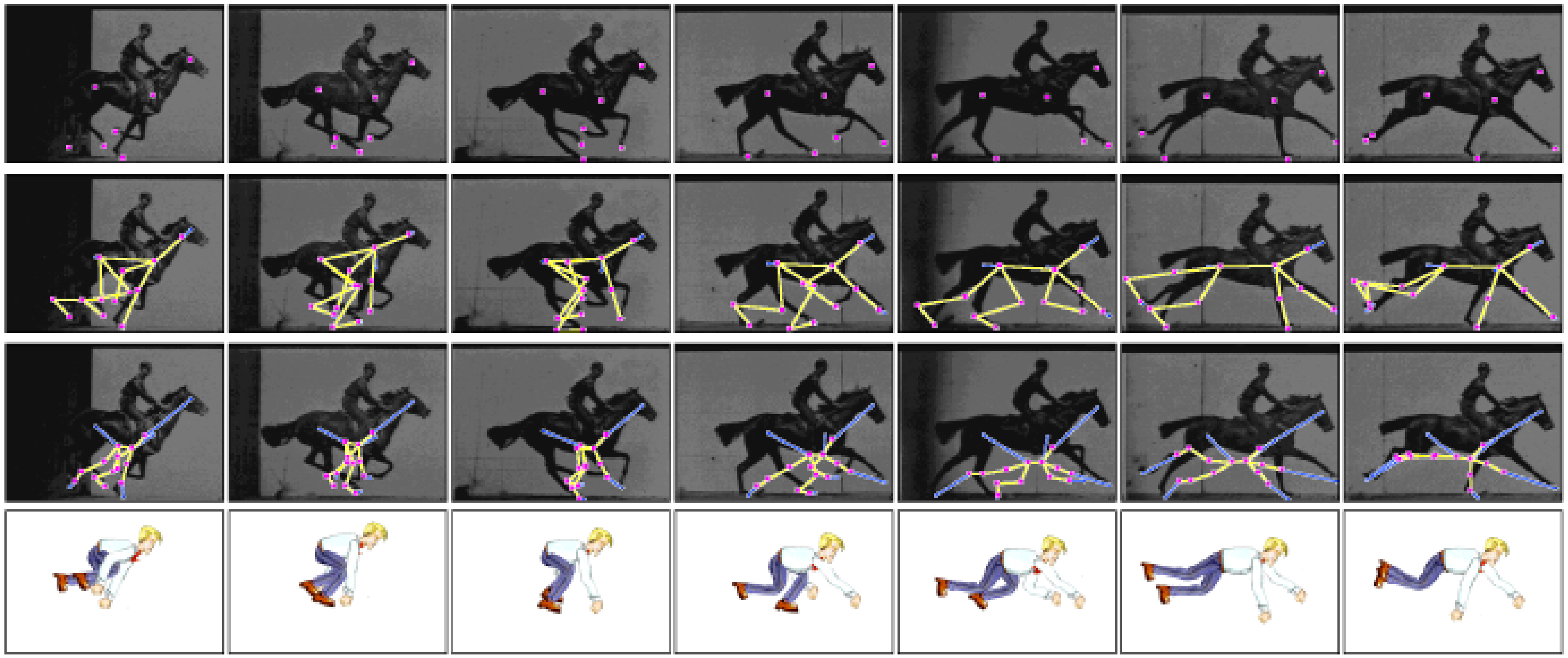


Figure 8: Making a person crawl like a horse gallops

a) Horse motion traced by hand.

b) Tall human figure adapted to follow the horse motion. Blue lines show constraints that are not completely satisfied.

c) Same as (b), but with a shorter human figure.

d) motion mapped to a cartoon character