Comparative Analysis of Constraint-Based Motion Editing Methods

Michael Gleicher
Department of Computer Sciences
University of Wisconsin, Madison
`http://www.cs.wisc.edu/graphics`

January, 2001

This is a corrected version of a paper that appeared at the 2000 Workshop on Human Modeling and Analysis, Seoul, Korea.

Abstract

Tools for assisting with editing human motion have become one of the most active research areas in the field of computer animation. Not surprisingly, the area has demonstrated some stunning successes in both research and practice. This paper explores the range of constraint-based techniques used to alter motions while preserving specific spatial features. We examine a variety of methods, defining a taxonomy of these methods that is categorized by the mechanism employed to enforce temporal constraints. We pay particular attention to a less explored category of approaches that we term *per-frame inverse kinematics plus filtering*, and show how these methods may provide an easier to implement system that has many of the benefits of other approaches.

1 Introduction

The proliferation of different techniques for creating motion for computer animation, coupled with the needs to best utilize these artistic assets, has made motion editing a rapidly developing area of research and development. The past few years have witnessed an explosion of new techniques for various motion editing tasks, as well as the deployment of commercial quality tools. Unfortunately, while each new technique almost always provides impressive new results, there has been little done to examine the differences and capabilities of various approaches. In this paper, we explore the relationships between a number of state-of-the-art motion editing methods. Specifically, we provide a unifying view of several constraint-based techniques. The methods provide explicit support for maintaining geometric features of motions. Our goal is to examine the range of techniques for insights. Comparing and contrasting the methods to understand the problems for which each is appropriate, and to give us insights as to unexplored regions of the design space.

All papers about motion editing introduce new techniques. Therefore, each paper is certain to focus on where the methods work best, and explaining why their approaches are superior to the previous work. In this paper, we free ourselves from the demands of selling a novel method.

Our discussion is organized as follows. We begin by defining our terms and laying a groundwork to introduce motion editing methods. We then present a taxonomy of recent approaches, describing each within a common set of terminology to aid in comparison. Section 4 focuses on a particularly unexplored category of approach, showing how a simpler variant of a previous technique can provide useful results.

2 Constraint-Based Motion Editing

Before comparing motion editing techniques, we must first define what it is we mean by motion editing. Motion editing is the act of changing the movement of an object[Gle99]. Outside of computer animation, we generally do not get to control the movement of things without controlling the things themselves. Even in more traditional 2D animation, the movement and appearance of objects are tightly coupled. The idea of discussing motion editing only really becomes a worthwhile topic when we deal with 3D animation.

This definition of motion editing is sufficiently broad that most techniques in computer animation cab be applied to the problem. We focus our attention on tools specifically designed for this task, that is, tools that are specifically designed for changing existing motions. This broad characterization points to an important feature of the techniques: since we are changing an existing motion, there must be a notion of preserving the original motion as well as adding new features to the motion. If we did not want to preserve some aspects of the original, we would not have needed it to begin with.

For the purpose of this paper, we focus even more narrowly on the character body motion for figure animation. Typically, such animation represents characters as articulated figures: hierarchical representations connecting a tree of rigid links. While most of the methods we discuss in this paper are not limited to such tasks, all of them address the primary difficulty of using such representations: the parameters of the character (joint angles and root position) are related to many of the most important geometric features (the position of end effectors, for example) by complex, non-linear relationships.

Constraint-based motion editing makes some of the features that are to be preserved or changed in a motion explicit as constraints. This is in contrast to a non-constraint-based technique which does not make such features explicit. For example, a filtering or blending operation applied to a motion does change it (and is therefore an editing operation), but does not explicitly describe the operation in terms of features in the motion, only in the terms of the underlying parameters used to represent the motion. A constraint-based technique would explicitly represent features in the motion. For example, the footsteps in a motion might be represented as an equation specifying the required position of the end-effector at a given time. Such features serve as constraints either to be preserved (maintain the footplants as the motion is changed) or to control changes (move a footplant to a new location).

Constraint-based motion editing techniques generally focus on spatial or geometric constraints. Spatial constraints provide features of poses at given instants. For example, they might specify that a hand must be in a given place at a given time, that the elbow does not bend backwards at any time during the motion, or that the character's gaze direction faces an event of interest when the event occurs. A distinguishing feature of constraint-based editing techniques is that they allow for spatial constraints independent of the parameters of the characters, positioning the end-effectors of an articulated figure, for example. This is in contrast to things such as pure signal processing approaches, as introduced by Litwinowicz [Lit91], Unuma et al [UAT95], Bruderlin and Williams [BW95], Popović and Witkin [WP95], and others, where the parameters are considered independently of their interrelationships.

In addition to the spatial properties of motions, editing techniques must consider temporal properties as well. There has been less formalism of temporal constraints on motion. While some authors (for example [UAT95], [Gle98], [LS00]) have used mathematically convenient terminology like frequency, these are typically crudes approximation to the properties that we might really desire to control and maintain in motions. One of the most effective temporal constraints presented to date [WP95] [Gle98] follows from the observation that high frequencies in motion are almost always noticeable and usually caused by significant external events. Therefore, the high-frequency content must be preserved during editing. That is, high frequencies should only be added or removed with care: it is just as incorrect to dull the "snap" of a karate kick or a chef's chop as it is to add a twitch to a graceful ballet movement.

2.1 Types of Editing Operations

To date, constraint-based motion editing operations have been primarily applied to motion transformation tasks. These are problems where good motions are adapted to apply to different settings, such as new characters or new environments.

Constraint-based motion editing methods can be applied beyond motion transformations. For example, a signal processing operation such as filtering may destroy a motion's spatial constraints. A constraint-based method could then be used to re-establish these constraints. Similarly, interactive manipulation of a frame of the motion might destroy the temporal constraints, which could be re-established with a constraint-based solver.

The editing paradigm where some operation is used to alter a motion and then a constraint-based solver is used to "clean up the mess" is only one of the possible approaches to constraint-based motion editing. A different paradigm begins with a motion that satisfies the constraints and then alters the constraints while maintaining their validity. An example of the former approach is our work on retargeting [Gle98], while examples of the latter include our interactive editing work [Gle97], Lee and Shin's hierarchical editor [LS99], and Popović's physically based approach [PW99].

2.2 Inverse Kinematics

The methods used to find the solutions to the geometric constraints are a key implementation choice in constraint-based motion editing systems. The handling of geometric constraints in animation is typically called inverse kinematics.

Because the term "inverse kinematics" has different interpretations, we define ours. Inverse kinematics (IK) is a process for determining the configuration of a character's parameters (known as its pose) based on specifications of resulting features of the pose, such as end-effector positions. Note that this is a generalization of another common interpretation of the term (popular in robotics texts) that defines IK as the process of determining joint angles from an end-effector position. For our definition, we neither restrict the types of parameters (i.e. all of a character's parameters, not just its joint angles are considered), nor the number or type of constraints (i.e. not just end-effector positioning). Inverse kinematics implies solving the geometric constraints at a specific instant in time.

All constraint-based motion editing methods must include a solver for IK problems in one form or another to allow for constraints on end-effector positions. The nature of inverse kinematics leads to three general types of challenges that any solver must address:

1. The relationships between the the end-effector being controlled and the pa-

rameters being determined will be non-linear, often quite complex. This means that solvers often must rely on sophisticated methods.

2. There exist goals for which there are no solutions. Such singular problems are particularly difficult because there are often goals close to these impossible situations that are barely achievable. The solvers need to distinguish the impossible from the merely difficult makes the problem of IK ill-conditioned and is shown by Maciejewski [Mac90] to be inherent in the mechanics of the problem.

3. There is almost always a multitude of possible solutions: most end-effector goals may be reached by a variety of poses. The solver must have a mechanism for choosing among the space of possible solutions.

2.2.1 IK Solvers

Any IK solver must address these three issues. The first two are critical to creating a robust and efficient method, essential to any application. The third criterion provides some freedom in solution methods. In choosing among correct solutions, solvers can differentiate from one another by choosing "better" answers in terms of the quality of the resulting pose, or the predictability of the results.

The what that an IK solver chooses amongst possible solutions to goals is an important issue in the solver's design. Some solver implementations make this choice explicit by casting the IK problem as a constrained optimization problem. For example, a solver might prefer the solution closest to its initial starting point, the one closest to the previous frame, or one that provides the "best" pose subject to posture, strength, or balance objectives. However, the choice is often implicit in the design of the solver since many solvers take a more ad-hoc strategy. Effectively, many solvers provide "the answer that they happen to choose." Another way to look at these algorithms is that they are optimizing something else: simplicity of code, or the speed of finding a solution.

While IK methods were developed in mechanics and robotics before their application in computer graphics, the demands of graphics applications (particularly in character animation) brought new challenges for the methods to address. The methods for IK solvers applied in graphics fall into two categories: geometric or analytic solutions, and numeric or iterative solutions.

Analytic solution methods use closed form geometric constructions to compute configurations for end-effector positions directly. Such methods must be carefully constructed to handle specific cases. Tolani, Goswani and Badler [TGB00] recently provided a method for a 7 degree of freedom limb as typically used in computer animation. This work extends earlier solvers, such as those by Korein and

Badler [KB82]. Analytic methods have the advantage that they provide guaranteed fast solutions. However, they lack flexibility in how they choose solutions in underconstrained cases and in the types of problems they can handle.

Numeric solution methods use equation solving or optimization techniques to provide a more general, albeit computationally expensive, IK solution. Because of the non-linear nature of the inverse kinematics equations, a numerical solver will necessarily be an iterative method. The methods employed fall into two categories: unconstrained and constrained. Unconstrained methods model IK goals with objective functions that are minimized by an unconstrained optimization routine. Such methods, typified by Badler et al. [BMW87], Welman [Wel93], and Phillips and Badler [PZB90, PB91], can employ simpler solvers and may trade accuracy of end-effector positions for other goals, like balance. Constrained methods treat end-effector goals and joint limits as hard constraints, and reserve the optimization objective to select among solutions. Realizations of this strategy, such as Zhao and Badler [ZB94], require the use of sophisticated non-linear programming algorithms.

Lee and Shin [LS99] present a solver that combines analytic and numerical techniques. They employ analytic solutions for the limbs, where closed form solutions are available, and reserve optimization for the computation of body posture.

3 A Taxonomy of Solving Techniques

The solvers used in constraint-based motion editing all address similar tasks. The main differences are the approaches they employ to achieve their results. In this section, we look at the various approaches used to maintain or establish the spatial and temporal constraints in editing techniques. One might hope that what occurs "under the hood" of an editing technique would be of little concern to the user. Our goal in this exploration is to see how the choices in solver strategy affect what the methods that use them can achieve.

By our definition in the last section, a constraint-based motion editing technique must provide some method for handling spatial constraints, that is mapping specified goals to a character's parameters. Therefore, all methods must include an inverse kinematics solver (in a general sense).

Our taxonomy categorizes methods based on their approach to handling temporal constraints. The existing methods vary greatly, from approaches that do not explicitly handle temporal constraints to those that provide explicit equational constraints for temporal properties.

3.1 Per-Key Methods

Traditional computer animation systems provide the user with the ability to control a set of "key" poses that are interpolated. Spatial aspects of the motion are created by controlling these key poses, and the temporal continuity between poses is generated by interpolation. We term such methods *per-key inverse kinematics* (PKIK) because they solve a problem to handle the spatial constraints on each key in the motion. Traditional animation tools fall under the category of per-key methods.

An important aspect of a per-key motion editing technique is that the "solver" changes each pose individually. The choice of how to set each pose is computed independently. These computations may consider other frames in the process, however each pose is computed separately. This attribute is shared with the per-frame methods described below. The reason that such methods often work so well is that the small number of key frames is often sufficient to describe the motion. If significant key frames are properly chosen, interpolating between them can provide sufficient temporal continuity and control.

A central difficulty with per-key methods for motion editing is that they require a set of sparse, well-chosen key frames to represent the motion. The temporal control of motion changes is an artifact of how the motion was created and represented, not necessarily determined by the goals of the changes in the motion. This is particularly problematic for motions that are generated algorithmically, with motion capture or simulation, for example. However, even manually constructed motion does not place key frames solely at semantically relevant instants. One possible approach to this is to use some process for constructing a convenient representation from the unstructured data.

3.2 Motion Warping or Displacement Mapping

One drawback of the per-key approach is that key frames may not be placed at spacings convenient for editing operations. Motion warping [WP95], also known as motion displacement mapping [BW95], provides a solution to this problem. In such an approach, the *changes* to a motion are keyframed. This allows keys to be placed conveniently for editing. We term a motion editing approach based on motion warps as *motion warping plus inverse kinematics* (MW+IK) because it must augment motion warps with an inverse kinematics solver to handle the spatial constraints.

When combined with an IK solver, Motion Warping provides a type of constraint-based motion editing method. The IK solver is applied to frames with specified geometric constraints, and the spacing of the keys of the displacement map can be used to enforce temporal constraints on the changes applied to the motion.

The MW+IK approach does have some drawbacks. First, there is no control over geometric constraints except at the keyframes. Secondly, the amount of temporal constraint possible depends on the amount of geometric control. To provide more geometric control, more keys must be provided, leading to a reduction in the amount of temporal constraint.

PKIK can actually be viewed as a special case of MW+IK where the keys in the displacement map are placed at the same times as the keys in the original motion.

3.3 Per-Frame Methods

A per-frame method manipulates each individual sample of a motion independently. We prefer the term per-frame rather than per-sample because it sounds better, although the latter is probably more technically correct, especially if we are dealing with motions that are algorithmically generated or captured at a high sampling rate. We refer to the class of methods that applies a solver to each frame of the motion independently as *per-frame inverse kinematics* (PFIK).

In a sense, a per-frame method is a special case of a per-key method where the keys are regularly spaced. However, this is an important distinction: with a per-key method, we assume that the keys are strategically placed at significant and important times and, therefore, it is more likely that simply getting these instants correct will provide for a desirable outcome.

With per-frame methods, the poses that we consider are typically densely sampled. That is, the timing between poses is typically quite small relative to the duration of the motion. Because of this, there is a greater demand to enforce the consistency between poses. When the poses are spaced farther the interpolation between poses becomes more significant and will create the necessary consistency.

To emphasize the difference between the per-frame and per-key approach, consider a motion that is the trajectory of a foot making a step. If such a motion was created manually, the artist might create key poses at the beginning, middle and end of the step and interpolate these sparse keys. This same motion could be sampled at full frame rate. If we were to change the height of the apex of the step, a quite different result would be achieved. With the sparse keys, we would obtain a higher step, while with the dense samples, we would simply put a spike in the motion at the apex.

The contrived nature of the simple example might make per-key methods sound more likely to give desirable results. However:

1. Per-key methods offer only the *opportunity* for choosing keys that are semantically relevant and, therefore, convenient controls over the animation.

2. Per-key methods generally decouple the editing operations from the frame

rate. The timing of the keys usually corresponds to events in the motion, while the timing of the frames is usually an artifact of how the motion will be displayed.

The characteristic of a per frame method is that the decision of how to adjust each frame is made independently. Each of these independent decisions may take into account more information than just the pose at the current frame. For example, a solver might refer to the previous frame in order to choose a solution for the current one that does not introduce a discontinuity. An example of this approach is the On-line Motion Retargeting work of Choi and Ko [CK99].

3.4 Per-Frame Plus Filtering

Once the decisions of how to alter each frame are made, some approaches then provide a "global" process that considers multiple instants together to remove artifacts of the independent nature of the first step. Almost always[1], this step is a low-pass filter meant to remove the spikes and other discontinuities introduced by the independence of the first steps. The first published motion editing system based on such an approach was the Hierarchical Motion Editing technique of Lee and Shin [LS99] that used B-Spline fitting to implement the low-pass filtering. We term the general approach *per-frame inverse kinematics plus filtering* (PFIK+F) and discuss the approach in detail in Section 4.

The PFIK+F approaches are distinguished by the existence of two separate phases. Generally, there is a phase that considers spatial constraints, applying inverse kinematics to make coordinated changes to all joints of a character simultaneously, followed by a signal processing stage that implements the temporal constraints. The spatial phase considers the entire character simultaneously, albeit at a single instant in time, while the temporal phase considers the entire motion (or a portion-thereof) simultaneously, albeit a parameter at a time. Because these two phases cannot consider the work of the others, one may "undo" the work done by the other, so they are often interleaved in an iterative process.

Treating each signal independently is technically incorrect. At a local level, groups of parameters are almost always coupled to represent rotations. For example, independently filtering each parameter of an Euler angle does not necessarily have meaning for the rotations themselves. In practice, we believe that this approach works on the displacements because the displacements are typically small, and in small-angle approximation, the orientation representations are linear. Recently, Lee and Shin [LS00] have shown that using low-pass filters on exponential

[1]All of the examples to date.

coordinates independently does actually have meaning, giving a theoretical foundation to the PFIK+F style of approaches.

3.5 Spacetime Methods

Spacetime constraint-based motion editing techniques distinguish themselves from the previously described methods in that they do not consider frames individually. Spacetime Constraints refer to methods that consider a duration of motion simultaneously in a computation. Rather than computing an individual frame, as an IK solver does, the solver computes an entire motion, or any sub-window of it. This allows it to consider constraints on the entire duration of the motion, and to have objective criteria that consider entire motions.

The initial use of spacetime constraints specified desired positions for a character and used the solver to compute the "best" motion that met these positions. These initial works, presented by Witkin and Kass [WK88] and Cohen [Coh92], included constraints that enforced the laws of physics and created an objective function that defined the "best" solution as one that minimized the amount of energy the character expends with its muscles. This synthesized novel motions that had a simple character perform simple motions that were physically correct.

The power of the spacetime constraints approach is also its drawback. While the approach provides tremendous opportunity to define constraints and objective functions that describe features of the resulting motions, these must be defined for the approach to work. Defining such mathematical characterizations for motion properties is challenging. While the approach offers the potential for high level properties to be employed as criteria, to date, concepts such as "graceful" or "angry" or "like–Fred–Astaire" have eluded a mathematical description that fits into the framework. Also, the approach requires solving a single mathematical problem for the entire motion. This leads to very large constrained optimization problems that are usually very difficult to solve.

We initially proposed applying the spacetime constraints approach to a motion transformation problem in [GL98]. Conceptually, the main difference with the standard spacetime work was that our objective sought resulting motions similar to the initial motions, rather than seeking results that minimized energy consumption. This allowed us to avoid the difficult problem of specifying motion details: we did not have to figure out how to describe a walk with constraints since we could define a walk by example.

The important properties to preserve in a given motion may not always be simple: "realism," "grace," "like-in-Singing-in-the-Rain," or other high-level properties may be desirable to preserve during adaptation. In practice, we are limited by our ability to define high-level qualities of the motion mathematically, by our ability to compute adaptations efficiently when the metrics become complex, and by the amount of effort we wish to expend in identifying (or having the user identify) these properties. Even if we encoded the desired animation completely in a constrained optimization, we would still need to solve these problems. Generally, richer sets of constraints and objective functions lead to more difficult problems to solve.

To date, the spacetime approach has been demonstrated on a variety of motion transformation tasks. We have demonstrated that the approach can provide real-time interactive performance on modest computers in real time [Gle97].

3.6 Physical Approaches

Physics provide a specific, and useful, source of constraints. While certain physical properties, such as contact with the floor, are conveniently represented as geometric constraints in the spacetime systems of the previous section, other physical properties are ignored for the sake of performance. Most significantly, Newton's laws are not included. The choice to exclude these constraints was a specific concession of the work of [Gle97]. Kinetic constraints and energy properties are particularly computationally expensive in the spacetime framework[2].

For some motions, the kinetic behavior is critical. In such situations, it is unacceptable to make the simplification of discarding Newton's laws as a constraint. Popović and Witkin [PW99] make a different set of tradeoffs to make the spacetime approach realizable. Rather than simplifying the physics by ignoring Newton's laws, they chose to simplify the geometry of the character instead.

To date, the physical approach of Popović has not been extensively demonstrated. It has only been tested on a small number of short motions. To create a walking motion, the method is applied to a single gait cycle and looped. In addition to its complexityt, the technique does have the disadvantage of being newer, and therefore less evolved, than other approaches. Several researchers are exploring ways to approximate momentum effects within a spacetime editing framework.

4 Per-Frame IK plus Filtering

In this section, we describe our experimental PFIK+F solver. In many ways, this solver is simply a re-implementation of the ideas presented by Lee and Shin [LS99] in their paper. While the concepts are the same, the details of the implementation are quite different. We believe this shows the power of their approach.

[2]Intuitively, such constraints are challenging because they create couplings among frames.

A PFIK+F solver uses a separate inverse kinematics solver applied to each frame of the motion in order to handle spatial constraints. The results of these solutions are then processed to enforce the temporal constraints. In the inevitable event that the second phase destroys the work of the first, the process is repeated.

Lee and Shin apply the temporal constraint suggested by Gleicher [Gle98]: that the *changes* to the motion should be band-limited. Alternatively, one might view their process as saying the spatial constraints should be met using as low a frequency change to the motion as possible. The advantage of this latter view is that it suggests that we are willing to do whatever is required to meet the spatial constraints, even if it means adding high frequencies.

The original paper by Lee and Shin describes a specific implementation of the PFIK+F approach that we generalize in this discussion. Their system makes specific choices for each component of the approach. With each step, they provided an innovation:

1. They use exponential maps as a representation for rotations and provide methodology for using this representation with displacement maps.

2. They provide a novel, simplified IK solver.

3. They use B-spline fitting to the displacement maps to perform the filtering.

The biggest innovation of the paper, is that it introduces the PFIK+F approach. The paper provides the first demonstration of a PFIK+F approach and illustrates its applicability to a wide range of motion editing problems.

Lee and Shin's work (including some of their subsequent papers such as [LS00]) represents only a single point in the design space of PFIK+F systems. Each of their innovations represents a tradeoff:

1. Exponential maps provide a methodology for implementing displacement mapping on Quaternions and give some theoretical meaning to the filtering operations (as presented in [LS00]). However, this representation has other implementation issues (see [Gra98]), does not necessarily provide "nice" motions when splined, and is difficult to integrate with existing methodols.

2. Their simplified IK solver has the advantage that it is extremely fast. This is important since each application of the PFIK+F process may require hundreds of individual calls to the IK solver. The tradeoff of this fast solver is that it supports only a limited set of constraints and sometimes seems to provide unnatural looking motions.

3. B-Splines implement low-pass filters with very wide support and can be easily adapted to weight different samples differently. However, they are

more difficult to implement (and potentially computationally expensive) than convolution-based (FIR) linear filters.

4.1 Our PFIK+F Solver

We have been exploring a different point in the design space of PFIK+F systems. For each of Lee and Shin's choices, we have made a very different decision:

1. We treat whatever underlying rotational representation used by the system as independent coordinates. In practice, this means that we do per-component arithmetic operations on Euler Angles.

   Treating each signal independently is technically incorrect. At a local level, groups of parameters are almost always coupled to represent rotations. For example, independently filtering each parameter of an Euler Angle does not necessarily have meaning for the rotations themselves. Recently, Lee and Shin [LS00] have shown that using low-pass filters on exponential coordinates does actually have meaning, giving a theoretical foundation to the PF+F style of approaches.

   Applying the PFIK+F solver directly to Euler Angles does appear to work in practice, despite its theoretical incorrectness. Rather than trying to make an argument to justify this theoretically, we prefer to acknowledge that such a "signal processing" approach, performing per-component arithmetic on Euler Angles, to motion editing is widely used. In practice, we believe that this approach works on the displacements because the displacements are typically small, and in small-angle approximation, the orientation representations are linear. The approach fails most spectacularly when the Euler Angles wrap around at 2π, as the simple arithmetic cannot notice that two very different parameter values represent the same angle.

   The advantage of applying the PFIK+F solver directly to Euler Angles is easy integration with existing systems. We feel this is an important asset of the PFIK+F approach, even if it comes at the expense of mathematical cleanliness.

2. We use our existing IK solver to solve the IK subproblems. Or, more specifically, we use our general purpose spacetime solver (as described in [Gle98]) on a single frame at a time. While this solver provides a wide range of constraints, this generality comes at the expense of speed.

   Again, the key advantage of this choice is to demonstrate integration with existing systems. Implementing a good IK solver is a challenging process, and such a solver is generally considered a valuable asset in an animation environment. Arguably, implementing an IK solver is the key challenge of creating a PFIK+F system. However, since an existing solver can be leveraged, this difficulty can be avoided.

3. We use linear FIR filters, rather than B-Spline fitting. This has advantages in generality and ease of implementation.

Each iteration of the solver produces a new motion $(\mathbf{m_{i+1}})$ from the results of the previous iteration. We denote the initial motion as $\mathbf{m_0}$.

1. The inverse kinematics solver is applied to each frame of the animation. To compute $\mathbf{m_{i+1}}(t)$, we apply the solver beginning with configuration $\mathbf{m_i}(t)$ and the constraints at time t. The solver is run on each frame of the animation independently.

2. The displacement map is computed as the difference of the new motion and the original motion, $\mathbf{d} = \mathbf{m_{i+1}} - \mathbf{m_i}$.

3. The displacement map is filtered by convolving it with a low-pass filter kernel. Each channel of the vector is treated independently. We denote this filtered result as $\mathbf{d_f}$.

4. The new motion is computed as the sum of the filtered displacement map and the motion at the beginning of the iteration, $\mathbf{m_{i+1}} = \mathbf{m_i} + \mathbf{d_f}$.

We note that (assuming the inverse kinematics procedures work), the "original" $\mathbf{m_{i+1}}$ in step 1, and therefore the displacement computed in step 2 $(\mathbf{m_i} + \mathbf{d})$, motions will satisfy the constraints on each frame. The filtering process may provide a motion that does not satisfy the constraints. In fact, there are no guarantees that the result of step 4 $(\mathbf{m_{i+1}})$ will be any closer to satisfying the IK constraints than $\mathbf{m_i}$ in step 1. We cannot make any theoretical claims about the convergence of the algorithm.

The intuition for the algorithm is that at each iteration we determine what changes to the motion are needed to satisfy the geometric constraints on each frame (step 1 and 2). We then remove any part of the changes that would violate the temporal constraints. We include only the portion of the changes that are acceptable. These changes are not enough to fully satisfy the spatial constraints since they are only part of the changes determined necessary.

To create an algorithm that prefers low frequency changes, but does whatever is necessary to satisfy the geometric constraints, we use different filters on each iteration of the algorithm. This is accomplished in Lee and Shin's system through

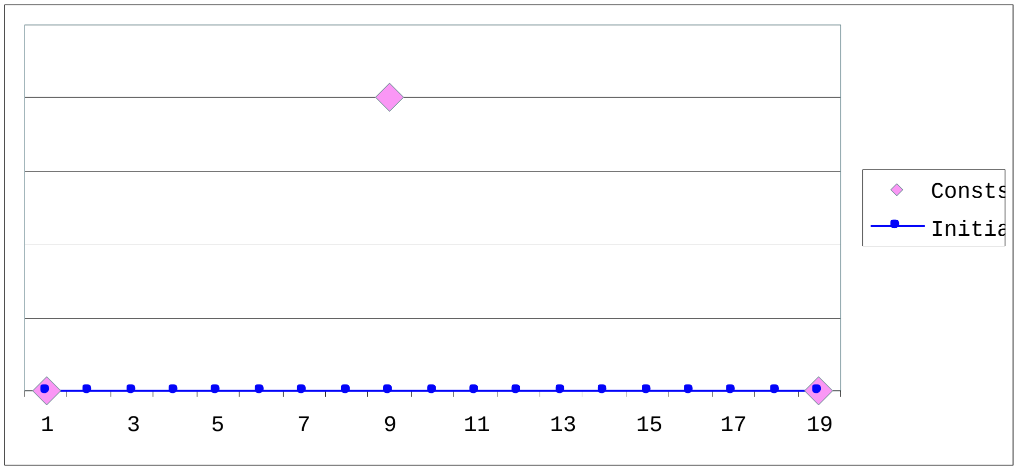

Figure 1: Initial motion and constraints for the trivial example
.

the use of B-Splines with increasing numbers of control points. In our approach, we use filters with increasingly high cutoff frequencies (e.g. smaller and smaller filter kernels). Intuitively, this causes as much of the change to be accomplished by the low frequency changes.

We believe that this approach could be implemented in the scripting language most major animation package. To date, we have not actually created such an implementation, however, we have done paper design experiments with 3D Studio Max and Maya.

4.1.1 A Trivial Demonstration

To provide some insight as to how the PFIK+F algorithm works, we examine a trivial example: a 2D particle whose initial motion is along the X axis. We constrain the particle to be on the X axis at the beginning and end of the motion, and to have a height of n units halfway through the motion. To simplify the illustrations, we do not allow the solver to alter the x positions of the particle (so that the x position of the particle is equal to the time step). The initial problem is illustrated in Figure 1.

Walking through the steps of the process, we begin by “solving” the constraints on each frame. For this trivial example, solving merely means resetting values at the constrained frames. Subtracting the input to this iteration from this motion provides a displacement map, which is filtered to compute $\mathbf{d_f}$, which is added to the starting point of the iteration to provide the result of the iteration. This process

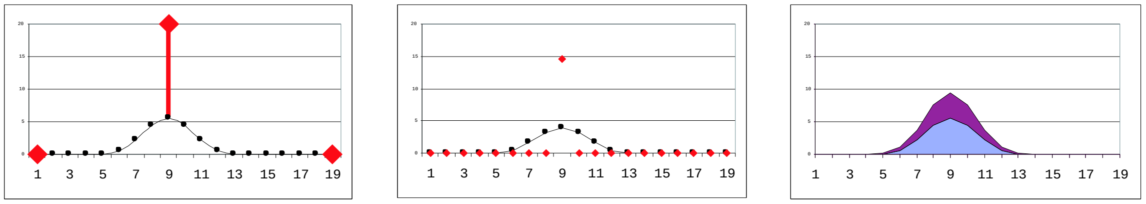

Figure 2: The steps of an iteration of the solver on the trivial example. The second iteration of the solver is shown. Left: the results of the per-frame solve provide the displacements. Center: the displacements are filtered to provide filtered displacements. Right: the final result sums the filtered displacement and the initial curve.

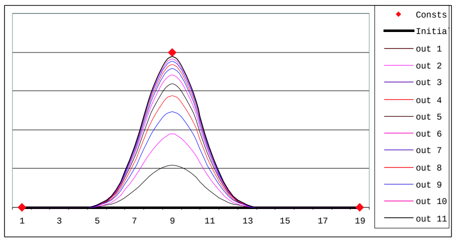

Figure 3: The path of the particle on each iteration of the solver.

is illustrated in Figure 2. The convergence of this process is shown in Figure 3. For this example, each iteration uses an identical filter: in this case a binomial filter of width 9 (the kernel $[1, 8, 28, 56, 70, 56, 28, 8, 1]$). In this case the solver will never converge exactly – each iteration only moves the center point 30% closer to the goal.

Figure 4 shows a different example illustrating how the addition of frequency bounded changes preserves the original motion. In this case, the initial signal has high frequencies and is subject to the same constraints as the previous example. Figures 5 and 6 mimic the illustrations provided for the simpler example.

To make the example more challenging, we add additional constraints in Figure 7. Again, the process is illustrated in Figures 8 and 9. Even this trivial example has the property that it will never converge: it is not possible to meet the spatial constraints without adding high-frequencies to the initial motion.To better handle

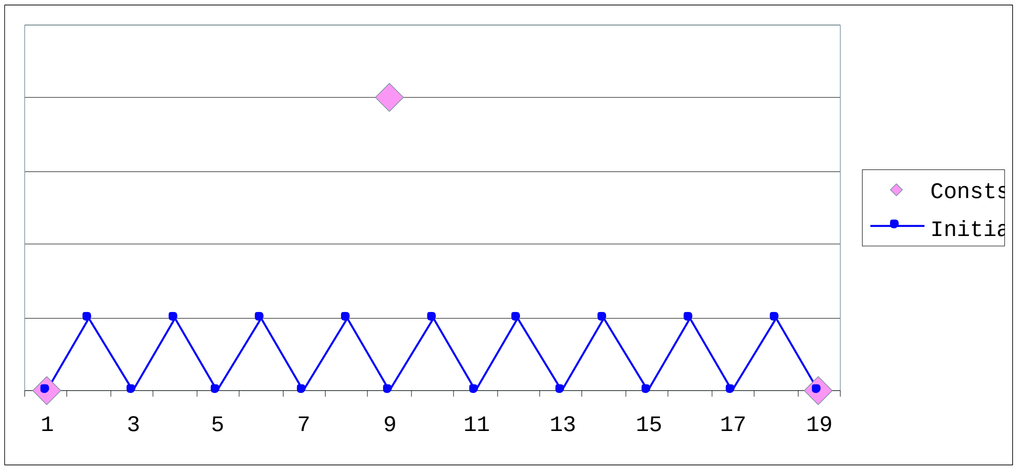

Figure 4: Initial motion and constraints for the trivial example

.

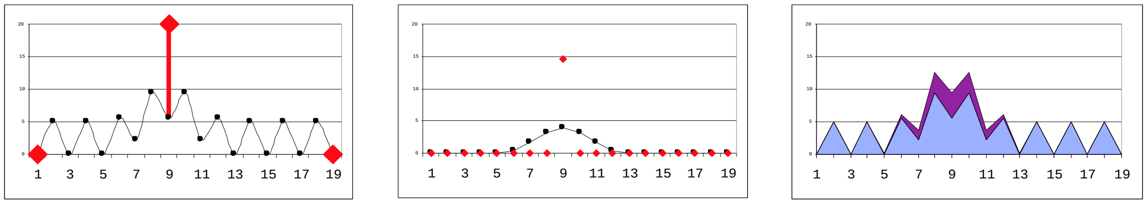

Figure 5: The steps of an iteration of the solver on the wave example. The second iteration of the solver is shown. Left: the results of the per-frame solve provide the displacements. Center: the displacements are filtered to provide filtered displacements. Right: the final result sums the filtered displacement and the initial curve.

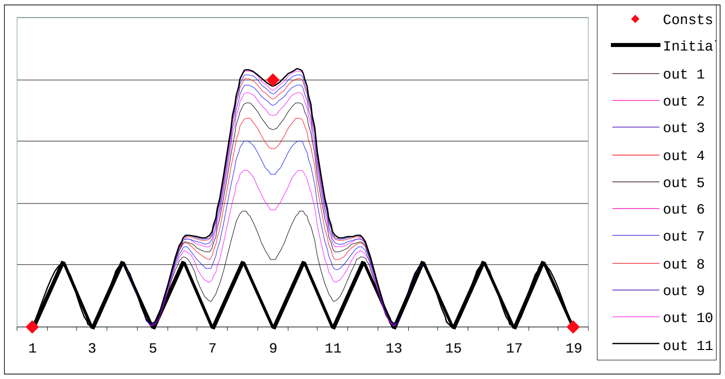

Figure 6: The path of the particle on each iteration of the solver.

such cases, we change the solver to use a decreasingly large filter kernel on each iteration. The result of this process is shown in Figure 10.

4.2 Experiments

We have implemented the PFIK+F approach within our Timelines animation testbed. Our spacetime constraints solver, described in [Gle98], is applied to each frame individually to serve as the IK solution. By implementing this approach in our existing framework we are able to apply it to the same problems that we have been exploring with other approaches.

We have applied our PFIK+F solver to a range of problems, including retargeting motions to new characters, constraint editing, and path editing. In the first and last type of problem, a change is made across the entire motion and a new solution is found to existing constraints. In a constraint editing problem, the constraints are manipulated interactively by the user.

When our spacetime solver is applied in batch mode, its performance is slower than when used interactively. This is because we make additional performance concessions (as described in [Gle97]) for interactivity . More significantly, in batch mode, we lose “differentialness” [Gle94]. That is, during dragging there are only small changes made between each solution. Therefore, the previous solution is a good starting point in the search for the next. Interestingly, our PFIK+F solver does not gain major advantage from differentialness. While a good starting point might make each individual IK solution slightly faster[3], the total number of iterations is the same.

[3]We believe the solver described by Lee and Shin takes a constant amount of time, independent of how far the initial guess is.

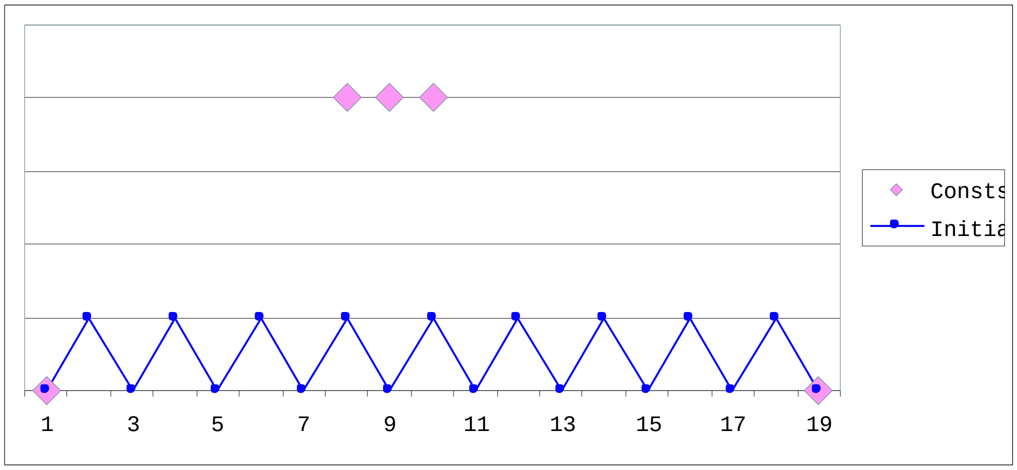

Figure 7: Initial motion and constraints for the third simple example

.

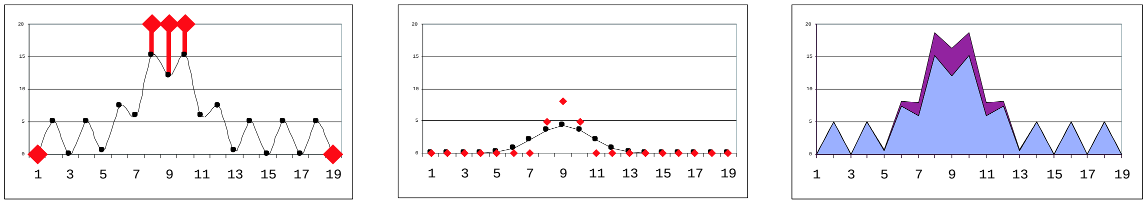

Figure 8: The steps of an iteration of the solver on the third example. The second iteration of the solver is shown. Left: the results of the per-frame solve provide the displacements. Center: the displacements are filtered to provide filtered displacements. Right: the final result sums the filtered displacement and the initial curve.

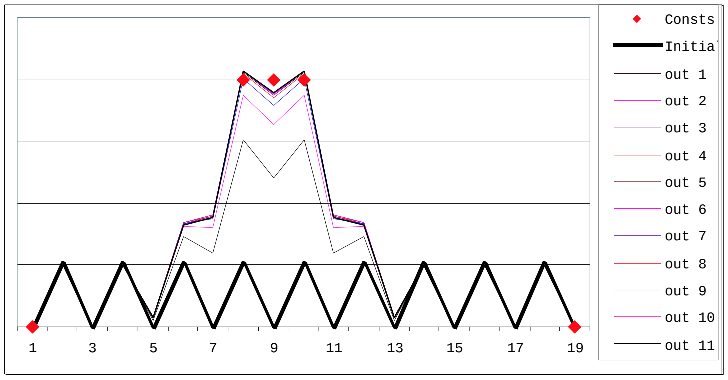

Figure 9: The path of the particle on each iteration of the solver.

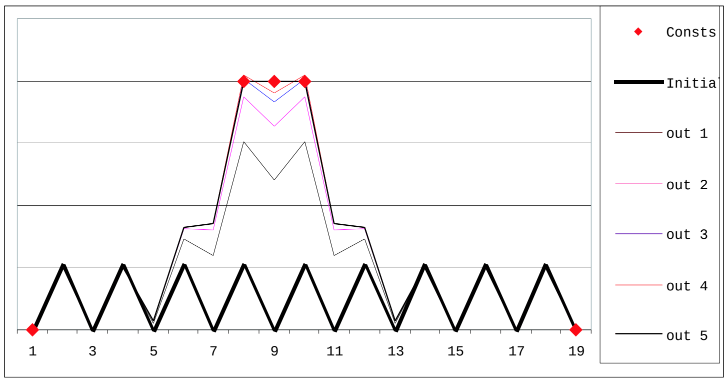

Figure 10: The path of the particle on each iteration of a solver where a filter of decreasing size is used on each iteration.

One of original stated advantages of Lee and Shin's original paper of the PFIK+F approach over the spacetime approach is performance. In our system, it is unfair to run a horse race between the two approaches because we have done different amounts of performance tuning on each. We will explore the performance issues in section 5.1.

We discuss a few representative experiments chosen to highlight some of the differences in the results of the two solvers. All timings provided are run on a Dell Precision 610 workstation with a 550mhz Pentium III Xeon Processor and 512M of RAM.

4.2.1 A Simple Retargeting Problem

We first apply the solvers to the task of retargeting the walk of a fashion model down a catwalk to a shorter character, 60% of the original's height. Footplant constraints are applied to the characters heels and the balls of her feet. The motion has 183 frames on which 954 scalar constraints have been placed [4] to the heel and forefoot strikes. The solver is permitted to move the positions of the footplants, however it must keep the feet on the floor and in place when planted. This example is similar to the one used in Section 5.1.2.

We should note that because the character was scaled uniformly, there is a trivial solution to this retargeting problem: scaling the translation by the same 60% as the limbs, as shown in Figure 11. This solution is found by the heuristic starting point finder described in [Gle98], therefore we omit the heuristic on this example. Instead, we force the character to walk using the footplant positions of the original, causing a more challenging retargeting problem.

Our spacetime solver (the least-squares method described in [Gle98]) produces a good result for the motion when the control points of the B-Spline for the displacement map are placed 5 frames away from one another (3 frame spacing shows noticeable "lunging" to meet footplants). The solver only moves the footplants a small amount from their original positions, so the small character does take very long steps in relation to its height (this is normally addressed by the scaling heuristic). Otherwise, the solver produces an appealing motion. In under half a second, the solver provides a solution where the total error magnitude over the entire motion is less than 2 centimeters.

5 iterations of the PFIK+F solver take slightly over 2.5 seconds to provide a solution where the total error magnitude is approximately 6 centimeters (no single frame shows more than a few millimeters of error). The resulting motion exhibits a little more "lunging" to meet the long stride lengths than the spacetime solution,

[4]We count scalar equations, so a constraint that positions a point in 3D counts as 3 constraints.

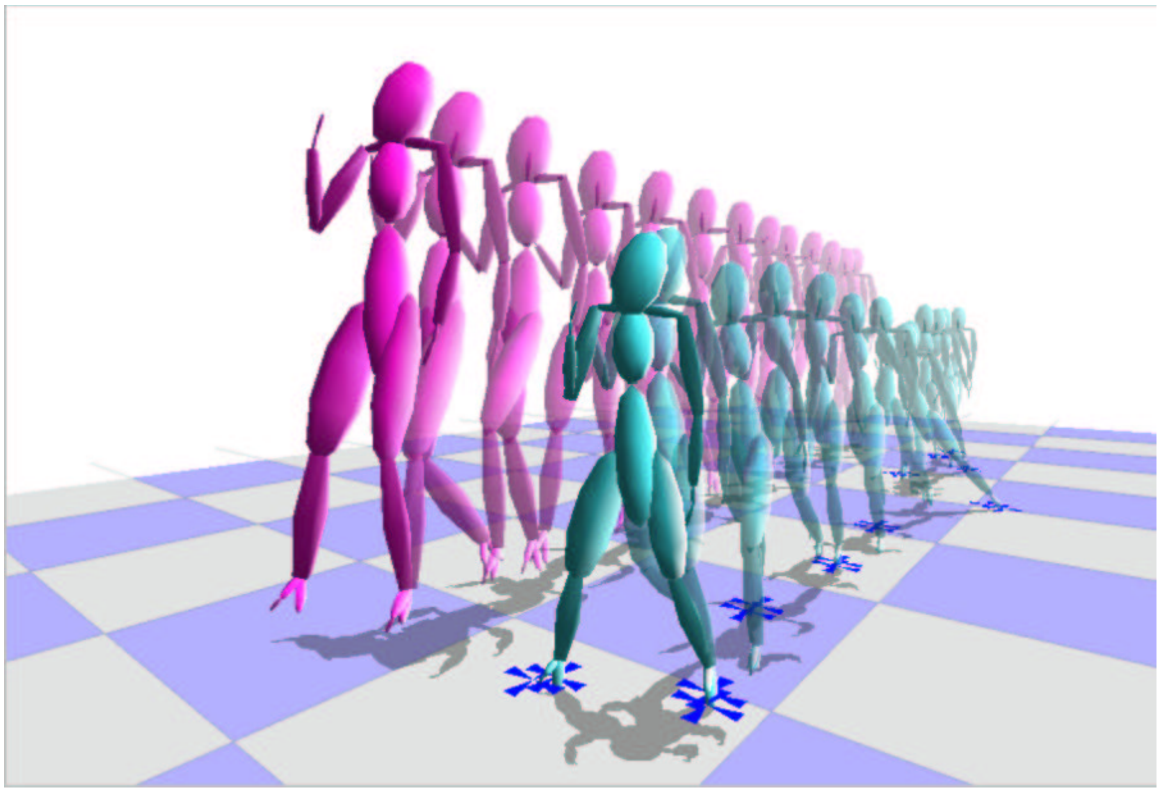

Figure 11: The model's walk is retargetted to a smaller character (right).

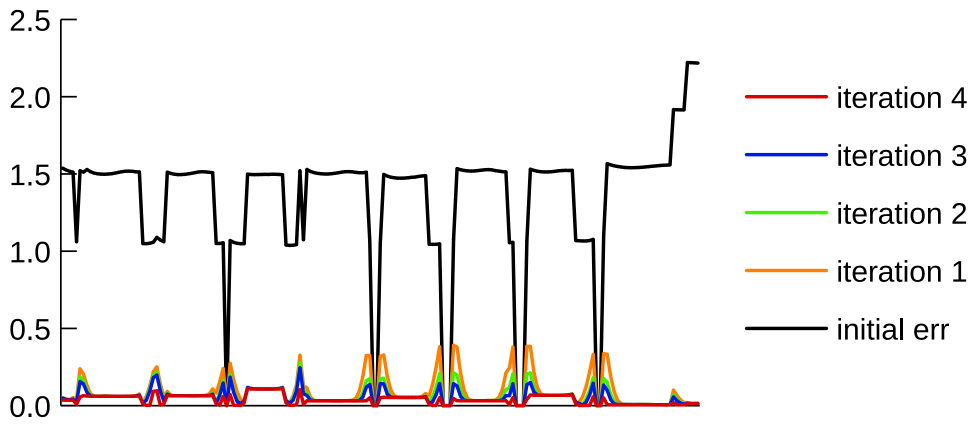

Figure 12: Total per-frame error for the catwalk example for each iteration of the PFIK+F solver. Error is measured in feet.

which is a sign that we should have applied more filtering to the first few iterations of the solver (our implementation presently only supports limited filtering).

Figure 12 shows the amount of error per frame after each iteration of the solver.

4.3 Real Retargeting Problems

We have also applied both the PFIK+F and spacetime solvers to a number of "real" retargeting problems where we adapt existing motions to new characters. In such examples, we find motion and character geometry in libraries, determine a skeleton to fit inside of the geometry, retarget the motion to work with this skeleton, and then use the skeleton to drive the movement of the geometry. Some of the menagerie of characters that were tested are shown in Figure 13. In all cases, the characters were found in libraries, as were most of the test motions.

The spacetime-based solver reliably provided subjectively acceptable results over a very wide range of motions and characters. The PFIK+F solver was not quite as reliable. On several examples, the solver failed to converge to a solution that met both temporal and geometric constraints. Also, conflicting constraints and anomalies with euler angles (filtering over zero-crossings) often caused the PFIK+F solver to fail to return a visually acceptable solution, even in cases where the spacetime solver was able to provide answers.

In terms of temporal performance, the differences are difficult to characterize. There were examples for each solver where it was the fastest. This is not surprising given the iterative nature of the algorithms (especially the spacetime solver).

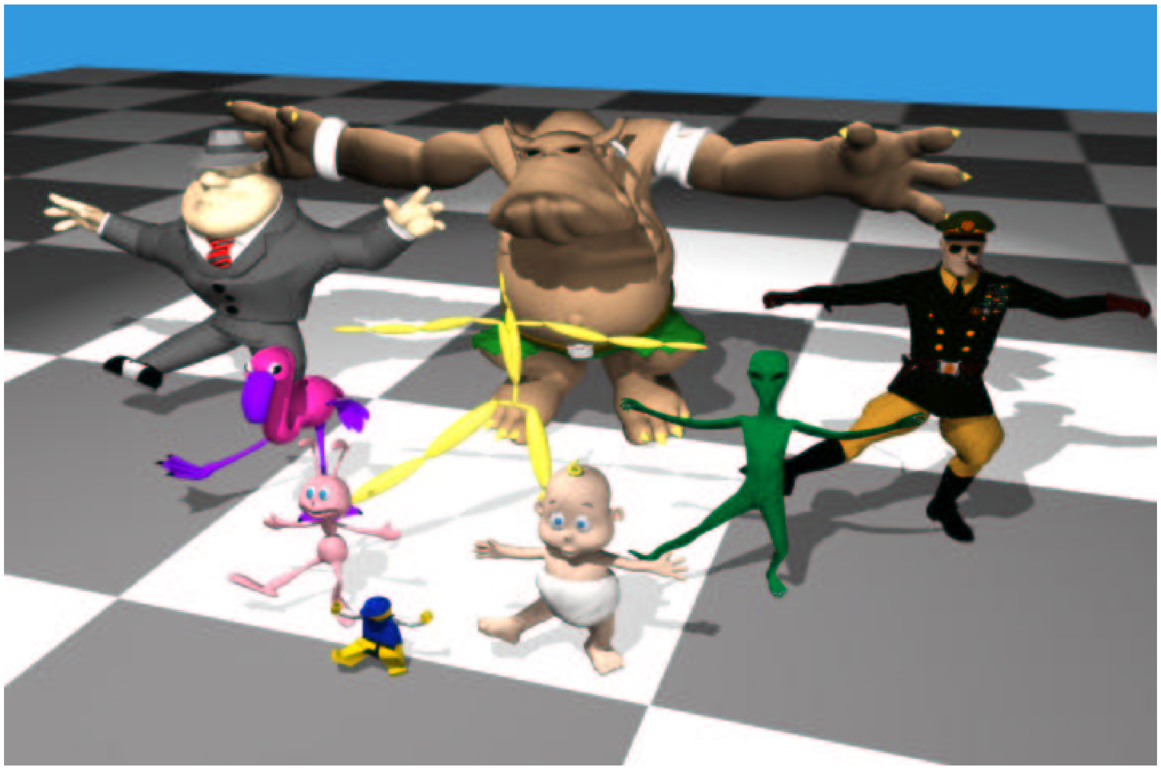

Figure 13: A variety of characters that we have retargetted motions to. The original skeleton from the motion character data is shown in the center.

4.4 Path Editing

In this section we consider the problem of transforming the path that a locomoting character moves along. For example, we transform a motion of walking in a straight line to walking in a curved line. The details of the method, described in [Gle01], are unimportant for this discussion: as far as the solver is concerned, the motion is simply transformed in a manner that does not preserve the constraints so they must be re-established. In practice, the path editing and solution take small amounts of time: we are able to interactively change the paths of motions with several hundred constraints.

One issue in using the PFIK solver for this approach is that a single position for each constraint must be found. This does not trivially extend to constraints that exist over a duration of time. The position of such a constraint may be mapped to different locations at different times. One way to view the problem is that a single instant must be chosen for the constraint to determine how to transform its position, otherwise its position may not be mapped to a unique location.

Different choices in time lead to different constraint positions which, in turn, lead to different characteristics of the motions. The three most obvious choices are the beginning, middle, and end of the duration of the constraints. Figure 14 shows how this choice affects a walking motion. The differences are subtle, and illustrate the challenge of motion transformation: there is no clearly "correct" answer given the limited information that we have. We are left with either building more sophisticated models to determine the movement, relying on heuristics (or user input) to select amongst the myriad of small choices, or reconciling ourselves to never having the perfect answer.

One specific caveat: entire footplants must be dealt with at a single instant, the heel and toe strikes cannot be given separate times. If they are transformed at the same instant, their positions will be transformed rigidly. Otherwise, the heel and toe strike positions may be transformed differently, changing the distance between them which is impossible if the character is a rigid skeleton.

The spacetime approach provides the option of allowing the solver to determine where the footplants are placed.

5 Comparing Approaches

At a high level, the different methods discussed so far all achieve the same result: they allow for finding motions that meet a set of spatial (geometric) and temporal constraints. Our goal in this section is to understand the differences between the methods in terms of what they accomplish.

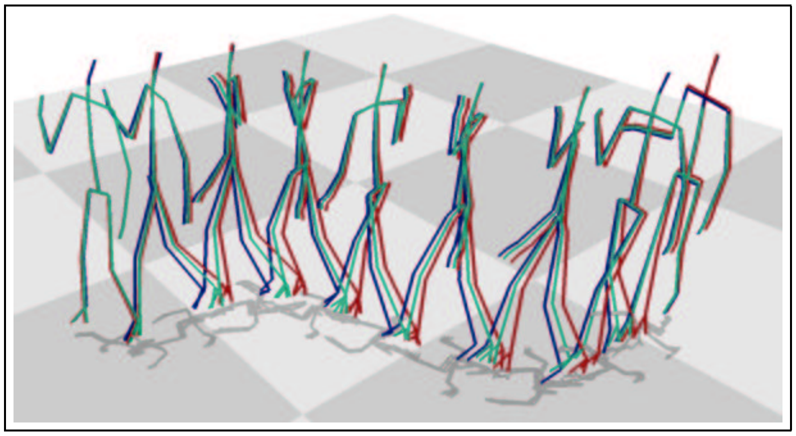

Figure 14: The example of Figure 11 is adapted using different methods for choosing the new footplant positions. The different colors represent the motion obtained by moving footplants to their beginning, middle or end of their duration.

Our focus is on comparing the PFIK+F and Spacetime approaches as they address similar problems:

- Both the spacetime and PFIK+F approaches are applicable to motion from any source: keyframed data can be sampled. The PKIK methods only work given good keyframes, and therefore are not applicable across the range of problems of these other methods.

- Both methods are inherently off-line. That is, they require the entire motion to examine. While this has the benefit that the methods can look at larger ranges of the motion to create better temporal constraints, the ability to look into the future may not be practical. The PFIK approach seems to be uniquely suited for on-line applications where future durations of the motion are unavailable, and past decisions cannot be changed [CK99].

- The PFIK approach can be viewed as a variant of the PFIK+F approach with an identity filter. In practice, a more sophisticated IK technique would be used to enforce temporal constraints. However, it appears that the class of temporal constraints that can be achieved this way is different than what can be achieved with the other methods.

- We do not consider the physical approaches as they are predominantly a different objective function and constraint for the spacetime approach. Also, because the methods have not been demonstrated on comparable classes of problems, analysis is difficult.

- The PFIK+F and Spacetime approaches are the only methods that have been demonstrated across a wide and similar set of editing problems, and have demonstrated viable interactive performance on these problems.

5.1 Comparison of Performance

The actual performance is as much a measure of the implementation as of the method. While modern processors provide impressive performance for applications, achieving the potential of these applications typically requires a great deal of tuning and care. [5]

More specific to our comparison, it seems that even within a given approach, there are a wide range of design decisions and tradeoffs that can be made. For example, compare the tradeoffs made in the spacetime approach to achieve interactive performance on a circa 1996 personal computer [Gle97] with the decisions made to create a non-interactive system on a high-performance workstation where maintaining physical constraints were essential[PW99].

Similarly, Lee and Shin make concessions to performance in their implementation of the PFIK+F approach. Through the use of the simple geometric IK solver, they trade solution quality and generality for performance. In our implementation, we made fewer concessions by employing a more general, however more expensive, optimization-based solver. For our analysis, we consider the latter because it is more able to handle the range of constraints that the spacetime approach can. Specifically, it can allow arbitrary points to be positioned, multiple points to be placed simultaneously, joint limits to be used, and other types of constraints to be applied.

The iterative non-linear solvers used for spacetime typically only require a small number of iterations to converge to a solution. This makes the performance difficult to characterize: a small perturbation in the initial conditions may require the solver to make an extra iteration or two, which is substantial when the solver requires only a handful of iterations altogether.

5.1.1 Computational Complexity

To consider the complexity of each method, we assume that the motion has n frames. Each frame of the motion has a small number of constraints, therefore the problem must consider $O(n)$ constraints. Similarly, in each frame, the character

[5] Anectdotally, we have achieved very substantial speedups in our code by attending to issues in memory system performance. A cache miss can take as long as a large number of floating point operations.

has the same number of variables to describe its configuration, so the total number of variables is again $O(n)$.

Solving a set of non-linear equations, whether an IK problem or a spacetime optimization, is non-deterministic. However, in practice, we do not seek an exact solution and typically bound the number of iterations by a small constant - if a suitable solution is not found in this number of iterations, we assume one will not be found. In practice, we find this number to be quite small for our solver.

At each iteration of the non-linear solver, constraint errors and derivatives are computed, a linear system is solved[6]. In [Gle94] we argue that constraint problems are sparse for many graphics applications because each constraint only affects a small number of variables independent of the overall size of the problem. This means that the constraints and their derivatives can be computed in time linear with the number of constraints and variables. The only step with worse than linear complexity is solving the linear system.

Under these assumptions, the computational complexity of the PFIK+F approach is $O(n)$. At each iteration of the solver, an IK problem must be solved for each frame of the motion. The amount of time each of these problems take is independent of the total length of the animation. Similarly, other steps in the process also are independent of the duration of the animation. We assume that the solver runs for a small constant number of iterations.

The analysis of the spacetime solver is more complex. Following the analysis of [Gle94], we can see that the amount of work done for each constraint in evaluating and differentiating is a constant independent of the total number of constraints. The constraint Jacobian (the derivative of each constraint with respect to each variable) will consist of $O(n)$ rows (one per constraint), each having a small constant number of non-zero entries (each constraint refers to only a small number of instants in time). This sparse linear system, with $O(n)$ non-zero entries, can be solved in $O(n^2)$ time with an iterative method ($O(n)$ iterations each requiring an $O(n)$ sparse-matrix multiply [BBC$^+$94]). [7] Exploiting the banded nature of these matrices can reduce the complexity even further, yielding $O(n)$ complexity.

For each of these analyses, we have limited the solver to a constant number of iterations. With the PFIK+F approach, we have the choice of ensuring that despite this bounded number of iterations that either the temporal constraints are satisfied (by terminating with a filtering step) or that the spatial constraints are satisfied (by terminating with a step that skips the filtering). The spacetime approach does not provide this choice: because we cannot be guaranteed of convergence, we cannot

[6] This characterizes a wide class of methods.

[7] Technically, iterative solution of the linear system with $O(n)$ non-zero elements takes $O(n^2)$ time only with infinite precision arithmetic. In practice, the use of damping and large tolerances allow us to achieve this performance.

Motion		PFIK+F	Spacetime		
Frames	Constraints	Time	Time	Iters	Time/Iter
890	4890	4.8	7.2	4	1.8
1780	9780	9.5	13.3	5	2.7
2670	14670	14.3	25.4	8	3.2
3560	19560	19.0	17.6	3	5.9
4450	24450	23.9	21.7	3	7.2
5340	29340	28.1	25.1	3	8.4
6230	34230	32.3	30.4	3	10.1
7120	39120	36.7	34.9	3	11.6
8010	44010	41.1	38.0	3	12.7
8900	48900	45.4	43.7	3	14.6
9790	53790	49.7	47.8	3	15.9
10680	58680	53.9	51.5	3	17.2
11570	63570	59.3	56.7	3	18.9
12460	68460	62.0	61.1	3	20.4
13350	73350	75.9	73.0	3	24.3

Figure 15: Results of the performance experiment. The timings provided are computed on a Dell Precision 610 workstation with a 550mhz Pentium III Xeon Processor and 512M of RAM.

know that the spatial constraints will be satisfied.

5.1.2 Performance Experiment

To explore the asymptotic performance of the algorithms, we devised an experiment where we created very long motions with many constraints. In order to keep the "difficulty" of the numerical problem constant, we needed to use similar constraints. To create the long examples, we began with a loopable 89 frame walking motion on which footplant constraints were applied to the heel and toe. We repeated this motion as necessary to create a motion of the desired length.

Our test was to retarget the motion to a character 60% of the size of the original. The results of this experiment are shown in Table 15. As in the examples of Section 4.2.1, the smaller character is forced to walk in the footsteps of the original character to create a more challenging retargeting problem and the scaling heuristic in not used.

As expected, the timings for the PFIK+F solver are linear (when the last example is excepted), as shown in Figure 16.

The behavior of the spacetime solver seems less simple. For each test, a small

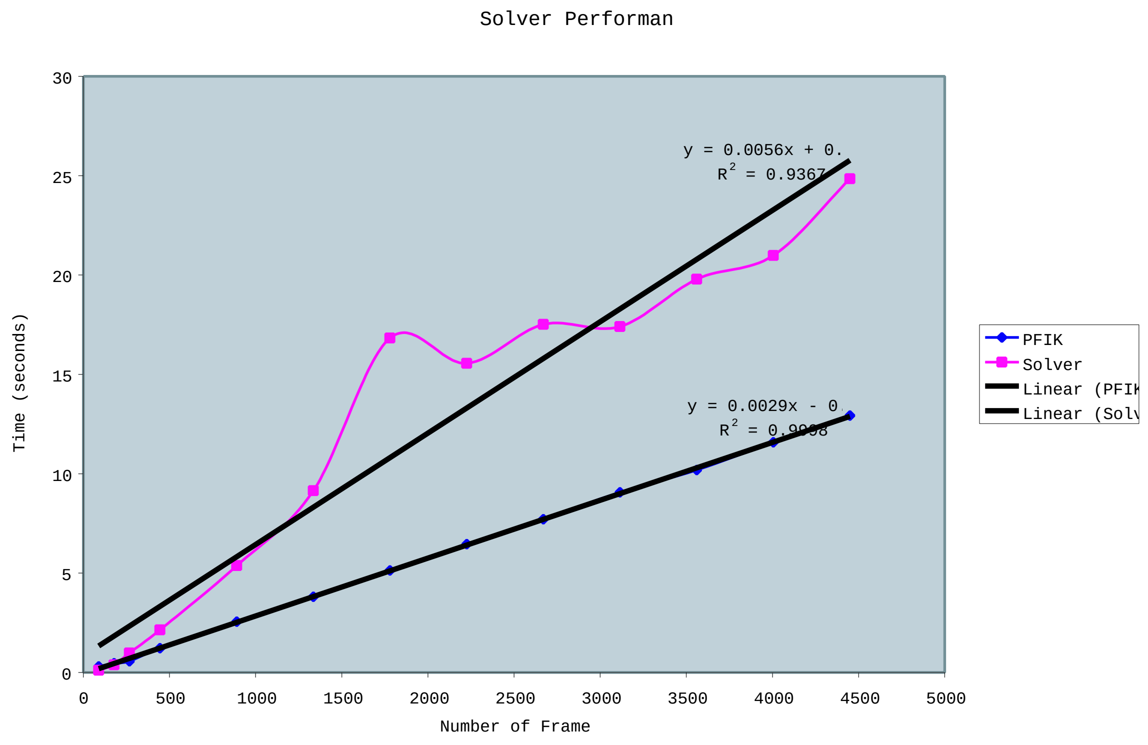

Figure 16: Performance of the PFIK+F and Spacetime solvers on problems of varying size.

number (between 3 and 8) of iterations was required by the solver. When the solver took more than 3 steps, it rarely made much progress (the last steps of the solver improved the solution by less than 1%). The solver's choice to take more iterations does not seem to be coupled to the quality of the result, or the size of the problem. In theory, it is tied to the non-linearity of the problem. In practice, there seems to be a degree of luck as to how well the solver does at finding a solution and determining that it is unlikely to make substantial improvements by taking more steps.

When the timing for the spacetime solver is examined on a per-iteration basis, we would expect the time to be dominated by the step of solving the sparse linear system. Because we use a conjugate-gradient solver (similar to the one described in [PFTV86]), we would expect $O(n^2)$ behavior. In Figure 17 we instead see a result that is linear (to a very high statistical confidence). The best explanation for this is that the matrix is well enough conditioned, and the stopping tolerance is large enough, that an effectively constant number of iterations is required. Rather than claiming that this is likely to happen across problems, we use this example as a demonstration that the time required for the spacetime solver depends as much on how hard the problem is as how big the problem is.

5.2 Range of Constraints Handled

Different types of solution techniques allow differing amounts of generality to the types of constraints that they may allow. The set of constraints that we might want to preserve in a motion is open. Ultimately, we would might have constraints to specify not only geometric details, such as the positions of end-effectors and limits on joints, but also high-level properties such as the "naturalness" of a motion, or its "mood." To date, mathematical descriptions of such high-level properties have not been identified. However, generality in a solver means that such constraints are more likely to be usable when they are developed.

The class of constraints a system can handle depends on much more than the solver's approach. The details of the implementation of the solver parts make limitations. For example, not all inverse kinematics solvers implement all constraints possible in such solvers. Sometimes limitations on constraints are more than a solver issue: for constraints to be used effectively, there must be methods provided to specify and visualize them.

Our question is whether the motion editing *approach* affects the range of possible constraints. Just because an approach can handle a particular constraint does not mean that all implementations of the approach can. In this section we explore the generality afforded by the spacetime approach over that provided by an IK-based approach. That is not to say that a sophisticated PFIK approach may not

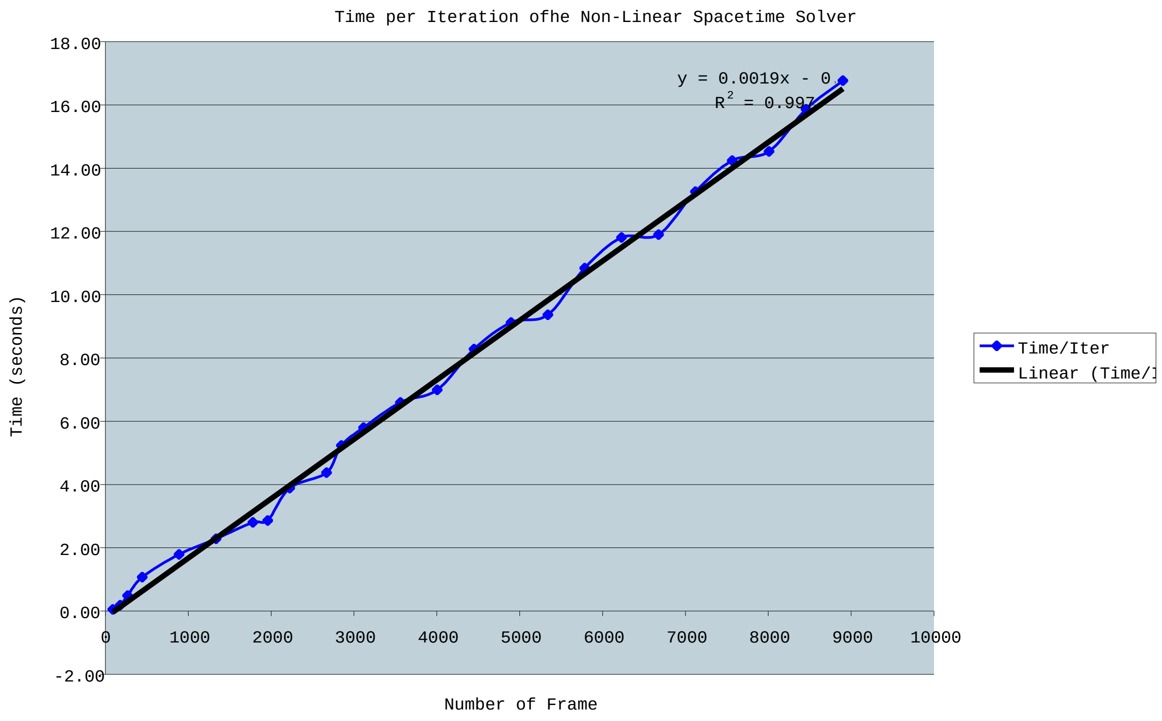

Figure 17: Time required per iteration of the spacetime solver for various problem sizes. Time is in seconds, the line represents a linear regression.

implement a wider variety of constraints than a simple implementation of a spacetime approach. In this section, when we refer to the constraints of an approach, we are discussing what constraints are *possible* with *an* implementation of the approach. We are seeking fundamental limitations in the approaches.

Our first observation is that the set of spatial constraints possible in a spacetime approach is a proper superset of what an IK solver can do. At worst, the same equations used in an IK solver can be used in a spacetime solver applying to a single instant in time. In terms of temporal constraints, the situation is less clear. A spacetime solver can recreate a PFIK solver by using an objective function that does not couple frames. However, the preference for low frequencies given by the PFIK+F approach (with soft temporal constraints) is more difficult to recreate. At worst, it can be implemented within the spacetime approach by making repeated calls to the spacetime solver with gradually increasing frequency limits.

The spacetime approach can handle constraints that a per-frame approach cannot. A simple example is that it is possible to create constraints that hold over time. We call such constraints variational, as they place limits on the function curves. In a PFIK-based approach, constraints can only be placed at discrete instants. To date, this advantage has not been exploited: variational constraints are implemented by sampling at discreet instants.

A related issue is inter-frame constraints: that is, a constraint that relates two or more different instants in time. For example, we might want to specify that a hand is in the same place in several different frames. We note that this implies that we do not have a specific value in mind: if so, we could have equivalently specified a position for each individual constraint.

A major flexibility of the spacetime approach comes from the fact that it handles “don’t care” constraints well. The information to determine values for the “don’t cares” can come from any part of the motion. Such constraints are common. While many constraints specify interactions with specific points in the world, an even larger class of constraints place only general restrictions. For example, most footplant constraints require that the foot be on the floor, and that the foot does not slip while on the floor. However, the exact position on the floor is typically unimportant (unless the character is walking in a mine field, crossing a river on rocks, or another such contrived case). Allowing the solver to determine the footplant positions based on criteria from other frames leads to more natural retargeting and more useful constraint-based control. For example, a character whose goal is moved may take larger steps in order to reach its destination.

5.3 Order Independence

Interframe constraints with “don’t cares,” as described in the above section, suggest another limitation of per-frame approaches: they are order dependent. We must choose the configuration for one frame before choosing another.

To illustrate the order independence property, consider placing a constraint on the motion specifying that the character must be standing in place. In a PFIK approach, each frame would have to look to the results of a previously solved frame to choose a position: were it to choose an unsolved frame, that frame might change in the future. Consider if an additional constraint specifying the character’s position is placed on one of the frames of the motion. Were this frame solved first, then all other frames would end up in the right place, if another frame was chosen to be solved first, when the positioned frame’s turn to be solved arrived, it would have a conflict: it would need to choose between being in the same place as the previously solved frames or meeting its positional goals.

Some of the advantages of the spacetime approach might be recreatable in a PFIK+F approach if there were a mechanism for determining the order in which to solve constraints. An illustration of this is in Figure 14 where a single time is chosen to determine the position of each footplant. The declarative, order independent, nature of constraints are one of the features that make constraint-based descriptions attractive.

5.4 Quality of Result

Getting an answer quickly is nice. However, it is also important to get “good” answers. Unfortunately, it is difficult to define what a good answer is since there is no ground truth answer to compare with. In a motion editing tool, one might define the correct answer as being what the user wants. Therefore, the quality of a solution is directly related to how well it can be controlled to provide a user with a desired result.

Just as the specifics of implementation may mean as much as the approach for the performance of solvers, they also have considerable influence on the quality of the resulting motions. For example, in a PFIK+F or PFIK approach, an IK solver that tends to pose the character in natural and plausible ways is more likely to provide good motions than a solver that doesn’t.

The specific examples applied have impact on the quality of the resulting motions. Achieving good results requires having good motions to start with[8]. It also requires good spatial and temporal constraints to be defined for the initial motion.

[8]Generally - the use of motion editing techniques to turn bad motions into good ones has not been fully explored yet

Both of these factors are somewhat independent of the motion editing approach. However, the approach can impact quality through the types of constraints it allows for and how well it operates in the face of “bad” constraints, such as conflicts, and under-specified cases.

5.5 Summary: PFIK+F vs. Spacetime

The advantages of the spacetime approach:

1. Wider range of objective and temporal constraints possible, however this flexibility has not yet really been exploited.
2. Order independence.
3. Better handling of “don’t care” geometric constraints.
4. Interframe geometric constraints.

The advantages of the PFIK+F:

1. Simpler to implement. A PFIK+F solver can basically be build from “standard” parts in an animation system.
2. Can insure solving the geometric constraints.
3. Easily allows for weak frequency limit constraints.
4. More predictable solution times.

6 Conclusions

In this paper, we have examined a range of methods for performing constraint-based motion editing. With this menagerie of techniques at their disposal, developers can choose techniques that meet their problems. Some classes of editing applications suggest particular solution approaches, for example PFIK for on-line applications. For general off-line editing applications, several approaches apply. Spacetime and PFIK+F methods, the two primary contenders in such applications, each have a set of strengths and weaknesses, and physically-based approaches offer promise as a viable approach in the future.

Acknowledgements

This research is supported by a grant from Microsoft Research, equipment donations from Intel, and IBM, and software donations from Pixar, Autodesk, Alias/Wavefront, and Metacreations. The author is supported by an NSF Career Award This research is supported by an NSF Career Award "Motion Transformations for Computer Animation" CCR-9984506.

The author would like to thank Andrew Prock for his assistance in the implementation of path editing, Taylor Wilson of House of Moves Motion Capture Studios for the sample data, Andy Gardner for his assistance in performance evaluation, Jehee Lee and Sung-Yong Shin for their discussions of the methods, and Alex Mohr for his help with proofreading.

References

[BBC$^+$94] 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.

[BMW87] Norman I. Badler, Kamran H. Manoochehri, and Graham Walters. Articulated figure positioning by multiple constraints. *IEEE Computer Graphics & Applications*, 7(6):28–38, June 1987.

[BW95] Armin Bruderlin and Lance Williams. Motion signal processing. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 97–104, August 1995.

[CK99] Kwang-Jin Choi and Hyeong-Seok Ko. On-line motion retargetting. *Pacific Graphics '99*, October 1999. Held in Seoul, Korea.

[Coh92] Michael F. Cohen. Interactive spacetime control for animation. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):293–302, July 1992. ISBN 0-201-51585-7. Held in Chicago, Illinois.

[GL98] Michael Gleicher and Peter Litwinowicz. Constraint-based motion adaptation. *Journal of Visualization and Computer Animation*, 9:65–94, 1998.

[Gle94] Michael Gleicher. *A Differential Approach to Graphical Interaction*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1994.

[Gle97] Michael Gleicher. Motion editing with spacetime constraints. In Michael Cohen and David Zeltzer, editors, *Proceedings 1997 Symposium on Interactive 3D Graphics*, pages 139–148, apr 1997.

[Gle98] Michael Gleicher. Retargeting motion to new characters. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 33–42. ACM SIGGRAPH, Addison Wesley, July 1998.

[Gle99] Michael Gleicher. Animation from observation: Motion capture and motion editing. *Computer Graphics*, 1999. Invited paper to appear in a special issue edited by S. Seitz and R. Szeliski.

[Gle01] Michael Gleicher. Motion path editing. In J. Hughes and C. Sequin, editors, *Proceedings 2001 ACM Symposium on Interactive 3D Graphics*, March 2001.

[Gra98] F. Sebastian Grassia. Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools*, 3(3):29–48, 1998. ISSN 1086-7651.

[KB82] J. U. Korein and N. I. Badler. Techniques for generating the goal-directed animation of articulated structures. *IEEE Computer Graphics & Applications*, 2(9):71–81, November 1982.

[Lit91] Peter C. Litwinowicz. Inkwell: A $2\frac{1}{2}$-D animation system. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 113–122, July 1991.

[LS99] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. *Proceedings of SIGGRAPH 99*, pages 39–48, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

[LS00] Jehee Lee and Sung Yong Shin. Multiresolution motion analysis and synthesis. Technical Report CS-TR-2000-149, Computer Science Department, KAIST, 2000.

[Mac90] Anthony A. Maciejewski. Dealing with the ill-conditioned equations of motion for articulated figures. *IEEE Computer Graphics & Applications*, 10(3):63–71, May 1990.

[PB91] Cary B. Phillips and Norman I. Badler. Interactive behaviors for bipedal articulated figures. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):359–362, July 1991. ISBN 0-201-56291-X. Held in Las Vegas, Nevada.

[PFTV86] William Press, Brian Flannery, Saul Teukolsky, and William Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, England, 1986.

[PW99] Zoran Popovic and Andrew Witkin. Physically based motion transformation. *Proceedings of SIGGRAPH 99*, pages 11–20, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

[PZB90] Cary B. Phillips, Jianmin Zhao, and Norman I. Badler. Interactive real-time articulated figure manipulation using multiple kinematic constraints. *1990 Symposium on Interactive 3D Graphics*, 24(2):245–250, March 1990. ISBN 0-89791-351-5.

[TGB00] D. Tolani, A. Goswanmi, and N. Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models*, 62:353–388, 2000.

[UAT95] Munetoshi Unuma, Ken Anjyo, and Ryozo Takeuchi. Fourier principles for emotion-based human figure animation. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 91–96. ACM SIGGRAPH, Addison Wesley, August 1995.

[Wel93] Chris Welman. Inverse kinematics and geometric constraints for articulated figure manipulation. Master's thesis, Simon Frasier University, September 1993.

[WK88] Andrew Witkin and Michael Kass. Spacetime constraints. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 159–168, August 1988.

[WP95] Andrew Witkin and Zoran Popović. Motion warping. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 105–108, August 1995.

[ZB94] Jianmin Zhao and Norman I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics*, 13(4):313–336, October 1994. ISSN 0730-0301.