Techniques for Motion Editing

This chapter discusses the problem of editing motion: changing or affecting the ways that things move. 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. Motion capture makes motion editing a subject in its own right.

As seen in the previous chapter, new tools are being introduced that specifically address tasks in motion editing. Functionality for altering motion is appearing both as extensions and plug-ins to existing animation systems, as well as part of the standard feature set of newer systems. Such tools will most certainly proliferate as motion capture becomes more common, creating more demand for tools for working with the data it creates.

While all types of functionality in animation systems are evolving rapidly, the tools and techniques for motion editing at an even more extreme pace. Therefore, an understanding of the fundamental issues in motion editing, the technologies underlying the current tools, and some notions of what types of tools may be possible in the future is essential to a user or developer of motion capture or editing technologies. It is important to understand which limitations are fundamental to the problem, and which are simply artifacts of the current generation of tools.

In this chapter, we will explore the basic challenges, concepts, and methods for motion editing, specifically on motions created by motion capture. We begin by looking at the connection between motion capture and motion editing, and how motion capture not only creates a need for special tools for motion editing, but also places some difficult demands on these tools. We will look at the problem of motion editing in general, to gain insights as to why it is a difficult problem, and use the challenges we encounter to motivate techniques. Some technical realities of dealing with motion capture data, such as the representation of rotation, are reviewed to prepare

Preliminary version 4/7/00.
This chapter will appear as Chapter 5 of Motion Capture and Editing: Bridging Principles and Practices, by M. Jung, R. Fischer, M. Gleicher, and J. Thingvold, copyright 2000 by A K Peters, Ltd.

us for the development of editing techniques. We will introduce a variety of techniques to address the challenges of editing, including key reduction and signal processing. We will conclude by describing a Spacetime Constraints approach to motion editing, which is a promising research development.

1.1. Why Edit Motion?

By this point in this book, you are probably well aware of a variety of methods for creating motions by capturing the movements of performers. Motion capture techniques (ideally) should provide you with wonderful motions - why should there be a need to change them? If everything was working correctly, your motion capture data should be an accurate reflection of the reality of a desired performance. Yet the discussion of how to change motions once we have them always seems to be a big part of the use of motion capture.

A common misconception is that the importance of motion editing for motion capture comes from the fact that motion capture is imperfect, and that tools are needed to "clean up" the motion after the fact. While we do not deny that there is a need for this kind of editing, the editing that we consider goes well beyond this. Even when the motion capture data perfectly represents a desired performance, there is often a need to make alterations to the motion, which is where motion editing comes in.

We have several categories of needs for motion editing for captured motion:

- Clean-up: making changes to motion captured data so that it does accurately reflect the performance and allows for an accurate reconstruction.
- Re-use: motion capture data exactly records an event. If we want to re-use the data for something slightly different, say a different character or a different action, we need to edit the data.

- Creating Infeasible Motions: because motion capture record real events, some editing is required to make “impossible” actions happen. Also, we often prefer motion to have an animated, rather than realistic, style, which must be added to motion capture via editing.
- Imperfections of reality: real motion isn’t perfect. For example, performers don’t exactly hit their marks and repetitive motions are not exactly cyclic. We often must edit motion capture data so it does exactly meet certain criteria.
- Change of intent: unfortunately, we can’t always predict what motion we will need, and even if we do, someone might change their mind as to what is desired after the fact.
- Secondary motion: motion capture data typically only provides the "gross" motion of the skeleton of a character. Other secondary motions, such as the movement of a character's clothes or soft tissue must be added using different tools. While motion editing most often refers to the problem of altering the existing primary data, increasingly tools will need to consider the issues in creating and manipulating the secondary motion as well.

The need to change motions is not unique to motion capture. In fact, the need to control the motions of objects is an essential part of any animation. Part of the uniqueness of animation is that it gives the creator control over the motion to better convey their message. If we only wanted to record some event for playback, we probably wouldn’t consider it animation, and could probably have used some other kind of recording technique like videotape or film.

1.1.1. The Challenges of Motion Capture Data Editing

To better understand the tasks and challenges of editing motion capture data, we consider a specific example. Suppose we have a motion of an angry female character walking across the room, and that this is a good, realistic motion that is “technically” correct. However, the story line of our animation changes such that the character is now furious, so we want to transform the motion such that it is angrier.

If our angry walk was created by an animator using keyframe tools, the animator could tweak the motion using the same tools used to create it. The animator might add a little more tilt of the

head, or make the footplants sharper, or whatever it takes to add some more angriness. The key is that the animator both understands where the angriness is in the motion, as well as understanding the specific details of this specific motion. For example, they may remember how they made the motion angry in the first place, or what details they added to the motion to make it fit with the character's personality. This familiarity is important: hopefully the animator can remember what made this a good motion so that the motion is tweaked, the good properties are retained.

At the technical level, the mathematical tools provided to the animator, e.g. adjusting poses, do not directly map to the task at hand. However, the animator most likely knows how the tools were applied to create the motion. If they were far thinking, they may have even designed the motion so that it would be easy to adjust, for example by putting the keyframes at convenient times.

If our angry walk were created by motion capture, the process of "angrification" would be different. The only way to use the same tools as used for creation for the adjustment problem would be to re-shoot the motion. Unfortunately, this may not be possible: we may no longer have access to the motion capture studio or the actress. At best, it is difficult, because it would require us to have the performer repeat the motion exactly, except for being angrier, recreating any other details we may have liked in the original.

If we had been far thinking, we might have captured the angrier motion during the initial session. However, this takes luck as well as planning: it is probably impossible to predict all possible changes that may have been needed. However, good planning is essential to making the editing process (and capture process) easier. This planning is also essential for other forms of animation too.

If re-shooting the motion is impractical, we are left with trying to use different tools for editing (compare this with keyframing where the same tools were available). There are two issues to be considered: first, the form of the motion capture data may not be convenient for editing; and second, because we did not create the animation by hand, we don't have the understanding of

how the motion "works," either artistically (what was done to make this motion angry?) or technically (which keyframes should be adjusted?).

We should note that these two challenges (form of the data and familiarity with the motion) are not unique to editing motion capture. In fact, any source of motion might suffer from these problems: even if the motion was created by keyframing, there is no guarantee that the creator made the data in a form convenient for editing, or remembered (or even documented) enough of the "why" of the motion. One important lesson is that planning during the creation process is extremely important to making editing easier later on: whether the creation process is keyframing or motion capture.

1.1.2. Types of Transformations

If motion editing is about changing aspects of a motion, a useful place to begin is to consider what types of things we might want to change. The short answer is anything. In general, a motion-editing task could be to change any aspect of the motion that we don't like.

Some kinds of changes are easier to describe than others. Many kinds describe abstract concepts or are unspecific. For example, we might want to alter the mood of a motion, or to make it more like a particular character would perform the motion. Generally, when we are using algorithmic techniques we need to be specific about the kinds of changes we make in motions (although, later we will explore a strategy which changes motions based on examples of the properties we are trying to achieve).

The kinds of things that might be important to preserve in a motion are also sometime difficult to describe. Exactly what makes a particular motion a particular action (walking, running, picking up a box), a certain mood (happy, angry, sad), or simply appealing to a director may be difficult to describe in sufficient detail to compute with.

Some types of transformations are more important for motion capture than for other types of animation. In fact, some of the transformations we may perform are designed to allow us to use motion capture data in ways that we use motion created in other methods. For instance, because

motion capture data records a real performance of a real person, it has several characteristics that hand-generated motions do not have. In order to use motion capture data like we use other forms of motion, we need tools to address these issues. For example:

- The motion capture performer may not have the same size or proportions as the character we are trying to animate. Therefore we need to adjust the motion so it fits on the new character. We call this problem *retargetting* a motion to a new character.
- When a performer tries to repeat a motion or return to a specific pose, they may not be able to achieve exactly the same state. Therefore, we cannot guarantee that two poses are exactly the same which makes transitioning between them more difficult. To address this, we need tools for making transitions between motions.
- Motion capture is saddled with the restrictions of reality, which are often inconvenient for the creation of animation. For example, a real actor can't jump as high as an imaginary superhero.

The science of motion editing provides us with a set of mathematical tools for describing properties of motions and manipulating them. As techniques improve, we get a richer set of things that we can do to motions. The art of motion editing creates the transformations beyond what we can describe mathematically. The challenge is to map from the high level goals to the specific details that the mathematical tools allow. Motion editing, like motion creation using keyframing, often requires the creativity of an artist to realize desired transformations.

Basic motion editing techniques come from the direct application of the mathematical tools. The operations provided by these basic techniques rarely map nicely to the high level operations we would like to perform: tools like frequency filtering that we will discuss in Section 1.6.3 or displacement-mapping that will will discuss in Section 1.6.5, do not directly implement changes such as "make sadder" or "stop the skating." As the tools evolve, however, we see the basic techniques serving as building blocks to achieve increasingly high-level operations. Therefore, the basic techniques are crucial to understand, both because they must be applied artfully to produce high-level effects today, and because they will serve as the basis for future technologies.

1.1.3. Properties in Motions

In much of our discussion we are intentionally vague about what the "properties" of motion are. The notion of a property of motion is critical for editing: they are the things that we are trying to change or preserve. Our vagueness comes from the fact that we may want to describe any type of change to a motion.

There are many statements that one might make about a motion. Outside of a few specific disciplines, like dance, our vocabulary for describing motion is typically not as well developed as for other types of domains like appearance. Therefore, discussions of motions are almost always filled with metaphor, or describe attributes that really describe something about the situation or the performer. For example, we often describe a motion by the mood or intent it conveys: an angry or sad walk. While it is a statement of the importance of motion that the movement of a character is so effective at conveying mood and other such properties, it also give us a problem that these things really are related to the actual low-level movements themselves.

When we look at a motion, there are many different "levels" of properties that we may consider. In our discussion, we use a vague notion of what level is. Roughly, we think of level as how abstract a property is. Some properties are very low level: at time 1, the leg is straight; at time 2 the toe touches the floor; or at all times, the elbow never bends backwards. Any of these properties can be easily observed in a motion without too much understanding or context, and can have concise mathematical descriptions. At the opposite end of the spectrum, there are high level concepts such as angry, regal, or like-Fred-Astaire. It is more difficult to characterize these in terms of what the mean to the movement. Motion concepts like "walk" or "jump" seem to fall somewhere in the middle of the spectrum.

One hypothesis that we will discuss later is that what seems high or low level, depends on the set of mathematical tools we use to look at the motion.

While it is hard to identify what makes a motion "an angry walk" (or any other such "high-level" property), many of these properties can be extremely easy to destroy. Changing one joint angle or position can turn a realistic motion into a useless teleportation.

1.1.4. The Unique Challenges of Motion Capture Data Editing

As we see in the example of Section 1.1.1, motion editing issues are not unique to motion capture. In fact, almost all of the methods described in this chapter can be applied to motion created with other methods, such as keyframing and simulation, as well. However, motion capture data creates some specific and unique issues that make it an extremely challenging task.

The example of the above example shows two key issues in editing motion capture data:

- The data is most certainly inconvenient for editing. Motion capture systems typically provide a pose for every sample or frame of the motion, not just at important instants in time. This means that a lot of data must be changed to make an edit. Also, motion capture data often uses skeletons parameterized in a mathematically convenient manner with strict hierarchies and measurements relative to a reference pose, whereas hand-made data often creates a skeleton that is more natural for manipulation.
- There is nothing but the data to describe the properties of the motion. There is little indication in the data to show what the important properties of the motion are, and what should be changed to effect the motion. Not only is there no animator familiar with the “why” of the motion, but the intent of the motion is unlikely to have ever been connected to the data. (Unless the performer knows things like “I bend my head 10 degrees to the left to make this action seem angrier.”)

These two issues may be summarized as "motion capture creates large amounts of unstructured data." The first techniques developed for editing motion capture data concentrate on the first issue. The idea being that if tools can be provided that make the data more convenient, then skilled artists could then use their creativity to address the higher level goals. As the tools improve, the second issue becomes more likely to be addressed.

Motion capture emphasizes a particular kind of editing operation where we start with a “basically good” motion, and try to make alterations that preserve much of what we started with. In general, the changes we make are probably small in the grand scheme of things: if we were going to completely change the motion, we probably wouldn’t have picked that as an initial

starting point. This is important for two reasons: one, because the kinds of operations we will want to perform on the motions will be quite different than when we were sculpting a motion from scratch; and two, it emphasizes that as we change motions, we often are interested in preserving other aspects of the motion as well. We will use the term motion transformation to refer to motion editing operations that attempt to change one good motion into another, somewhat different one.

Because motion capture almost never provides any secondary motion, the addition of secondary motion must be done after the capture process. For this chapter, however, we will focus on the problem of altering the existing primary motions.

Simulation methods for creating animation share many of the same properties as motion capture data. However, since most animation created using physical simulation has been targeted at honestly portraying the results of these methods, there has been less interest in applying editing methods to simulation results.

1.2. Representation of Poses and Motions

Given the challenge of altering a motion, we can now look at the details of actually realizing these kinds of changes. Central to changing motion is understanding how the actual data is represented. The specific details of the numbers that are stored to record the motion are significant because ultimately, these are the control knobs for the motion that must be changed. Our tools may present higher levels of abstraction to the user, but the software must ultimately map these effects into the underlying representation.

There are two aspects of a motion representation. The first is the representation of the character at any given instant in time. The second is how these specific instants are varied across times.

1.2.1. Representing Poses

Motion capture data gives a pose of the character at each instant in time. This pose consists of values for all of the characters parameters at a given instant. Choices in how we represent pose play a critical role in how editing is performed as these are the actual numbers that an editing operation must alter.

In editing motion capture data, we typically demand that the parts of our character do not change size, that is, that our character is made up of a linked set of rigid pieces. To a first approximation, this is the way to describe the most significant features of the motions of most limbed creatures such as humans. Mathematically, the definition of a rigid body is that the distance between any pair of points on the body does not change as the body moves. A rigid body's motion through space is limited to motions that preserve distances: it can only translate and rotate. Therefore, we can describe the configuration of the rigid body by a rotation and a translation.

For character animation, we often require the rigid pieces of our figure to remain connected. The end of one rigid "bone" always must stay attached to the next. Typically, we use a rigid skeleton to represent a character. That is, a character is made up of a set of rigid segments (often called bones) that are required to remain connected. A skeleton can be parameterized by the position and absolute orientation of one of the pieces, and the relative orientations between connected pieces (commonly called joint angles). The piece for which we specify position and absolute orientation for is commonly called the root.

The root segment of a skeleton is the only piece of a character that we specify the position and orientation in absolute (world) coordinates. Technically, any piece of the skeleton could potentially serve as the root. When animating by hand, it is sometimes useful to make the root of a character be some piece that needs to have its position specified, for example a foot, otherwise the position of this piece must be set by controlling the joint angles of the character. With motion capture data, we typically choose the root to be the "center" of the character (typically the pelvis).

The good news about a skeletal representation is that changing the parameter values will not destroy the form of the character. The arms and legs will remain connected no matter how we change the data. The bad news is that the hierarchical representation can be non-intuitive, creates dependencies between different parameters, and forces us to represent and manipulate 3D rotations. We will re-examine this tradeoff in a later section, and for now consider editing this skeletal data as it is the most common way that motion editing is done.

We should add that a skeletal representation can only record the configurations of the rigid pieces of a character. It does not allow us to represent the motion of a characters clothes or soft tissue moving. Skeletons are usually sufficient for the coarse level motions of characters, and almost always are all the data we get from a full-body motion capture system[1]. The additional details of secondary motion are typically layered on top of the skeletal motion to produce a final result.

1.2.1.1. Representing Rotations

To represent the configuration of a rigid body, we must provide information about the position of a point on the body, and the orientation of the body. Orientation is expressed as a rotation relative to some other coordinate system, either a fixed "world" coordinate system (in which case the rotation gives us an absolute rotation), or some other moving object. The latter is commonly used in skeletons to express the rotation between two connected joints.

The representation of a 3D rotation is a difficult problem. A full discussion is beyond the scope of this chapter. The basic issue is that it is impossible to represent a 3D rotation using an unconstrained set of real numbers without having a singularity. In other words, there are many possible ways to represent a 3D rotation using a set of numbers, such as Euler Angles, unit Quaternions, and rotation matrices, however, all of these are plagued by difficulties. If there are fewer than 3 numbers, than all rotations cannot be represented, if there are only 3 numbers then there will be singularities: a mathematical condition that means that for certain values of the

[1] Facial motion is generally non-rigid, therefore facial motion capture systems do not use skeletal representations.

parameters, there may be movements that cannot be achieved. If there are more than 3 numbers, we must put limitations (for example, that the magnitude of the 4 numbers of a Quaternion is 1) on the numbers to make sure that they represent a rotation.

Fortunately, in working with human motion, there are few joints that require the full range of rotations. Many joints, such as the elbows and knees, can be represented with one or two degrees of freedom. However, for the 3 degree of freedom joints such as the shoulders and hips, a representation for a 3D rotation must be selected.

Because there cannot be a perfect representation for a 3D rotation, any method we choose is subject to tradeoffs. For a good discussion of these tradeoffs see Sebastian Grassia's excellent tutorial on Exponential Maps (Grassia 99) that discusses tradeoffs among several representations for many different tasks. Motion editing provides a different set of needs for a rotation representation, so we briefly describe the use of some of the more common representations.

A basic representation for a rotation is a rotation matrix. Usually, we convert any representation that we use to a matrix in order to feed it to a graphics library, so we might consider simply storing and manipulating it. The obvious problem is that it requires 9 numbers to encode the 3 degrees of freedom of the rotation. What is worse, is that only a special set of the possible values (those matrices that are orthonormal) actually represent a rotation. Given a set of 9 values, it is unlikely to be a rotation. If we begin with a rotation matrix and change any one value, we are certain to get a matrix that is not a rotation.

Some of the operations that we must perform on rotations can be performed easily with rotation matrices. For example, rotations can be composed by multiplying matrices, and inverted by inverting the matrices. Other important operations on rotations, such as interpolation, are very difficult.

Euler Angles

Mathematician Leonhard Euler made the observation that any rotation in 3D space could be constructed as a sequence of 3 individual rotations around the coordinate axes. This leads to a

compact representation for a rotation commonly called Euler Angles. For example, any rotation can be expressed as a rotation around the X axis, followed by a rotation around the Y axis, followed by a rotation around the Z axis, as shown in Figure 1. Since rotation around a single axis can be easily measured by a single parameter, we can represent and rotation in 3 dimensions by three parameters.

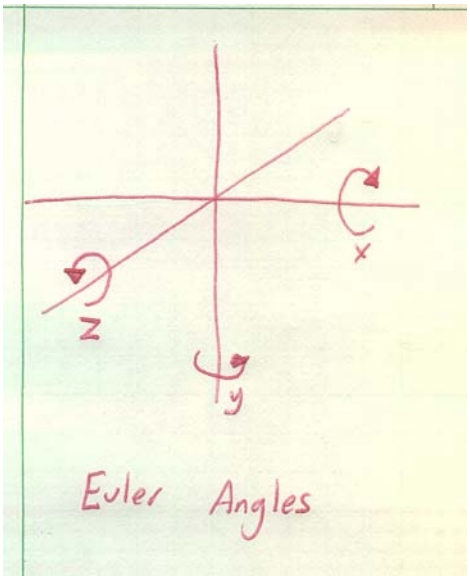


Figure 1: XYZ Euler Angles

In fact, any set of three orthogonal axes can work. In our example, we used the convention XYZ, but we could have just as easily picked ZXY or even ZXZ[2]. What is important is that we know what convention that we are using, as different orders of rotations lead to different interpretations of the same parameters. Another detail is whether the axes are named in the coordinate frame of the observer, or of the rotating body. Either decision works, but we must be consistent. In computer graphics colloquialisms, the term "Euler Angles" is often used for any

[2] While it may seem that the Y axis is forgotten with the ZXZ set of angles, it is not. A rotation about the Y axis can be achieved by first rotating the object around the Z axis, such that the original Y axis is now along the X axis, rotating around the X axis, and then rotation back around the Z axis to undo the first transformation.

representation of a rotation as 3 rotations around the coordinate axes, although technically, Euler Angles refer to a specific convention (ZXZ in the local coordinate system of the object) [Goldstein]

Euler angles have the advantage that any 3 numbers represent a rotation. So for editing purposes, we can change the numbers however we like and can be sure to have a rotation. Euler Angles seem like they give us 3 independent controls for controlling a rotation. Unfortunately, the controls are not independent. Changing one of the angles alters the meaning of the subsequent angles. This is significant because it means that many important operations cannot be performed on them.

Another problem with Euler Angles is that there is no simple mathematical method for composing two rotations. x,y,z * a,b,c is not x+a, y+b, z+c. In fact, there is no simple, closed form method. A related problem is that it is difficult to interpolate Euler Angles in any meaningful kind of way. Halfway between two Euler Angles in terms of the parameters is not necessarily "in-between" the two rotations. This makes the creation of nice rotations difficult. A feature of Euler angles is that performing simple mathematical operations, such as linear interpolation, still leads to a rotation. The problem is that these simple operations may not necessarily provide the desired rotation.

The best known complaint about Euler Angles is that they exhibit what is called "gimbal lock." Gimbal lock is a phenomenon that occurs because of the singularities in the mapping between Euler Angles and rotations. Simply put, for some values of the angles, the other two angles represent the same rotation, as illustrated in Figure 2. The more general form of this problem is that for any rotation, there are many ways to represent it. And that there is not necessarily a connection between how close the numbers are and how similar the rotations are.

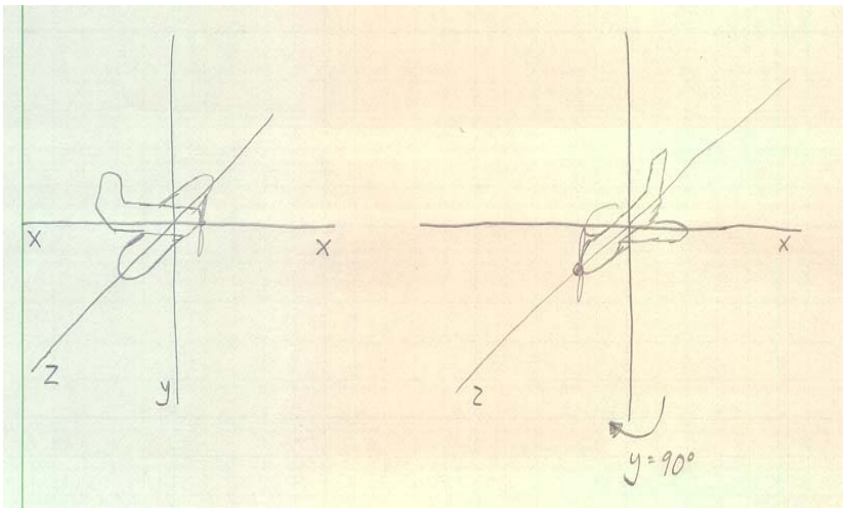


Figure 2: Euler Angle Singularity: With XYZ Euler Angles, a 90-degree rotation about the Y axis causes the original X axis and the final Z axis to align, and therefore represent the same rotation. In the example, rotating the plane 90 degrees about the Y axis causes what was the original X axis (the forward direction of the plane) to point along the Z axis. Therefore, both the X and Z rotations have exactly the same effect: the "roll" the plane around its center axis. When Y is 90 degrees (so the plane is turned left or right), there is no way for the XYZ Euler angles to describe a rotation that changes the attitude of the plane (pointing the nose up or down) which requires a rotation around what was originally the Z axis.

Despite these problems, Euler Angles are the most commonly used method for representing angles in performing motion editing. Some of this is historical: Euler Angles were well known and widely used by the graphics community long before practical alternatives were “discovered.” Most of the alternatives also require a greater degree of mathematical sophistication to implement.

With care, Euler Angles generally can be made to work in practice for motion capture data editing, despite their problems. Foremost, Euler Angles work fine when only one of the angles is being used, which is the case with most of the joints on a human figure. Euler Angles generally work when the first or last angle doesn’t change much. Also, with motion capture data, the differences between successive frames’ rotations are small, and in small angle approximation, many of the problems of Euler Angles have small effect.

The gimbal locking problem with Euler Angles can be avoided by choosing the rotation order such that the singularity occurs in a configuration that is unlikely to occur. However, if an object has a wide range of motion, the singularities cannot be avoided. One scheme is to switch between different parameterizations (for example switching from XYZ to ZXZ) when a singularity is approached as the singularities occur in different places for different representations. Unfortunately, such dynamic reparameterizations can be difficult to implement in practice

.

Axis/Angle

An axis angle representation of a 3D orientation makes a different use of Euler's observations about angles. Rather than recording three rotations about fixed axes, the Axis/Angle represenation measures a single rotation about a single axis. With this representation, four numbers (a scalar and a 3D vector) are required to encode an orientation. Any rotation can be encoded as a continuum of axis angle values, as the length of the axis vector does not matter.

Axis/Angle representations have the advantage that the intuition for understanding or describing an orientation is simple. However, because of their redundancy, and their inability to be interpolated or composed, they are difficult to use for motion editing applications. The following two representations are closely related, but do not suffer from as many drawbacks.

Quaternions

The most common alternative to the Euler Angles parameterization is the unit Quaternion, which were introduced to the graphics community by Shoemake (Shoemake 85). A Quaternion is a set of four numbers that is interpreted in a particular way. Quaternions have a rich history, and a number of applications beyond representing rotations. More generally, they may be though of as a multi-dimensional extension of complex numbers.

Like with rotation matrices, only Quaternions with a specific property, namely unit magnitude, map to rotations. However, Quaternions are far less redundant than matrices, and the constraint

can easily be enforced to convert any set of 4 numbers to a unit Quaternion. Changing any one particular value of a unit quaternion does create a non-unit quaternion.

The advantage to Quaternions is that a "calculus" of basic operations, such as multiplication and inversion, has been defined for them, and these operations correspond to useful operations on rotations. For example, to compose two rotations, we multiply the two corresponding Quaternions. The quaternion operations preserve the unit magnitude of the Quaternions, so if we start with a unit quaternion rotation and apply quaternion interpolation and multiplication, we will end up with a unit quaternion.

Interpolating Quaternions involves the use of special methods that preserve the Quaternion properties. One advantage to using Quaternions is that the most basic spherical linear interpolation (commonly called SLERP) provides results that are both mathematically well-founded and visually appealing. Recently, methods for creating spline curves in the Quaternion space have been developed, such as the one of Kim and Shin (Kim 95) or the rational formulation of Johnstone and Williams (Johnstone 95).

The main disadvantage to using Quaternions for motion editing is that they are not intuitive parameters. The 4 numbers have little intuitive meaning, and cannot be directly manipulated by the user. In general, Quaternions are a nice method for representing rotations inside of a system, but not a good interface to present to a user. Also, not all operations have been defined on Quaternions (for example, they cannot be added). And, while projection is a useful method for correcting for error accumulation, simply manipulating the 4 numbers of a Quaternions and then trying to convert back to a unit quaternion is not necessarily a mathematically meaningful operation. In short, quaternions are not the panacea pronounced by their first "discoverers," they simply have a different set of advantages and disadvantages than the more common Euler Angles.

Exponential Maps

Exponential Maps are another representation for rotations that have a number of desirable properties for performing many of the computations needed in working with articulated figure

motion. The use of Exponential Maps in computer animation has only begun recently, and has not yet spread widely. For an excellent tutorial on the use of exponential maps for animation, see the paper by Grassia (Grassia 99).

Like Euler Angles, the Exponential Map reprsentation uses three numbers to describe a rotation. This means that it must have a singularity. However, unlike Euler Angles, it is simple to use dynamic reparameterization to keep away from the singular regions of the space. Unfortunately, many of the other difficulties with Euler Angles, such as lacking a meaningful interpretation for interpolation and lacking a simple method for composition, also seem to be apply. These issues may be resolved as Exponential Maps are explored more closely in the context of animation problems.

1.2.1.2. Are Skeletons a Given?

A hierarchical (e.g. skeletal) representation for motion is almost an assumption for motion editing. While some of the techniques we will discuss apply regardless of representation, they are almost always applied to the joint angle representation of the figure. In this section, we reconsider this choice. Our goal is not to advocate the alternative, but rather, to help provide some insights into the skeletal representation.

The alternative to a hierarchical representation of a figure would be a non-hierarchical one. That is, rather than describing a pose by the position of one part of the figure and the relative orientations of all the other parts, we describe each part independently of the others. For example, we might simple represent the position of each joint in space, or the position and orientation of each limb segment, or even the positions of many points scattered over the body (like the positions of optical markers). What is significant about all of these representations is that each piece is independent from the others.

The biggest drawback of the non-hierarchical representation methods is that a change in the parameters may destroy the connectivity or size of the skeleton. Changing any individual

parameter will probably make the configuration “invalid” unless extreme care is taken: bones may become disconnected or stretched.

One way to look at this problem is that the non-hierarchical representation has more parameters: we are allowed to change not only what we want to be able to change, but also, things that we may not want to change.

For a simple example, consider an object that can only rotate around the center in 2d, so the position of one of its ends and its length are fixed. The equivalent to a hierarchical representation would be to store the angle of the object. Alternatively, we might store the position of the end of the object. These two representations are illustrated in Figure 3.

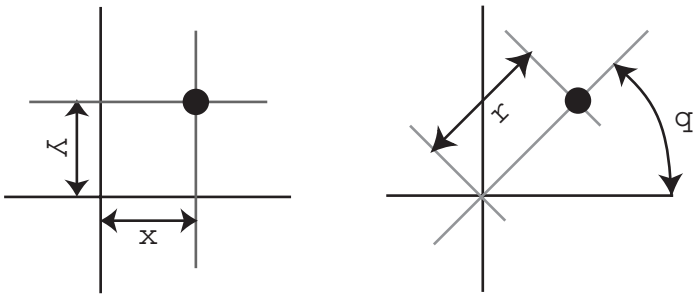


Figure 3: Two representations of a point in 2d.

With the more compact representation, any value that we give to the parameter will be legal. With the 2-parameter representation, only a certain set of values will maintain the length of the object. Another way to look at this is that both parameterizations have the same number of parameters, except that with some parameterizations it’s more obvious which combination of parameters are valid. For the point example, we might represent the point in rectangular (x, y) or polar (distance, orientation) coordinates. In the later, it is obvious how to keep the object at the same length - just don’t change the distance. Keeping the length with the rectangular coordinates requires us to restrict the values of x and y such that the equation x^2+y^2=d^2 is true.

With different parameterizations, different limitations are easy to create. For example, if we wanted to restrict the end of the object to always be on the floor (but not restrict the size of the object), it would be easy in the rectangular coordinates, but difficult in the polar coordinates.

The same view can be applied to representations of pose. The hierarchical representation makes it easy to constrain that the lengths of the bones stay constant and that the bones stay connected. However, an independent representation makes it easier to specify the position of an individual piece. With the hierarchical representation, it is possible to specify the position of the endpoint, although it requires solving a more complex equation including a number of variables. This process is commonly called inverse kinematics.

In general, animation systems have preferred to keep the bones connected and of constant length, even though this makes it more difficult to specify the positions of the end effectors. Because of this, we often see animation where the foot skates across the floor or floats above the floor, but rarely see animation where the limbs become stretched or disconnected. Inverse kinematics, the mathematical tool to control the positions of end-effectors in a hierarchical representation, is a much better developed tool than what would be required to re-assemble a broken character.

Hierarchical representations do have problems: they require us to deal with interlocked representations and to compute over angles. To specify the positions of end effectors, we must solve a complex inverse kinematics problem. In contrast, a non-hierarchical representation has independent parameters and the parameters are positions (which are much easier to deal with than angles). However, in order to maintain connectivity, we must solve a large number of constraints simultaneously. Few existing tools do this.

1.2.2. Representing Motions

A motion is a function that converts from a time to a pose. We can think of it as a black box that can be asked questions of the form “what is the pose at time t.”

$$p = f(t)$$

Typically, we will only ask for the pose at times corresponding to the frame times, unless we are supersampling to create motion blur.

Even though we only ask for values of t that are discrete, we often have motions as continuous functions, that is, that we can ask for any real-values for time t. For example, with keyframing, we create a piecewise polynomial curves (typically cubic) that interpolate the key points.

With motion capture, we know the values of the parameters only at discrete intervals. Technically, we do not know what happens in between these samples, and make some assumptions that we haven't missed anything important. (see appendix for a discussion of the theory and ramifications of this). We call such a representation of a motion a sampled representation.

Despite the fact that samples are close together, we sometimes do need to examine the value in between two samples by interpolating them. This can be problematic with Euler Angles: we can have data that seems perfectly fine (because each sample is), but is not interpolatable. These issues can show up whenever we try to manipulate the data. Non-interpolatability often occurs when there are multiple, equivalent values that represent the same thing, for example, 180 and -180 degree angles.

One way to look at motions is as parameter curves, that is, graphing the values of parameters, as in Figure 4. These simple curves have become a mainstay of animation systems. Despite the fact that the curve itself may represent an angular value, and therefore have no direct meaning, users have grown accustomed to interacting with them.

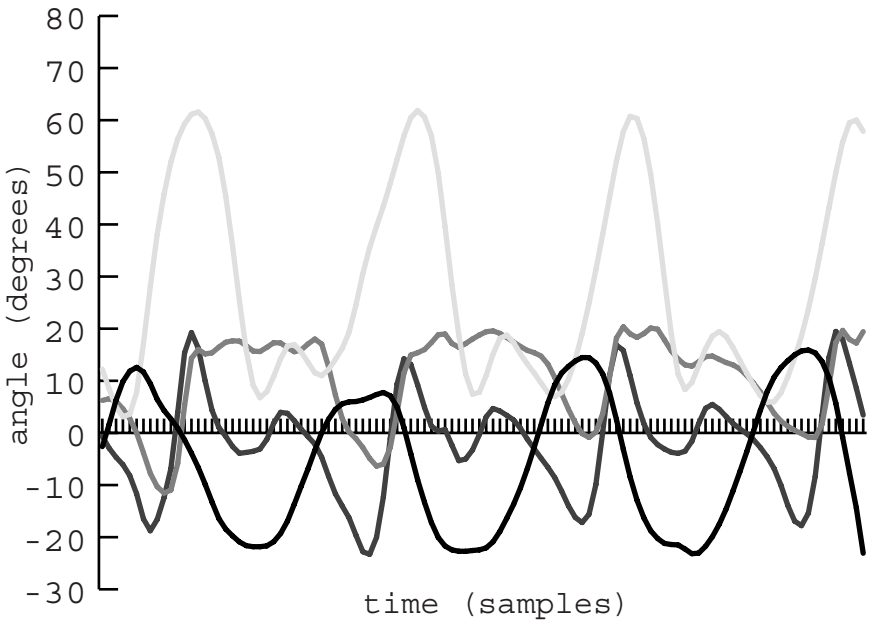


Figure 4: Motion Curves: 4 curves taken from a walking motion. Each curve on the graph represents a different joint on the character.

Motion capture represents a motion as a dense set of data: there is a number (sample) for every instant in time. A by-product of this is that there are a lot of numbers. This provides an opportunity for the motion capture data to represent more fine details than sparser keyframes can (e.g. the denser samples allow higher frequencies to be represented). The down side is that motion capture data has lots of data that must be manipulated in editing. When dealing with a dense representation of motion data, changing a single value only alters the pose of the character at that one instant in time.

1.2.3. Editing's Dependence on Representation

The way that a curve is represented has an important impact on how it is edited. Consider a curve that is defined by interpolating a number of points. Depending on how many points there are, the effects of changing any given point changes.

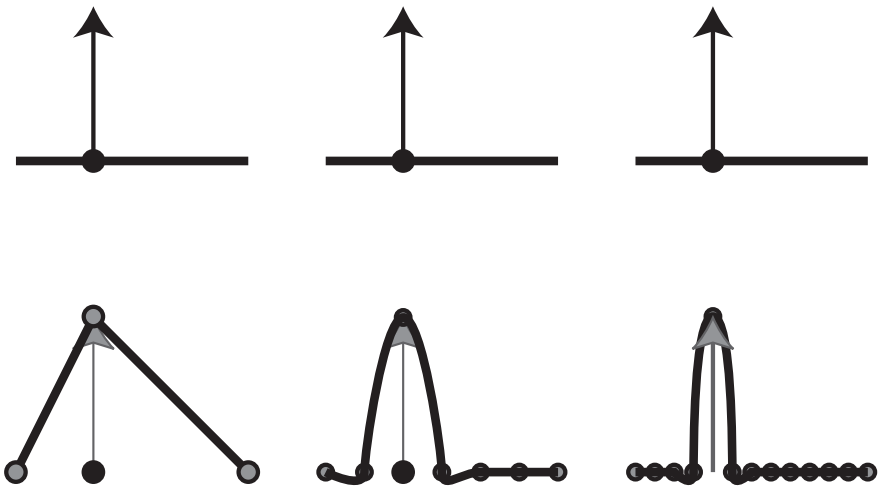


Figure 5 Changing one control point of an interpolating curve has a different meaning depending on how the curve was represented. (NOTE: word numbering seems to skip Fig #5)

The above example in Figure 5 shows that the kinds of changes in a curve depend on how the curve is represented. This suggests that depending on the kind of changes that we want to make, a different type of representation might be most desirable. Unfortunately, we do not get total freedom in representation. Also, we cannot necessarily shift representation for each edit we may want to do as each may require a different representation to be accomplished effectively.

Motion capture data almost always provides a specific form of representation: a dense set of samples of the values of the parameters. This representation is inconvenient for many kinds of manipulations.

1.3. Cleaning Motion Data

One type of editing operation that we often must consider is clean-up, the process of making the data that we received from our motion capture process more accurately reflect the performance we were trying to capture. Because clean-up is very closely tied to the problems that

occur in actually obtaining data, and therefore, very much dependent on how the data was obtained. Therefore, we limit our discussion here to a brief, high-level discussion of the problem.

The clean-up process is a specific type of change (or edit) to motion. We want to change the motion from not reflecting what happened to representing what did. What makes the clean-up problem unique is that since we don't have an exact record of what happened (if so, we could use that instead of our dirty data), its difficult to know when our data differs from what happens, and even more difficult to know what to replace it with. In fact, the very name "clean-up" implies this problem: in order to do cleaning, we must be able to separate out the dirt.

In general, clean-up procedures attempt to identify elements of a motion that are more likely to be created by a failure mode of the capture process than by the performer, and to make an educated guess as to what really happened. Sometimes the former problem is easy: for example, in optical motion capture data, if a marker is obscured we often get an indication that there is no data; or, if we see a motion contain an impossible occurrence, such as a foot passing through the floor, we can identify a problem to correct. Other problems are more difficult to identify. For instance, if we see a character shake slightly, is this sensor noise or did they have a nervous twitch? In general, we must resort to heuristics (general rules) to both identify the dirt in data as well as to create a replacement for it.

One of the most common heuristics used to identify noise in motion data is that in general, things do not change very quickly. If we see something changing too quickly, it most likely was caused by a misreading of the sensor (noise), than by the actual performer. In signal processing terminology (see the appendix), motion tends not to have much high frequency content, and the noise we encounter often does. This suggests an easy solution: filter the high frequencies out of the motion to remove noise. Some basic methodologies for implementing this are described in the Appendix.

The problem is that the same high frequencies that create the annoying noise are the ones that give crispness to the motion when it does make abrupt changes: the snap in a karate kick, or the impact of two objects. As seen in FIGURE, while filtering can help reduce the noise, it also

reduces the crispness in a motion, leading to a sometimes objectionable look, all too common in computer animation. Low-pass filtering should be applied carefully as to not create an unwanted look.

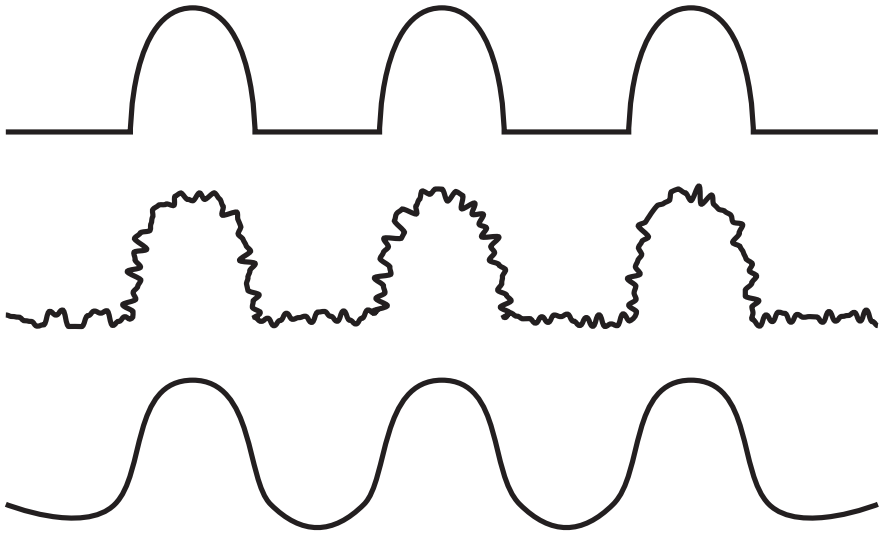


Figure 6: Low-pass filtering a motion may remove high-frequency noise, however, it also removes the "crispness." Top: the original motion, Middle: motion with noise added, Bottom: result of low-pass filtering the middle.

Cleanup editing is much like other editing tasks: we aim to make a change that preserves as much of what is good about the original as possible.

1.4. Types of Motion Editing Techniques

Now that we have motion curves, the problem of motion editing is simply changing these curves to meet our new needs, while preserving what we don't want to change.

The fundamental challenge is that the properties of the motion are not obvious from the curves themselves. Just looking at the data doesn't necessarily give us any indication of "where" the various high-level properties are in the data. In looking at the curves, it is impossible to say that there is a specific piece that creates the angriness, the realism, the walk, the particular actress.

While we may like to change these high-level properties, ultimately what need to control are the low level details of the characters pose at each instant.

Even “middle level” properties of the motions come from the inter-related behaviors of a large number of parameters. Something as simple as the foot touching the floor is determined by all of the parameters that effect the foot (the position of the root and all of the joint angles in the leg). Fortunately, many of these parameters do have concise mathematical descriptions that allow us to identify them. For example, we can compute the position of the foot based on the number of parameters and determine whether or not it is passing through the floor.

While the high-level properties in a motion may be hard to identify in data, and hard to create, they are very easy to destroy. Changing just one number in the motion data of our realistic walking motion can easily destroy the realism, change the motion from walking to teleporting, or just add an annoying twitch.

There are several basic strategies that motion editing tools use to help bridge the gap between the high level descriptions we would like to use, and the low level details we must control:

- We can describe desired features by example. For instance, we may not be able to say why a motion is “angry” but we can provide a motion that is.
- We can attempt to control and maintain certain mathematical properties in the motion with hopes that preserving these properties can preserve the higher-level motion properties.
- We can identify and control details in the motion, for example footplants, and make sure that these are maintained (or allow changes to them to be specified).
- We can generally try to make editing the basic details more convenient so that it is easier to make the changes to the parameters. Specifically, changing individual frames is usually impractical with the densely sampled motion data. Almost any change would require altering a large number of individual data elements.

In practice, motion editing techniques for motion capture data must address the pragmatic issues of handling the large amount of unstructured data. There are two general approaches to this:

1. We can convert the data to a more convenient representation. For example, we might try to create keyframe data from the motion capture data, allowing the use of more traditional tools for editing the motion. These methods are commonly called key-reduction methods because their core must be a process to reduce the large number of motion capture data points to a more manageable number of keyframe poses.

2. We can create methods that describe changes in ways that are independent of the underlying representation of the curves. Such methods often fall under the name of signal processing approaches because they tend to view motions more abstractly as signals. The theory of signal processing (see appendix) allows us to discuss signals (which motion curves are) independently of what they represent or how they are represented. Signal processing provides methods that can be described independently of the fact that we have the sampled curve representation, and can be implemented directly on the sampled representations.

In the following two sections, we describe each of these general approaches. The chapter then concludes with a discussion of a spacetime constraints approach to motion editing which attempts to gain the benefits of both key reduction and signal processing.

1.5. Key Reduction

Since motion capture data is so much more difficult to edit than keyframe motion data, an obvious approach is to convert the motion to the more editable form. Such an approach is called key reduction since it tries to reduce the number of specified frames. The basic idea is to find curves that have simple representations, but fit the original data. Mathematically, this problem is called curve fitting or regression.

When fitting a curve to data, the curve may not be able to exactly fit the data. In general, there is a tradeoff between the complexity of the curve and how expressive it is. If we want to have a simpler curve, it is less likely to be able to fit our data closely. Usually, we choose a form of a curve for simplicity, and then try to find the values of its parameters that get as close to the data as possible.

There are a number of different ways to measure the closeness of a curve to the data. The most common measure is sum of squares. Other measures attempt to discard elements of the data that seem irrelevant or erroneous, with the intent of making a better fit to a portion of the data. These methods are called robust statistics. An example is shown in Figure 7.

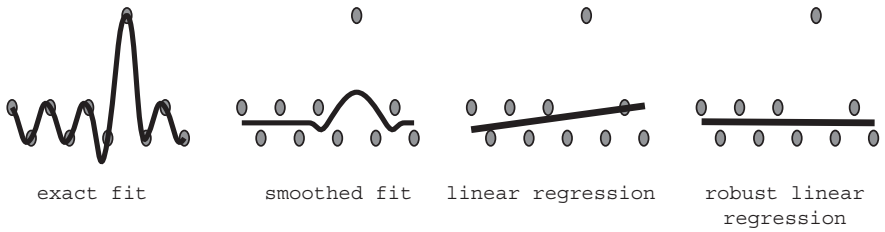


Figure 7: Different ways to fit a curve to a set of data, trading off closeness to the data vs. Compactness of representation.

In doing fitting, there is a tradeoff between the closeness of the fit and the expressiveness of curve. For example, suppose we want to fit a piecewise cubic spline (like we use for keyframing) to some motion data. If we allow ourselves to have a control point at every time step, we can exactly represent the motion data, however, we will have a curve that is as difficult to manipulate as the raw data itself. At a different extreme, we could have a curve with only a few control points that would encode the basic shape of the curve, but miss out on the smaller details.

Controlling the complexity/closeness tradeoff becomes an artistic decision when using curve fitting for motion data. It might be acceptable to lose some of the smaller details in the motion in order to make it easier to edit. However, these small details are often what give motion capture data its character.

While automatic key reduction addresses part of the difficulty of motion data by putting it into a more editable form, it does not address the issue of lack of familiarity with the motion. While the automatic process may be able to place a minimal number of keys at optimal spacing, what is best mathematically may not be useful from the standpoint of a person trying to understand the data.

Kinetix's Character Studio is a commercial product that performs key reduction on motion capture data to make it easier to edit. The program has a sophisticated key reduction algorithm that attempts to identify specific features in the motion, such as footplants, and uses these as keys in order to create a more easily editable representation.

1.6. Motion Signal Processing

By using the theory and methods of signal processing (see the appendix for a brief introduction), we gain three things for motion editing:

- We get a set of analytical tools for thinking about motions.
- We get a vocabulary for discussing motions and alterations to them.
- We get a set of operations to apply on motions.

The parameter curves that we use to describe motions are certainly signals in the standard sense of the term. If we can treat each parameter as a separate signal, the traditional methods of signal processing apply. However, each of the parameters is not truly independent. A character's movement is created by coordinated changes in a combination of its parameters, so considering the parameters individually has limitations. Also, there may be tight couplings between parameters that cannot be altered. For example, if 4 parameters represent a unit quaternion rotation, changing one parameter will destroy the unit magnitude property. For this reason, motion editing is typically done on Euler Angle representations.

With signal processing, we view our curve as a black box

$$p = f(t),$$

where p is the value of the parameter of interest, and f is a function that maps from times to parameter values. We do not consider what the function f is that generates the value. Without the ability to change f, we are stuck with two possibilities for altering the outcome:

- We can alter the t that gets "fed into" the black box.
- We can change the values that come out of the black box.

The first category we call time transformation. These are often very basic operations that are common to animation. We begin with them to show how they lead to representation independent editing of motion curves.

Technically, the signal processing methods that we will apply to motion signals will really apply to sampled representations of them. This is not too much of a restriction because any signal can be converted to this form by sampling.

The term Motion Signal Processing generally originates in the paper of that name, published in SIGGRAPH '95 by Bruderlin and Williams (Bruderlin 95). In this paper, the authors demonstrate how a variety of standard signal processing tools could be used for interesting effects on motions. The techniques surveyed in this paper were not all new to it.

1.6.1. Motion Frequency Content

Before discssing specific methods for performing signal processing operations on motions, we digress to review how a fundamental concept in signal processing applies to the analysis of motions. This concept, frequency domain analysis, is at the core of many signal processing methods and provides an important vocabulary for discussing motions in a mathematical way. Frequency domain analysis is reviewed in the Appendix.

When we look at a motion curve, it is hard to see where the properties of the motion lie. What about a particular motion curve makes it a good motion? A walking motion? A sad motion?

Part of the problem is that we simply might not be looking at the motion the right way. Or, that our "view" of the motion lets us see certain properties easily, but not others. By taking a different "view" of the motion curves, we might see a different set of properties.

Signal processing provides a number of different views for signals (like the motion curves). The standard view, like the curves we have been looking at, is called the time-domain. Another common one is the frequency domain (see the Appendix). We should emphasize that the frequency domain merely gives us a different view of the same data.

The time domain representation tells us what is happening at a given instant. With a motion, it is useful for telling us what is happening at a particular time. The frequency domain representation provides us with a view of the kinds of changes the signal makes. For example, the time domain representation of a motion signal might tell us that at the 45^{th} frame of the animation, the angle is 30 degrees, while the frequency domain representation of a motion may show us that the angle makes some abrupt changes at some time, but generally makes gradual changes.

Like the time-domain view, a frequency-based view of a motion is a mathematical tool, and does not necessarily correspond to any particular qualities in a motion. Some researchers, such as Unuma et. Al. (Unuma '95), have postulated that different frequencies are more pronounced in some kinds of motions, or that some frequency information directly corresponds to emotional content of motions. More generally, the low frequency components of a motion convey the basic movement of a motion while the high frequencies give the details. It has also been speculated, for example by Witkin and Popovic (Wikin '95) that part of what gives motion captured motion is unique character is the existence of these high-frequency details.

One place where the frequency view of motions is most useful is in understanding the implications of sampling. The frequency domain gives us a clear picture of what information will be lost by expressing a signal as a discrete set of samples, and will provide us with a set of tools

to avoid certain types of problems that can occur in sampling. Some specific examples will be described in the next section.

1.6.2. Simple Time Transformations

The general idea of a time transformation is that we still use the same motion curve, we just alter the time that we check. That is, we define some way of computing a new time given a time, and use this computed time to check what value the motion curve has when asked

$$p = f(g(t)) \quad g: t \rightarrow t$$

The most basic time transformation is to use addition of a constant

$$p = f(t + d).$$

In signal processing terminology, this is called a time or phase shift, or a delay. While this is obvious and trivial to implement, it is immensely useful for working with motions as it allows us to change when motions occur. It does, however bring up issues about what happens at "the ends" of motions: e.g. if we have a 3 second motion, and we delay it's start by 2 seconds, what does the character do for the first 2 seconds or after the 5 second mark? Often, we will have the character perform some other motion before and after. This brings up the issue of making transitions between motions. We will discuss transitions later in this chapter.

1.6.2.1. Time Scaling

Another simple example of a time transformation is speeding up or slowing down a motion. This can be implemented quite easily as:

$$p = f(k * t)$$

where k is some scaling factor. This operation allows us to speed up or slow down a motion.

The simplicity of the time scaling example makes it a good illustration of the issues in dealing with motion capture data. First, we should notice that the way we have described the time scaling is independent of how the motion is represented. We simply change which time we "look up" when we ask the "black box" for the value. Despite the simplicity of this change, it does change properties of the motion. For example, if we scale time by 4 (effectively quadrupling the rate of speed), we create a transformation that destroys many properties of the motion, such as physical realism.

While the time scaling may be defined independently of how the motion curve is defined and used, its correct implementation does require us to deal with the underlying representation of the motion curve. If the motion is initially represented as samples (as motion capture data is), then the process of time scaling is effectively a resampling of the data. In another view is that amount of time the motion takes (e.g. the motion takes 1 unit of time), we are really just changing the number of samples of the motion.

In performing resampling, there are two cases we must handle. First, we must be able to determine what happens in between the individual samples, especially if we are slowing down the motion. Second, we must handle properly throwing away information that will be lost when we speed up a motion, since we will have fewer samples with which to encode things. The former case can be easily handled by interpolation. Most often, linear or cubic interpolation is used for resampling.

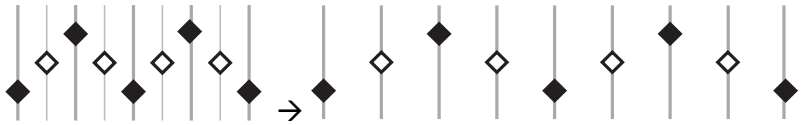


Figure 8: Resampling with linear interpolation.

The latter case may require considerable care to do correctly. For a simple view of the problem, consider a simple motion that oscillates every other frame.

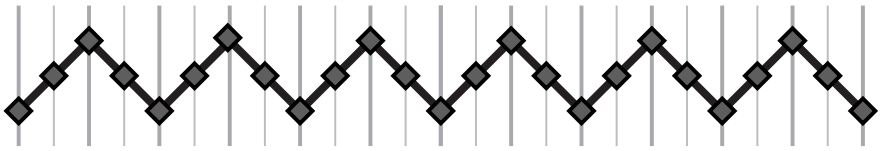


Figure 9: A simple motion

The obvious way to resample is to simply ask the black box of our motion for the new times. If we normally would have looked at samples in order (0,1,2,...), we would instead look at sample (0,n,2*n,3*n,...) where n is the change of speed factor.

If we speed the example motion up by a factor of 2 by simply looking at every other sample, we get the expected behavior of the motion being three twice as fast.

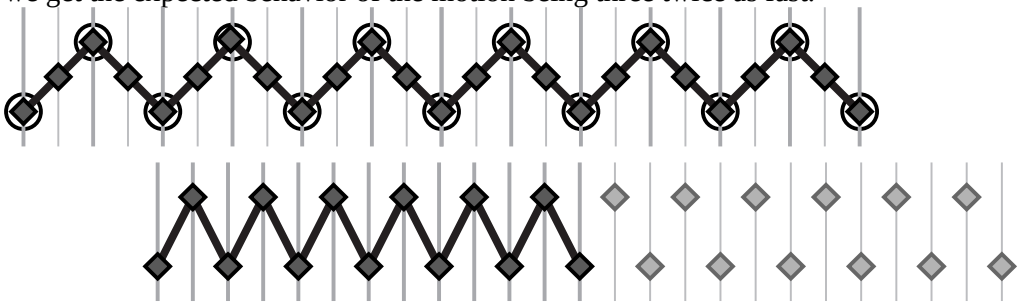


Figure 10: Speeding a motion up by a factor of two by taking every other sample.

However, if we speed up by a factor of 3, something unexpected happens

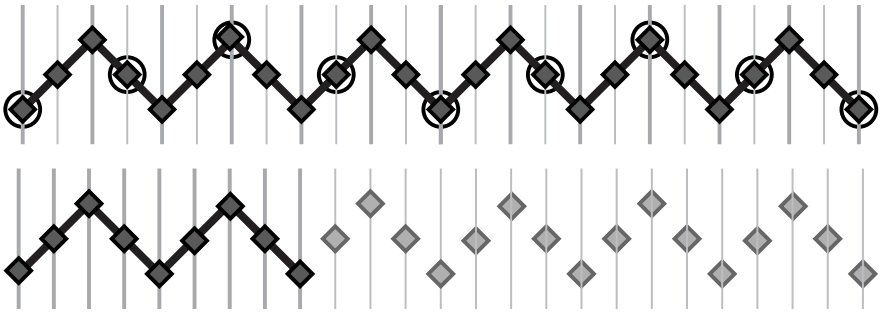


Figure 11: Speeding a motion up by a factor of three may not speed the motion up at all!

Rather than speeding up, the motion has its original speed! If we speed the motion up by a factor of four, the motion stops alternating altogether.

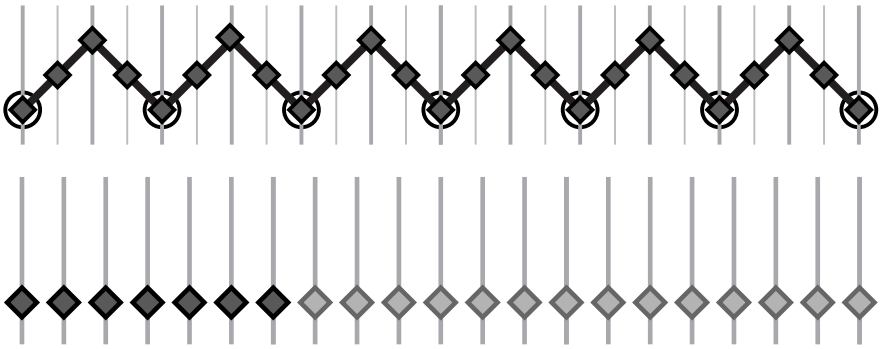


Figure 12: Speeding a motion up by a factor of 3.

Speeding up by a factor of three and a half (which would require us to interpolate to get values in-between samples) would actually slow the motion down.

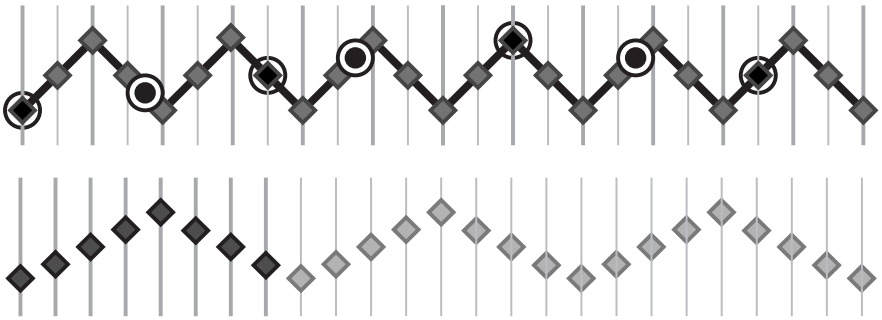


Figure 13: Speeding up by a factor of three and a half slows the motion down

In general, when we speed up a motion we often get some undesirable results depending on how much we speed up the sampling. This phenomenon is called aliasing. Basically, what is happening is that the motion changes too quickly to be adequately captured by the sampling. It therefore appears as a slower motion than what is really happening (in some cases, it slows all the way down to 0).

One way to look at this problem is that speeding up a motion requires us to throw away some of the information. At the same sampling rate, a motion that is speeded up by a factor of n has 1/n as many samples. Clearly this smaller number of samples carries less information about the motion than the original. Aliasing occurs when the information that gets thrown away is significant. To avoid aliasing, we must first alter the original signal to remove any portions of the signal that cannot be represented in the smaller number of samples. This analysis is straightforward using the concepts of frequency, described in the appendix.

The aliasing problem can occur with whatever representation we use for the motion signal. Ultimately, we will sample the motion in order to produce the images for the animation. If we had continuous representations for our motion curves, we could rely on performing the proper sampling visually by using motion blur.

Technically, the correct way to perform a resampling would be to construct as good a continuous representation of the signal and then to do proper sampling (see the appendix) on that to create the new sampled representation. In practice, we implement this by pre-filtering the

initial signal to remove any frequencies too high to be represented at the new sampling rate, and then perform the "naive" sampling. This filtering can be achieved by performing a discrete convolution of the signal with an approximation to a low-pass filter kernel, such as 1/4 [1 2 1].

Any time we change the timing of a motion, we must be careful about how resampling is performed. Fortunately, the issues in resampling a motion are the same as resampling any signal, so methods used from warping images or processing audio will also work. The most common way to handle resampling is by first pre-filtering the initial motion to remove any changes that will be too fast to be captured by the new sampling rate, and then to sample this limited signal. The pre-filtering process applies a low-pass filtering operation to the motion. This process will be described in more detail later.

1.6.2.2. Time Warping

Time warping is a more general form of time transformation. It effectively scales different parts of time in different ways to achieve desired results. We can specify times that correspond from the initial motion to the final, and then compute a transformation that maps the times, interpolating in between, as shown in Figure 14.

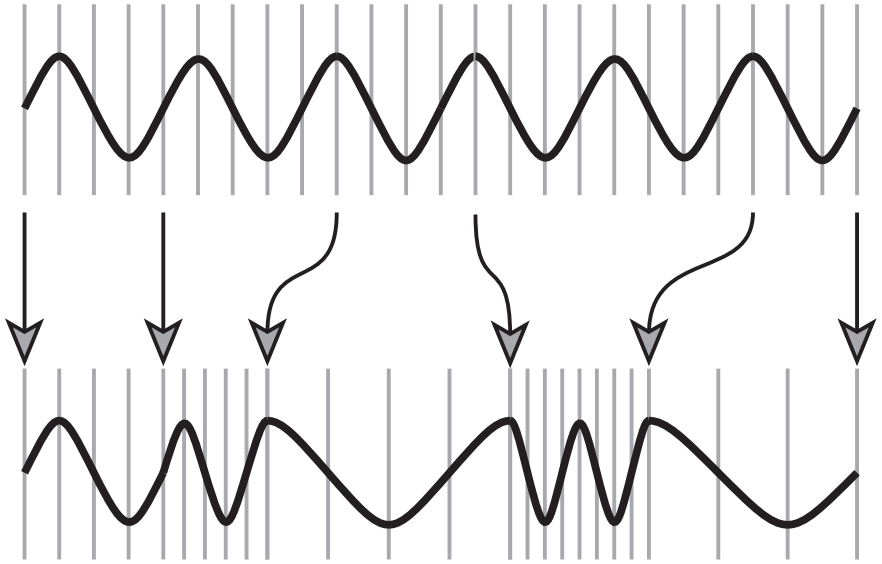


Figure 14: A Time warp makes specified times match, compressing or expanding time between the specified points.

In general, time warping defines a function that maps from times to times. This function is defined by a number of points that it interpolates. Any type of interpolation can be used, and the literature of common techniques for graphics and animation can be applied.

Time warping is useful when we would like to make one motion's events occur at specific times, such as at identical times as some other motion. For example, consider animating two characters marching. If the first character's footsteps occur at times 20, 40, 60, and 80, and the second characters footsteps occur at times 25, 39, 63, and 84, these correspondences define a time warp. This is illustrated in Figure 15.

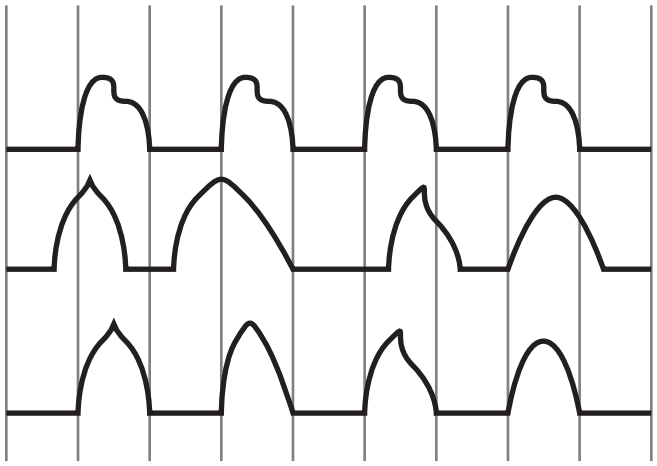


Figure 15: An example of a Time Warp. The top two rows show two different marching motions. The lower one (middle row) is altered to have its footstrikes be simultaneous with upper motion, resulting in the bottom motion.

Creating an interpolating function to create the time warp is simple. However, care must be taken in resampling: because various parts of the motion will be sped up and slowed down by various amounts, resampling may not be simple. Techniques in the signal processing literature address the issues of non-uniform resampling.

Methods from the signal processing community are capable of automatically determining a time warp that causes two signals to correspond. These methods have been applied to animation problems, for example by Bruderlin and Williams in (Bruderlin 95).

1.6.3. Filtering

In signal processing terminology, a filter is a process that transforms one signal into another signal. This incredibly broad definition includes any transformation we might make to a signal. There are some standard transformations that are useful across a variety of signal types. These may be applied to motions as well.

1.6.3.1. Linear or Frequency Filtering

One very common and general type of filter performs a scaling of a signal's frequency components. For example, a filter might allow only certain frequency ranges to pass through and block others. Such filters are called high-pass, low-pass, or band-pass filters. A low-pass filter allows only frequencies less than its cutoff to pass through, while a high-pass filter allows frequencies above its cutoff. The term band-pass is used to denote a filter that allows a range to pass, and includes high- and low-pass filters as special cases. A filter that completely cuts off frequencies outside its range without disturbing frequencies inside it is called an ideal filter. Real filters generally have approximate rolloffs.

A filter can be thought of as dividing its input into two parts: the part that passes through, and the part that is cut. A filter bank is a set of filters that divide up a signal into a number of different signals by having each filter cut a different part of the original signal. A common technique is to use a filter bank to divide a signal up into a number of components, manipulate each component independently, and then to reassemble the signal. The most common use of this is to attenuate (scale) each individual component. An audio graphic equalizer is a device that does this for audio signals.

Bruderlin and Williams (Bruderlin '95) demonstrated the use of a graphic equalizer for motion signals. Such a tool is useful for understanding the meanings of the frequency content in motions, because it allows a user to experiment with adding and subtracting different frequency information.

Unfortunately, such direct frequency domain control is rarely useful on motions. Some low-pass filtering may be useful to reduce noise, but generally, frequency manipulation creates unusual effects in motion, while destroying various properties such as positions of end effectors. The main utility of frequency domain analysis is to provide insight to signal properties of motions so that other tools can be developed.

1.6.3.2. Non-linear and Other Filters

There are many other types of filtering operations that are useful on motions, but are not linear operations or have descriptions in terms of the frequency domain. While the mathematical analysis of such operations may be more difficult, many are still very useful to perform changes to motion data.

One common non-linear filter used for motion is the median filter. A median filter produces a new signal whose value at a given time is the median of the values in the original signal at times around the given time. For example, a 3 sample wide median filter would compute:

$$M(f(t)) = \text{median}(f(t-1), f(t), f(t+1))$$

Median filters have the effect of discarding outliers, single data elements that differ from the surrounding data. This can be useful in removing spurious samples from motion capture.

A generalization of a median filter is to use robust statistics that compute averages but discard outlying elements. For example, we might take the same three elements as the example median filter above, discard the one that was farthest from the average, and average the other two. More generally, we might take some number of nearby samples, discard some portion of them as outliers, and average those. Such filters can be more effective at noise reduction than simply performing frequency-based filtering.

Other filters may perform non-linear operations on individual values. This is sometimes called wave-shaping because it tends to change the "shape" of the curve. In their work, Bruderlin and Williams (Bruderlin 95) showed how a variety of simple mapping operations could lead to interesting effects on motions.

1.6.4. Additive Motion Editing

In this section, we consider a time domain operation on motions very different from the filters of the previous section.

When we view motions as signals, it is obvious that we can trivially apply basic mathematical operations to them. For example, we can multiply a signal by a constant. Just about any mathematical operation can be applied to a motion signal, but the question is to find ones that make for useful changes on the resulting motion. Addition of two signals, combined with scaling of the signals, can perform a surprising array of useful operations.

Adding two signals together simply adds their values at each time instant

$$a(t) = f(t) + g(t).$$

For motions, we can think of this as "pose-wise combination." At any time, the pose would be the combination of the poses at the same time. We emphasize that this operation is independent of the representation of the motions being added.

Simply adding two poses together does not necessarily give us a meaningful new pose. What is more likely to give us a meaningful pose is to take a weighted average of the poses, for example:

$$a(t) = \tfrac{1}{2} f(t) + \tfrac{1}{2} g(t)$$

or, more generally,

$$a(t) = \text{alpha } f(t) + (1\text{-alpha}) g(t).$$

At each individual instant in time, the value of the signal (or the pose) will be part way between one motion and the other. That is, this blends two motions together, and is therefore called motion blending.

From our earlier discussion of rotations, we should remember that simply taking the number half-way between the numbers representing two rotations does not actually give us a rotation half-way in between, especially for Euler angles. In practice, such linear combination blending is still used, despite the fact that it is mathematically ill defined. As we will see, in general, blending

tends to work better in practice than in theory. If our rotations are expressed as unit Quaternions, the problem of combining two rotations for a blend is more explicit. We cannot add the Quaternions, but instead must interpolate between them.

Blending two poses may or may not be meaningful. For example, if we try to blend a person standing up with a person lying down, we might get something halfway in between that is floating diagonally in space, as seen in Figure 16.

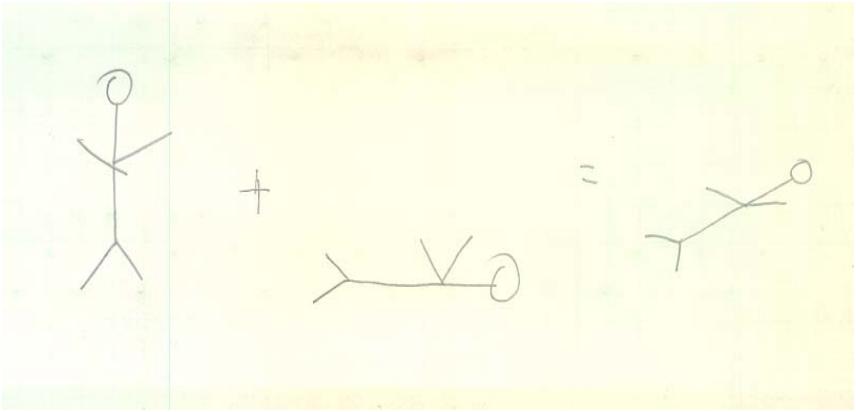


Figure 16: Blending two poses may not provide a meaningful result

Similarly, blending a pose from a walking motion that has the foot at the top of the step and at the bottom of the step leads to the foot being in the middle, which may be acceptable for one pose, but not for a motion, as illustrated in Figures 17 and 18.

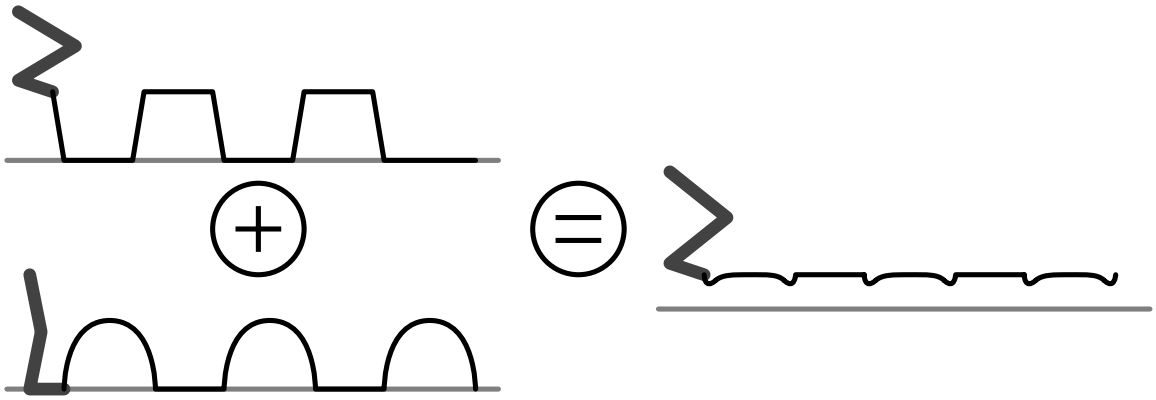


Figure 17: Blending may not work with non-aligned motions. Here, a walk and a march are blended into a hovering quiver because they are not time aligned.

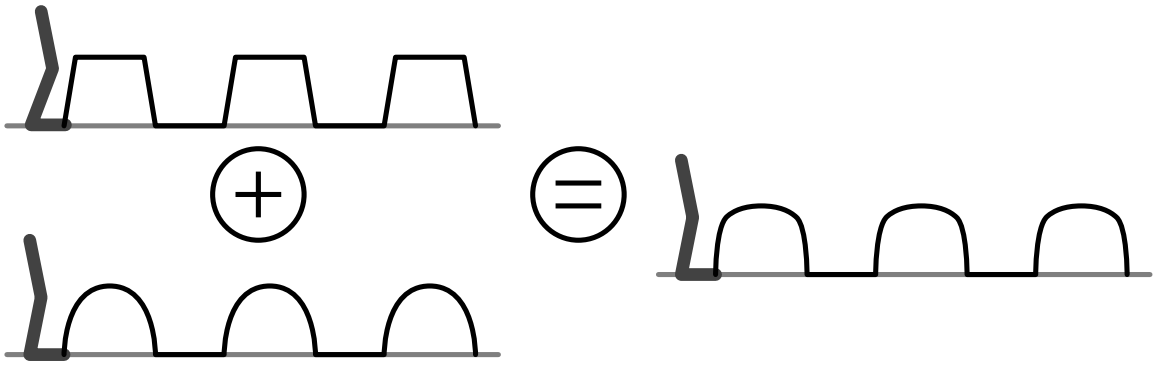


Fig 1-16b. Blending of the same motions as in 1-16 gives a desired result that is "half-walk, half-march" when the signals are time aligned.

In practice, blending requires motions to be similar, and to be time aligned. That is, if we try to blend two walking motions, we must make sure that the up and downs parts of the foot swings are at the same time, so we don't end up with the foot always being in the middle. This is often accomplished by adjusting the timing of one (or both) of the motions such that the key events in the motions correspond. Timewarping is a particularly useful method for this as it can synchronize two motions based on known events. Methods in the signal processing literature, such as dynamic Timewarping, can attempt to automatically find the correspondences.

There is no reason why we need to limit our blends to two motions. If we have several motions, we can add them together as well

$$\mathrm{m}(t) = a_1\,\mathrm{f}(t) + a_2\,\mathrm{g}(t) + a_3\,\mathrm{h}(t) + \cdots$$

This is often called multi-target blending, and is most useful when we have a number of very similar motions. For example, Rose et al. (Rose 98) used this technique to combine walking and running motions of various moods. The most difficult part of performing the multi-target blending is to find a set of motions that are similar enough to be blended.

Blending provides a way to alter motions "by example." While we may not be able to provide a mathematical definition of what makes a walking motion angry or sad, we can identify motions with these properties when we see them. By blending in motions with desired properties, we gain the ability to specify changes despite not being able to create mathematical descriptions of them.

Blending in Practice

Blending is one of those things that works in practice, but not in theory. In theory, there is little reason to believe that simply adding two motions together necessarily gives something meaningful. In practice, blending works quite well in situations where motions are similar and synchronized. While this may not be every case, there are some specific uses where these conditions are met. An example is a transition between two motions.

Blending is particularly useful for making transitions between motions. In such cases, we often have motions where the beginning of one motion and the end of another are similar, but not exactly the same. For a brief period, we use a motion that is a combination of the two motions, and vary this combination so that we use the first motion at the beginning, and the second motion at the end, as illustrated in Figure 19.

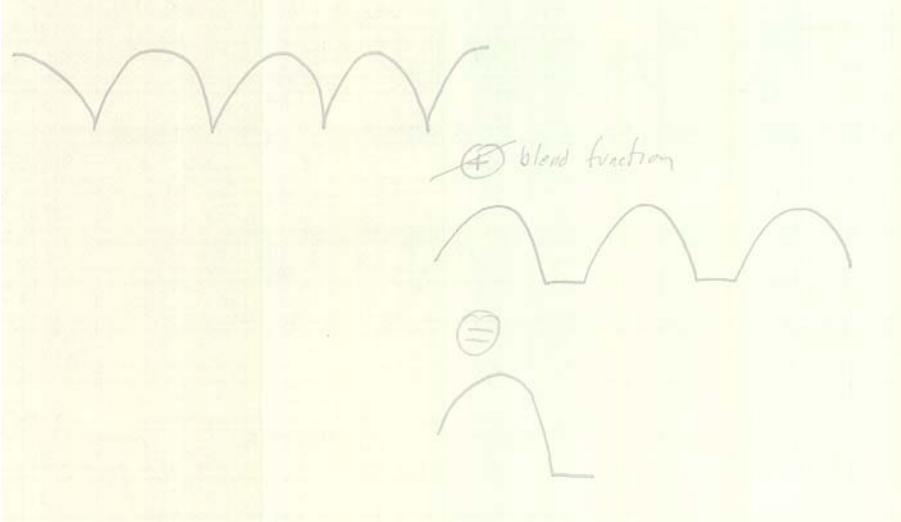


Figure 19: Using blending to create a transition

This method works well because the motions are similar during the blending period, and the time is short, so even if the poses created are not perfect, they aren't seen long enough to make a difference.

Making transitions when the motions' ends are not similar enough to blend can be more problematic. Rather than blending, we can use interpolation between the end of one motion and the beginning of another. Just as with blending, interpoating between two poses is only meaningful if the poses are similar. More sophisticated methods of generating transitions between more disparate poses are an important research direction. Rose et. Al (Rose '96) present one promising approach.

1.6.5. Motion Warping

Blending is an extremely useful tool, when we have motions that can be blended together. In this section, we consider creating special motions that are created just for blending. One example of this might be to capture very similar motions just for the purposes of blending, for instance, if

we capture a happy walking motion and a sad walking motion, we can blend the two to make combinations. Motion warping is a different kind of technique where we specifically create a motion that effects our original motion in a specific way.

For the motion warping technique, we create a new motion by adding our original motion with a specially created motion

$$m(t) = f(t) + d(t).$$

By being careful in our choice of the special motion, we can create desired effects in the result, without disturbing certain properties in the original. This specially created motion is sometimes called a motion displacement map, and this method is sometimes called motion displacement mapped. Bruderlin and Williams (Bruderlin ’95) used this term, while Witkin and Popovic (Witkin ’95) used the term Motion Warping.

For a simple example of motion warping, consider a motion curve that we have a few known changes that we would like to make, as shown in Figure 20.

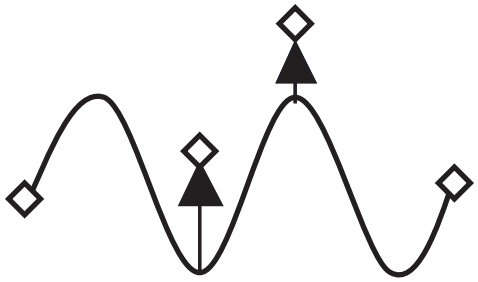


Figure 20: A motion curve with desired changes specified at specific times. New poses are specified for several instants.

If we change the motion at only these times, we would get an undesirable result of Figure 21:

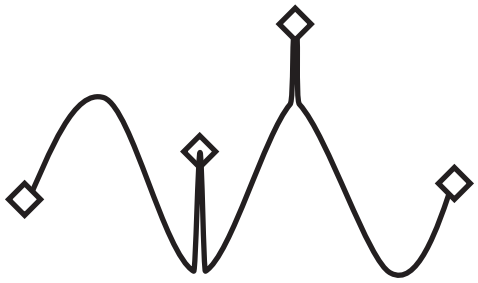


Figure 21: Simply changing the motion at the specified instants leads to an undesireable result.

If we change the motion by adding in another motion, each of the changes tells us something about the added motion. It must have values that create these desired goals.

We can pick any motion that goes through the points as our displacement maps. One useful thing to do is to interpolate the changes, as seen in Figure 22, so that there are no abrupt changes in the resulting motion, shown in Figure 23.

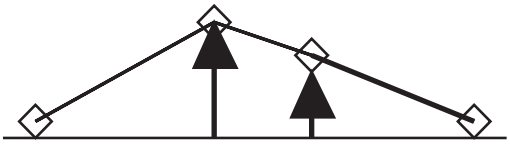


Figure 22: Interpolating the requested changes yields a displacement map that can be applied to the original signal.

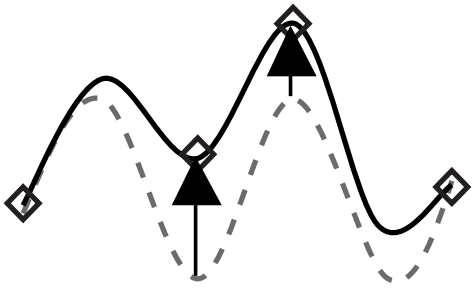


Figure 23: Adding the displacement map to the original yields the modified signal.

In a sense, we are keyframing the changes that we would like to make to the motion.

If we specify a single desired change, we can control the scope of the effect by our choice of the displacement map.

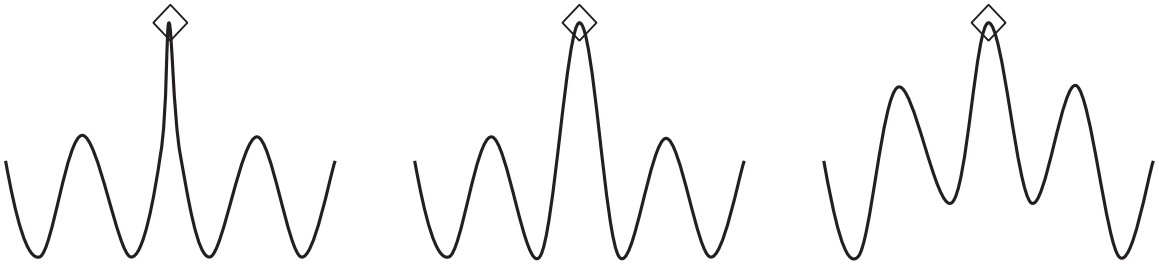


Figure 24: Different scopes of change can be created by controlling the properties of the displacement curve. In this example, each variant uses a change at the center and two specifications of zero change at differing distances.

The steps in this process were:

- We find desired changes on individual frames. We can use any tool we like to change the individual frames (for example, we might use inverse kinematics to position the figures)
- We determine the displacement values at each of the changed frames by subtracting the original from the new poses.
- We construct a motion displacement map (a.k.a. a motion warp) that interpolates these known displacements.
- We compute the final motion by adding the original motion and the computed displacement map.

The simplicity of this process belies its power: it permits us to use any per-frame or keyframe tools that we like, and have their effects applied to motion capture data without having to modify each individual frame of the motion. In brief, the motion warping process decouples the editing of motion from the representation of the motion.

Let's consider a specific example of editing a walking motion. Suppose we want to reposition one of the footsteps of the motion. Using inverse kinematics (or other tools for operating on an individual frame), we can change the pose during the footplant such that foot has the desired

position, and the character is in a reasonable pose. We have defined a change on a single frame of the motion. Looking at a single curve (for example the knee)

If we simply applied this change to the motion capture data, we would only be changing a single frame of the motion, this would give an undesirable result of the figure simply “jumping” to this pose on this one frame, probably in a flash so brief that we would not see it.

With motion warping we can get control over the nature of the change that we make. Rather than being limited to simply changing individual frames, we can choose the scope of the change by picking other frames that we do not want to change.

We can now reconnect motion warping with the frequency concepts we were discussing in an earlier section. The motion warp allows us to change the “big picture” of the motion, without changing the details. As we discussed earlier the low frequencies give us the big picture, while the high frequencies of the motion provide the details. In a sense, motion warping allows us to choose the frequency at which we make the alteration to the motion.

Changing the individual frame of the motion adding a large amount of high-frequencies into the motion by introducing the large, unnaturally rapid changes. By choosing a larger range for the motion warp (as in illustration), we get a smoother “bump” that adds a lower frequency to the motion (because it is less abrupt).

While motion warping is convenient in that it allows us to alter overall characteristics of the motion without changing the details, this actually is one of the problems in using the technique: often the details must be adjusted to keep important properties of the motion. While motion warping allows us to specify any changes we want to the key frames, it gives us little control over what happens on the frames in between.

For example, consider altering a character’s walking motion such that the character steps up onto a bump in the road. Ideally, the motion editing process should allow us to specify a change to one of the frames. For example, Figure 25 shows the motion of the character's foot, with a desired change specified.

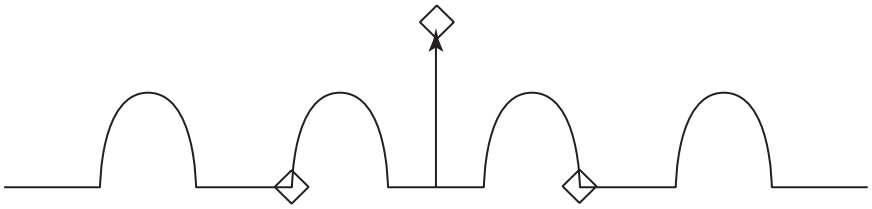


Figure 25: The height of a foot in a walking motion. We would like to specify that the character steps over a bump in the road by changing one frame.

Unfortunately, this is ineffective in practice. Because the motion warp does not “know” that the foot’s height must remain constant when planted, we get a motion that is more like an escalator ride than walking over a bump.Figure 26 shows a linear displacement map added to the motion of Figure 25.

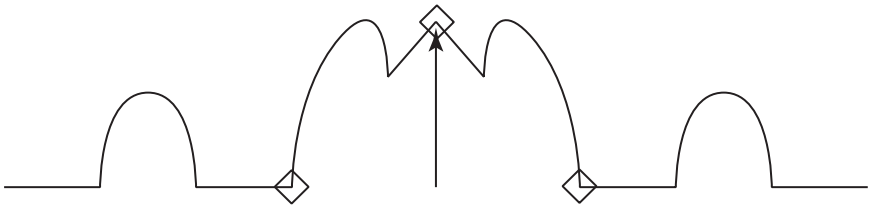


Figure 26: When a displacement map is used for the change specified in Figure 25, an undesired result occurs..

The obvious solution is to simply specify more key frames. For example, had we specified a key frame at the beginning and the end of the footplant, we would get a more reasonable motion warp, as seen in Figures 27 and 27.

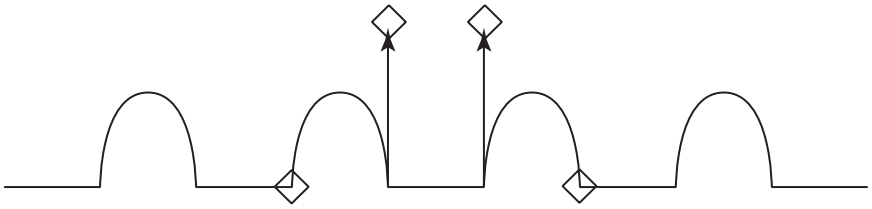
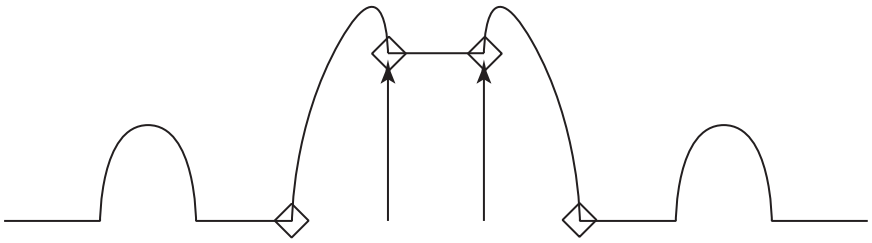


Figure: The bump in the road is specified at both the beginning and end of the bump.



Figure, the character steps over the bump.

While this is simple when we are simply keyframing the one parameter, with a real figure, the solution is more difficult. Creating this change would require repositioning the figure in both frames in a consistent manner. This can be challenging, but is probably less work than having to alter all of the frames.

There are better alternatives to manually specifying multiple keyframes. We will use this same example to motivate a very practical approach in the next section, and later in the chapter, describe an approach which addresses the issue by using a more complex mathematical technique.

Layering Motion Warps

Motion Warping shows the utility of adding one specially created motion to the original motion in order to edit it. There is no reason to stop with just one motion: we can add in several displacement maps simultaneously to achieve additional effects. The addition of multiple warps is sometimes called layered editing because each new displacement may be considered as a new "layer" added to the top of the existing work[3].

The most obvious use of layered editing is to have several independent edits to a motion. For example, we might take a walking motion and add in a hand wave, an alteration to the feet, and a nod in the head. Because each of these may need different timings, it may be easiest to create each alteration independently, and then add them in as needed.

[3] Beware, there are other usages of the term "layering" in computer animation and motion editing.

In fact, one way to use this approach is to create a “base” motion to which smaller details can be added. For example, we might have the base walking motion, and a variety of things that might be added to it, like hand waving and head nodding.

Another use of layered motion editing is to create a single motion warp that would be too difficult to specify as one layer. As an example, consider the step example of the previous section. We could make a single motion warp as in the previous section, and then use this as a starting point to create another motion warp that gets the footplant details correct.

Part of the idea of using this kind of layering is that the first edit is a broad stroke, and then finer and finer details are added in until the desired effect is achieved.

Motion Warping Summary

Motion warping is an extremely powerful technique for a number of reasons:

- It is relatively easy to implement.
- It allows any of the existing techniques to be used on individual frames, often without any change to the software.
- The editing done is independent of the representations of the existing motions.
- We gain control of the scale and scope of editing operations.
- We can create edits that preserve the frequency content of the original signal.
- We can use layering to combine multiple warps.

The technique does have a number of drawbacks:

- Addition of new curves may not be well defined (for rotations). This may be addressed by using a correct composition operation (for example multiplying Quaternions or rotation matrices).

- We need to perform interpolation of the displacement maps. This can be tricky for some types of parameters (such as rotation), and may require us to specify what kinds of interpolation to do. In general, we may want to have a "complete" set of keyframe tools for our keyframed changes.
- We get no control over what happens in between the keyframes. This may lead to constraints being violated.

1.7. A Constraint-Based Approach to Motion Editing

Thus far, we have considered what types of mathematical tricks can be applied to motions in a useful way for motion editing. In this section, we return to the original statement of the motion editing problem and use this to motivate a different set of mathematical tools for addressing motion problems.

Broadly, a motion editing operation attempts to change some aspect of our initial motion, while preserving as much of the rest of the motion as possible. We might state our motion editing problems as follows:

> Find me a new motion that that while meeting my specified new needs, preserves as much of the original motion as possible.

Let's consider a simple example. Imagine that we have a motion of a character waling across the room. We would like to have a motion of the character walking across the room and reaching for a doorknob. To change the former into the latter, we must adjust the motion in a way that meets the new goal (reaching the doorknob), yet preserves what we liked about the original (that it is a realistic walking motion, that it has a somewhat depressed mood, …) A 2D example of this is shown in FIGURE.

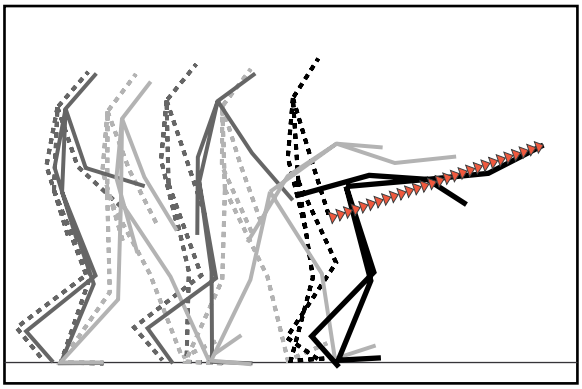


Figure: A walking motion is edited by specifying a new position for the character's hand in the last frame of the animation. The original motion is shown as dotted lines.

Notice that despite the fact that we only specified a goal on the last frame, the entire motion needs to be adjusted. If we only changed the last frame, where the character must be all the way on the right side of the room in order to reach the doorknob, the character would need to jump quickly between the last and next-to-last frame. Instead, we need to adjust the entire motion, having the character take larger steps in order to reach the far side of the room, as well as, bending over to reach. The motion editing tool must pick some new motion that preserves the important properties of the original motion (that it is a realistic walk, that the character is somewhat sad, …), but meets the new need.

To pose this as a math problem, we need ways to describe the kinds of changes that we might like to make, as well as ways to measure how well the resulting motions match the original motion.

Good transformations preserve important aspects of the motion by altering less important ones: in a walking motion, it is important that the feet touch the floor, not that the pelvis is 32 inches above the floor as in the original. The requirements for a good transformation are (in order of importance):

- That any specific requests for change are met;

- That any specific, defining characteristics of the motion are kept;
- That the transformed motion is as similar as possible to the original motion.

Mathematically, this can be phrased as a constrained optimization problem: subject to meeting the constraints (the specific definitions of 1 and 2), minimize the difference (or maximize the similarity) with the original motion.

The constrained optimization view of motion transformation requires that the "desirable properties" of a motion be encoded into constraints and the similarity metric between motions. Ideally, the constrained optimization problem would fully encode our desires mathematically: there would be a single solution that was the desired motion. Realizing this ideal requires a rich set of constraints and objectives. For example, we could find constraints that enforce the laws of physics, biomechanical limitations due to strength, and proper ballet form. We could define objective functions that measure visual properties such as "grace," "Charlie—Chaplin—ness," and "like—Joe—did—it—yesterday—ness." We would aim to maintain the constraints that were satisfied in the original motion while minimizing the amount of change in the important properties.

There are central difficulties in realizing this constraint-based ideal for motion transformation: first, some properties are difficult to encode mathematically as constraints or objectives either because the forms of the equations are complex or because they elude a mathematical encoding; second, we may not know all the properties required, such as the mass distribution of an imaginary character or the physical laws of an imaginary world; third, we must decide which properties are important in a given setting; fourth, many of the properties and constraints may be specific to a small set of examples, and therefore not worth the effort to define. All of this, of course, presumes that we can pose and solve the mathematical constraint problem.

The constraint-based approaches to motion editing are, at present, still experimental, and far from meeting the ideal of permitting the rich and complex descriptions of motion properties we would like. We describe some of our early work with these tools to give a preview of what might be possible in the future, and to give insight on the limitations of other techniques.

1.7.1. Constraints on Motion

Constraints are a specific statement that can be posed mathematically, typically about a particular pose of the character. For example, we might say that the foot touches the floor in frame 5, or the elbow is less than 180 degrees in frame 10. Some constraints are convenient to specify over a range of frames, for example, that the elbow is never bending backwards, or that the foot never goes through the floor.

Constraint-based techniques are already part of the tools used to edit motions. Inverse kinematics is a common use of constraints: the inverse kinematics solver determines a set of values for the character's parameters that satisfy the constraints.

When solving constraints, there are often many possible ways to satisfy them. For example, if we specify a constraint that places the hand of a character, there are many possible poses of the character that might place the hand in the specified position. The constraint solver must choose the "best" solution, where best is based on some criterion that it has for choosing. While sometimes solvers may use ad hoc methods that pick solutions in an unprincipled way (the "best" is defined as whatever is easiest for the solver to find), good solvers define a measurement that they try to minimize. That is, the solver tries to find solutions that subject to meeting the constraint (of placing the end effector), they minimize the objective function. The most common objective is to minimize the amount the angles are changed.

An inverse kinematics solver allows us to specify details of a motion, such as that the character's foot must be in a particular location. However, it performs the computation on an individual frame. Because of this, there is no guarantee that what happens between frames will be consistent. What the solver decides is best on one frame may be quite different than what the solver decides is best for some other frame.

Consistency can be helped by using a solver that uses a well-defined criterion that achieves similar results on similar problems. For example, if the solver attempts to find a solution as close as possible to the original, and the original motion is consistent between frames, and the

requested end-effectors positions' are consistent across frames, then we should expect to get motions that are continuous across frames

Another choice is to have the solver use consistency as its criterion. For example, rather than trying to match the original pose, the solver might try to match the previous frame in the resulting motion. While this may lead to motions that are continuous, they will not keep as much of the original motion.

Neither of these schemes work in cases where the constraints switch on and off. For example, imagine a constraint for a footplant on the walking motion that keeps the foot on the floor. This constraint only exists during the footplant. If the solver only considers one frame at a time, the frame before the footplant will know nothing of the footplant. Therefore, the foot might snap to the floor when the footplant begins. This problem of being able to look ahead also works in reverse: frames after the footplant may need to look behind to make sure they have consistency.

One way to look at this problem is as a limitation on how much information the solver considers in determining what solution to choose. When given limited information (a single frame), the solver is unable to consider properties such as smoothness or physical realism, which require looking at a longer segment of the motion. By permitting the solver to look at a wider window, we can have the solver attempt to maintain (or create) more complex properties.

Even considering more of the motion is insufficient. Simply knowing that a footstep is coming does not necessarily say where it will be. Therefore, to truly perform solutions on more complex criteria, the solver must be able to either perform the computations in the correct order, or to solve for all of the frames simultaneously. This latter approach is what we call Spacetime Constraints.

1.7.2. Spacetime Constraints

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 criterion 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 (Witkin 88) and Cohen (Cohen 92), 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. While the approach offers the potential for high level properties to be employed as criteria, to date, concepts such as graceful or angry 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, very difficult to solve constrained optimization problems.

We initially proposed applying the spacetime constraints approach to a motion transformation problem in (Gleicher and Litwinowicz 98)[4]. 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—ness, 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

[4] Which initially appeared as an Apple Computer Technical Report in 1996.

the user identify) these properties. Even if we encoded the desired animation completely in a constrained optimization, we still need to solve to these problems. Generally, richer sets of constraints and objective functions lead to more difficult problems to solve.

1.7.2.1. Our Pragmatic Approach to Spacetime Constraints

Our approach to realizing the spacetime approach to motion transformation has been pragmatic: we make simplifications such that we can create tools that we can apply to realistic problems. To date, we have made many sacrifices to achieve practicality: we tell our solver little about the original motion or general motion properties, and our choice of the mathematical problem is heavily influenced by what can be solved efficiently. Specifically, we have restricted ourselves to specific geometric constraints and mathematically simple metrics of differences in motions. For example, we augment the motion of Figure [] with constraints that are essential to the action: the hands must grab the box in the middle frame, the hands must remain the correct distance apart while carrying the box, and the feet must be planted and not skid when they are on the ground.

We sometimes pay for these sacrifices in the quality of the resulting motions. For example, because our system did not consider gravity or posture we get an unrealistically unbalanced result in the right frame of Figure []. However, the payoff is that our approach provides a practical solution to the transformation problem and a framework in which to employ more sophisticated constraints, like strength and balance, in the future. Also, because we can get the solutions quickly and interactively, the user immediately can see unacceptable results and can rectify the problem by making adjustments or specifying additional constraints interactively.

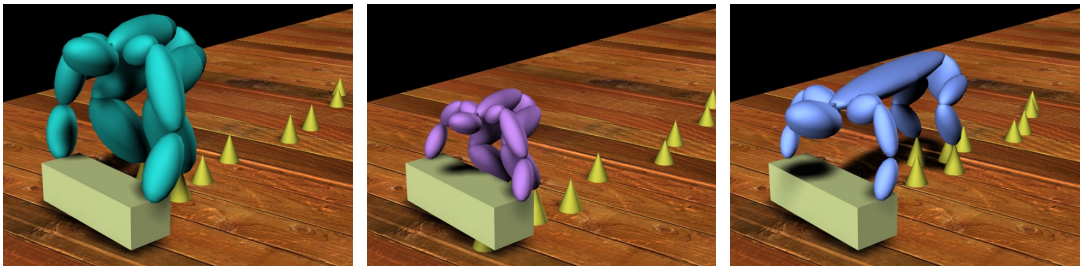


Fig. 0-1. Differently sized characters pick up an object. Their positions are determined by the position of the object. The left shows the original actress. The center shows a figure 60\% as large. The right shows a figure with extremely short legs and arms and an extremely long body. The yellow cones represent footplant positions

1.7.2.2. Comparing Motions

Our objective for transformations is to minimize the amount of changes that damage the desirable properties of the initial motion. However, even this could lead to difficult to define, high-level, problem specific metrics. For example, in a walking motion, a slight bend of the knee may not make a difference, but this same slight bend of the knee could be a noticeable deviation from perfect ballet form in a dance.

To date, we have sought simple, generally applicable metrics that can be computed with efficiently. We rely on constraints to specify details of motions that must be retained, reducing the importance of the objective function..

Generally, the high frequencies of a motion (or the lack thereof) are important, and therefore must not be disturbed. An adaptation that removed the snap from a karate kick might be just as inappropriate as adding a snap to a slow walking motion. We therefore limit the frequency content of the transformations to avoid disturbing the high-frequency content of the original motion.

We use our choice in how to represent the motion as a tool for placing frequency limits on transformations, as well as to aid the efficiency of solution. Liu et al. (Liu 94) first made use of a

carefully selected representation by using wavelets to speed computations. We introduced the use of motion-displacement maps as a representation (Gleicher 97, Gleicher and Litwinowicz 98) for spacetime problems where the objective function related two motions. This approach defines

$$m(t) = m_0(t) + d(t)$$

and uses the solver to find d(t). The approach has a number of advantages. First, it decouples the solution from the form of the initial motion, providing generality. Secondly, it simplifies placing constraints and objectives on the changes. Third, it permits us to choose a representation that has mathematical properties that are efficient to compute with, even though it may not be good for representing the motion. Fourth, it allows a representation for d(t) to be chosen that includes constraints on the changes so they do not need to be expressed as explicit functions. To constrain the displacement signal not to include high frequencies, we use a representation for it that cannot represent the high frequencies: specifically, cubic B-Splines with control point spacing determined by the desired frequency limits. The control points of the displacement curve need not be uniformly spaced: we can place controls closer together for portions of the motion where higher frequencies are acceptable. B-Splines also have mathematical properties conducive to efficient solving.

The objective function for the constrained optimization must compare the initial and resulting motion The most basic comparative objective function would be to compare the values of the parameters, matching pose in parameter space. For example,

$$g(m) = \int_t (m(t) - m_0(t))^2 ,$$

minimizes the magnitude of signal differences in the motions over time. This simple objective relies on the constraints and motion representation to provide coupling between time frames, but simplifies he constraint solving problem to afford an efficient solution.

We have begun to explore the design space of objective functions, for example creating frequency criteria by minimizing the output of a filter that selects undesirable elements. In practice, we find that pragmatic concerns outweigh most other choices in the design of an objective function. Increasing the complexity of the objectives leads to considerably more difficult optimization problems.

1.7.2.3. Constraints

Constraints in our approach serve two purposes: they encode specific aspects of the motion that should be maintained during subsequent edits and they serve as handles to drive changes to the motion. There is little distinction in our system, in fact, constraints are often moved between the categories. Most constraints that we consider in our approach are kinematic, that is that they place a restriction on the configuration of the character at a given instant. These constraints have the form

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

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

In our prototype system, the user never needs to see an equation: the system includes a variety of pre-defined constraints that can be applied to a motion through a graphical user interface or via a scripting language. Some constraints can be identified semi-automatically (such as finding footplants by examining foot height). Once the constraints for a character or motion are defined, they can be used for any adaptation made to the motion or using the character.

We have emphasized finding (and using) constraints that we believe are applicable over a wide range of motions. Some useful examples include: limiting the range of a joint angle (so elbows can't bend backwards); specify the location of a hand (to grab an object) or foot (step in a

particular place); keep a body part in a certain region (stay above the floor); have a hand or foot follow a specific path; prevent a foot from skidding when planted; or keep two points a specified distance apart (useful for when a character is carrying an object of a fixed size).

The architecture of our system is designed to minimize the effort required to add new types of constraints.

1.7.2.4. Successes of Our Approach

To date, we have used our pragmatic spacetime approach on two important animation problems: retargetting motions to new characters and interactive editing of motions.

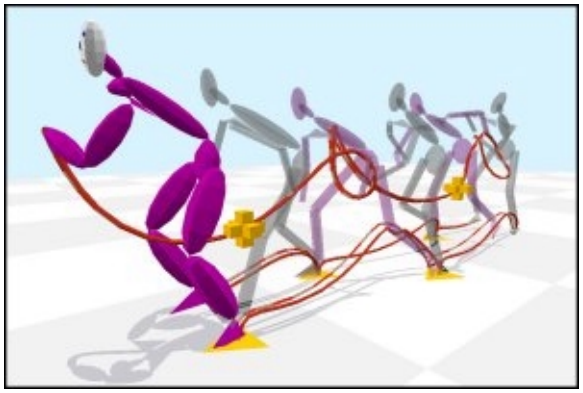


Fig. 0-2. Visualizing a spacetime constraint-based motion edit of a walking motion. Strobing (with color alternation and transparency to help contend with clutter) and streamers (the thin lines following the feet and hand) are used to convey the motion. The striped streamer shows the initial (pre-edit) motion. Yellow symbols represent constraints

We have applied the spacetime constraints approach to the problem of interactive editing of motions (Gleicher 97). The user can directly manipulate constraints, such as the position of a hand in a particular frame or the location of a footplant, and the motion is altered in real time. We take an extremely pragmatic view of spacetime, making as many simplifications as possible in order to achieve interactive solutions. The idea is that it is less important to achieve the "right" solution to the constraints because the user can make further interactive adjustments to achieve

the desired results. Through careful implementation and choice in problem design, our prototype can allow the user to interactively manipulate motions with thousands of constraints.

A difficult problem in the spacetime editing system is helping the user visualize the results. As seen in the example of Section [], a change can effect the entire motion, so the user must be able to see the entire motion to understand what they are doing. As seen in Figure [], we have used a number of simple tactics to try to convey the editing operations to the user including strobing, trace streamers, and multiple windows showing motion loops. We use multiple visual clues, including motion, color, texture, transparency, and shadows, in our attempt to convey the behavior.

We have also applied the spacetime approach to the problem of retargetting motions from one character to another (Gleicher ’98). Our focus is on adapting the motion of one articulated figure to another figure with identical structure but different segment lengths, although we use this as a step when considering less similar characters. For retargetting, we define constraints on the original motion, apply the motion to the new character, and then use the spacetime solver to compute a new motion that re-establishes the constraints.This process is shown in the following FIGURES.

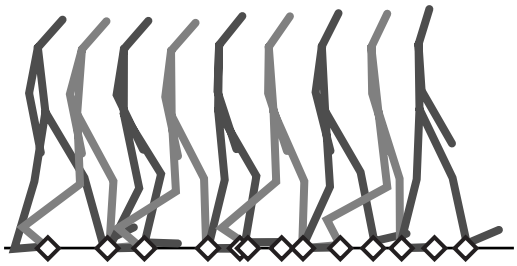


Figure: The original motion of a tall character walking. Constraints (in this case footplants) are identified.

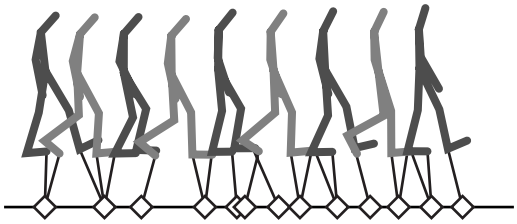


Figure: Step 1, the motion is applied to the smaller character. The constraints are no longer satisfied.

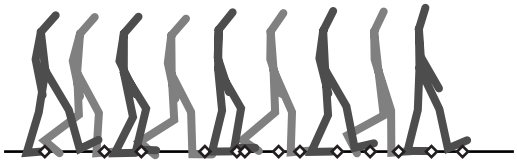


Figure: Step 2, an initial guess is made to get an answer closer to a solution. In this case, the motion is simply translated such that the characters feet touch the floor. This is not a solution because the horizontal position of the feet are not correct, causing skating. Note, this step can be omitted.

Figure: Step 3, the constrained optimization problem is solved. The constraints are re-established in a way that preserves as much of the original motion as possible.

Without adaptation, our motion data does not apply to figures of different sizes or proportions than the original: the resulting motions have the feet skating and the hands failing to reach the

object. Our method enables us to re-use this data on figures of varying proportions, as shown in Figure []. Figure [] shows a more challenging retargetting example. In this case, we allow the spacetime solver to transform both dance partners' motions to adapt to the change in one character's size.

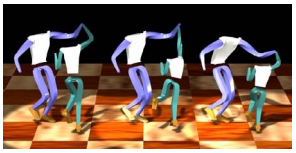


Fig. 0-3. Adaptation of a motion-captured swing dance (left) to a smaller character. Our method can adapt the female's motion such that her feet touch the ground and she holds hands with her partner (center). We can also adapt both character's motions to achieve a better result (right).

Our retargetting approach also works for cases where the character is changing over time, as shown in Figure []. We also demonstrated how the technique could be used to retarget motions to characters with very different structures, such as making a can skip like a person. Our technique first adapts the human motion to a human with the same proportions as the target, and then finds a motion for the target character so that corresponding parts follow the same paths through space.

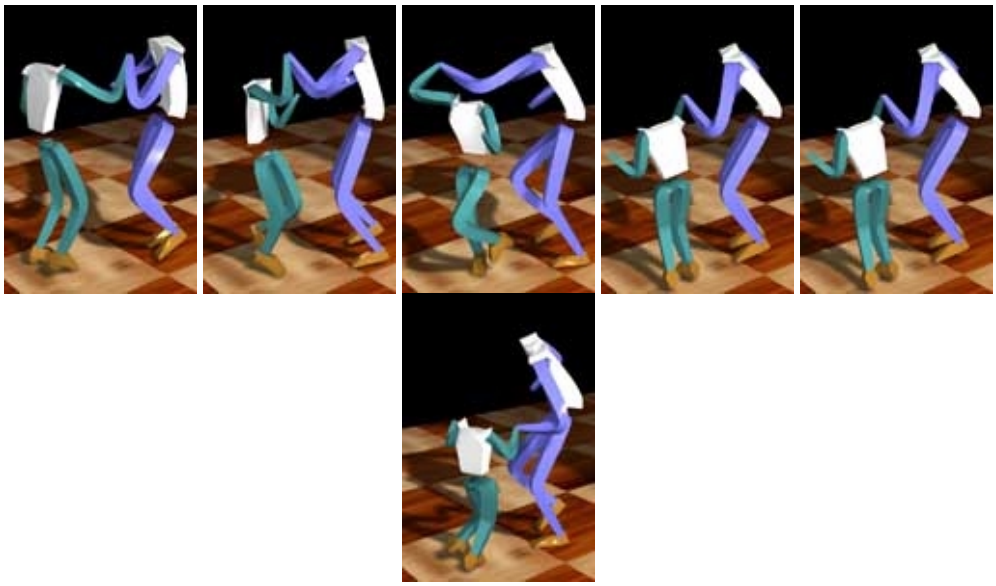


Fig. 0-4. The female character morphs into a smaller character during her spin.

The retargetting method described works when the characters have an identical structure, that is, they differ only in the length of their limbs. In this case,the motion can be trivially applied from one to the other to provide an initial guess. The problem of automatically retargetting a motion to a character with different structure is more challenging as the solutions almost always require a great deal of creativity to create.

1.8. References

The use of hierarchical representations for 3D character animation dates back to the earliest experiments, including Catmull's animation of a human hand (Catmull 72). The work of Zeltzer (Zeltzer 82) demonstrated some of the earliest use of a full body skeleton. The concepts for hierarchical modeling are presented in any standard graphics text or reference, such as Foley, van Dam, et al (Foley 90), Hearn and Baker (Hearn 97), or the OpenGL reference book (Woo 99) provide good introductions.

The examination of rigid body motion is the primary motivation for the development of the mathematics of rotation. A mechanics text, such as Goldstein (Goldstein 80), presents the formalism of rigid rotations and the mathematics of their representation, including Euler's important results. Quaternions were introduced to the graphics community as a representation for rotations by Shoemake (Shoemake 85), and are now common enough to be included in graphics texts such as Hearn and Baker (Hearn 97). The exponential map was introduced to the animation community by Grassia (Grassia 99) who provides an outstanding discussion of the relative merits of many representations for rotations for a variety of animation problems. Bregler and Malik (Bregler 97) use exponential maps to represent human figures for a video tracking application.

Few key reduction-based animation tools have been reported in the literature. Gobbetti and Balaguer present one (Gobetti 95). In graphics, the more common use of functional approximation has been in curve and surface design.

The term "Motion Signal Processing" is typically attributed to Bruderlin and Williams who published a paper with that title at SIGGRAPH 95 (Bruderlin 95). Many of the simpler signal processing methods had been in use in practical settings and simply had not been described in the academic literature. Litwinowicz's Inkwell system (Litwinowicz 91) used a variety of filtering operations to create effects on motions. Perlin's dancer (Perlin 95) demonstrated the applicability of blending motions together to create interesting human motion, as well as the utility of adding noise to a motion to make it more "alive." Blending was probably in use in video game settings before Perlin's work. Similarly, the motion-displacement maps introduced by Bruderlin and Williams (which were simultaneously introduced as Motion Warps by Witkin and Popvic (Witkin 95)) were available as motion layering in some early character animation systems. An extensive use of Motion Blending is described by Rose et al (Rose 98) which blends between many motions simultaneously.

The idea that frequency content has specific meaning in terms of the content of a motion was described by Unuma et al (Unuma 95). Other work, such as Witkin and Popovic (Witkin 95), proposes that high-frequencies provide the details that give motion its character, but do not place

specific meaning on the various frequency bands. This view leads to the multi-resolution tools for motion editing, such as the Spacetime editing methods of Gleicher (Gleicher 97, Gleicher 98), or of Lee and Shin (Lee 99).

For a slightly more thorough discussion of Signal Processing, especially filtering and sampling used in SECTION, consult APPENDIX. The Appendix also suggests introductory texts that can provide a more thorough introducion. While time warping and other temporal transformations are common in animation systems, Bruderlin and Williams (Bruderlin 95) introduced the use of dynamic time warping for human animation problems.

The constraint-based approach to motion editing discussed has been presented by Gleicher, first in the context of interactive editing (Gleicher 97) and then in the context of motion retargetting (Gleicher 98). An earlier version of the work was presented by Gleicher and Litwinowicz (Gleicher 98b). The first work to present a spacetime solution for a human motion problem was Rose et al. (Rose 96) that used a variant of the approach to synthesize transitions between captured motions. Popovic and Witkin (Popovic 99) present a variant of the spacetime approach that simplifies that character such that it is possible to preserve the physical correctness of a motion. Lee and Shin (Lee 99) have presented a method that achieves similar results to the spacetime solutions by solving a large number of single-frame problems.

For details of the system architecture used to create the constraint-based motion editing system, we refer the reader to our earlier work on constraint methods for interactive systems (Gleicher 94).

(Bregler 97) *Bregler, C.[Christoph]*, *Malik, J.[Jitendra]*, **Tracking People with Twists and Exponential Maps**, *CVPR98*(8-15).

(Bruderlin 95) Armin **Bruderlin** and Lance Williams. Motion Signal Processing, Proceedings of SIGGRAPH 95, Computer Graphics Proceedings, Annual Conference Series, pp. 97-

104 (August 1995, Los Angeles, California). Addison Wesley. Edited by Robert Cook. ISBN 0-201-84776-0.

(Catmull 72) Edwin E. **Catmull**. A System for Computer Generated Movies , Proc. ACM Annual Conf., pp. 422-431 (August **1972**).

(Cohen 92) Michael F. **Cohen**. Interactive **spacetime** control for animation, Computer Graphics (Proceedings of SIGGRAPH 92), *26 (2)*, pp. 293-302 (July 1992, Chicago, Illinois). Edited by Edwin E. Catmull. ISBN 0-201-51585-7.

(Foley 90) James D. **Foley** and Andries van Dam and Steven K. Feiner and John F. **Hughes**. Computer Graphics, Principles and Practice, Second Edition, (1990, Reading, Massachusetts). Addison-Wesley.

(Goldstein 80) Herbert Goldstein. Classical Mechanics, second edition. Addison-Wesley, 1980.

(Gleicher 94) Michael Gleicher. A Differential Approach to Graphical Interaction. PhD Thesis. School of Computer Science, Carnegie Mellon University, 1994.

(Gleicher 97) Michael Gleicher. Motion Editing with Spacetime Constraints , 1997 Symposium on Interactive 3D Graphics, pp. 139-148 (April 1997). ACM SIGGRAPH. Edited by Michael Cohen and David Zeltzer. ISBN 0-89791-884-3.

(Gleicher 98) Michael Gleicher. Retargeting Motion to New Characters, Proceedings of SIGGRAPH 98, Computer Graphics Proceedings, Annual Conference Series, pp. 33-42 (July 1998, Orlando, Florida). Addison Wesley. Edited by Michael Cohen. ISBN 0-89791-999-8.

(Gleicher 98b) Michael **Gleicher** and Peter Litwinowicz. Constraint-based Motion Adaptation, The Journal of Visualization and Computer Animation, *9(2)*, pp. 65-94 (1998). ISSN 1049-9807.

(Grassia 99) F. Sebastian Grassia. Practical Parameterization of Rotations Using the Exponential Map, Journal of Graphics Tools, *3(3)*, pp. 29-48 (1998). ISSN 1086-7651.

(Hearn 97) Donald Hearn and M. Pauline Baker. Computer Graphics 2nd edition, C version. (1997, Englewood Cliffs, NJ). Prentice-Hall.

(Johnstone 95) John K. **Johnstone** and James P. Williams. Rational Control of Orientation for Animation , Graphics Interface '95, pp. 179-186 (May 1995). Canadian Human-Computer Communications Society. Edited by Wayne A. Davis and Przemyslaw Prusinkiewicz. ISBN 0-9695338-4-5.

(Kim 95) Myoung-Jun Kim and Sung Yong Shin and Myung-Soo Kim. A General Construction Scheme for Unit Quaternion Curves With Simple High Order Derivatives, Proceedings of **SIGGRAPH 95**, Computer Graphics Proceedings, Annual Conference Series, pp. 369-376 (August 1995, Los Angeles, California). Addison Wesley. Edited by Robert Cook. ISBN 0-201-84776-0.

(Lee 99) **Jehee** Lee and Sung Yong Shin. A Hierarchical Approach to Interactive Motion Editing for Human-Like Figures, Proceedings of SIGGRAPH 99, Computer Graphics Proceedings, Annual Conference Series, pp. 39-48 (August 1999, Los Angeles, California). Addison Wesley Longman. Edited by Alyn Rockwood. ISBN 0-20148-560-5.

(Litwinowicz 91) Peter C. Litwinowicz. Inkwell: A 2 ½-D animation system, Computer Graphics (Proceedings of SIGGRAPH 91), *25 (4)*, pp. 113-122 (July 1991, Las Vegas, Nevada). Edited by Thomas W. Sederberg. ISBN 0-201-56291-X.

(Liu 94) Zicheng **Liu** and Steven J. Gortler and Michael F. Cohen. Hierarchical Spacetime Control, Proceedings of SIGGRAPH 94, Computer Graphics Proceedings, Annual Conference Series, pp. 35-42 (July 1994, Orlando, Florida). ACM Press. Edited by Andrew Glassner. ISBN 0-89791-667-0.

(Perlin 95) Ken **Perlin**. Real time responsive animation with personality, IEEE Transactions on Visualization and Computer Graphics, *1 (1)*, pp. 5-15 (March 1995). ISSN 1077-2626.

(Popovic 99) Zoran Popovic and Andrew Witkin. Physically Based Motion Transformation, Proceedings of SIGGRAPH 99, Computer Graphics Proceedings, Annual

Conference Series, pp. 11-20 (August 1999, Los Angeles, California). Addison Wesley Longman. Edited by Alyn Rockwood. ISBN 0-20148-560-5.

(Rose 98) Charles **Rose** and Michael F. Cohen and Bobby Bodenheimer. Verbs and Adverbs: Multidimensional Motion Interpolation, IEEE Computer Graphics & Applications, *18(5)*, pp. 32-40 (September - October 1998). ISSN 0272-1716.

(Rose 96) Charles F. **Rose** and Brian Guenter and Bobby Bodenheimer and Michael F. Cohen. Efficient Generation of Motion Transitions using Spacetime Constraints, Proceedings of SIGGRAPH 96, Computer Graphics Proceedings, Annual Conference Series, pp. 147-154 (August 1996, New Orleans, Louisiana). Addison Wesley. Edited by Holly Rushmeier. ISBN 0-201-94800-1.

(Shoemake 85) Ken Shoemake. Animating Rotation with Quaternion Curves, Computer Graphics (Proceedings of SIGGRAPH 85), *19 (3)*, pp. 245-254 (July 1985, San Francisco, California). Edited by B. A. Barsky.

(Unuma 95) Munetoshi **Unuma** and Ken Anjyo and Ryozo Takeuchi. Fourier Principles for Emotion-based Human Figure Animation, Proceedings of SIGGRAPH 95, Computer Graphics Proceedings, Annual Conference Series, pp. 91-96 (August 1995, Los Angeles, California). Addison Wesley. Edited by Robert Cook. ISBN 0-201-84776-0.

(Witkin 88) Andrew Witkin and Michael Kass. Spacetime Constraints, Computer Graphics (Proceedings of SIGGRAPH 88), *22 (4)*, pp. 159-168 (August 1988, Atlanta, Georgia). Edited by John Dill.

(Witkin 95) Andrew **Witkin** and Zoran Popovic. Motion Warping, Proceedings of SIGGRAPH 95, Computer Graphics Proceedings, Annual Conference Series, pp. 105-108 (August 1995, Los Angeles, California). Addison Wesley. Edited by Robert Cook. ISBN 0-201-84776-0.

(Woo 99) Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. OpenGL Programming Guide. Addison-Wesley, 1999.

(Zeltzer 82)