Published in ACM Transactions on Graphics (SIGGRAPH 2003)

Layered Acting For Character Animation

Mira Dontcheva Gary Yngve Zoran Popović

University of Washington

Figure 1: While watching a real-time display of her actions, an animator performs a hopping motion, sketching the trajectory of a kangaroo.

Abstract

We introduce an acting-based animation system for creating and editing character animation at interactive speeds. Our system requires minimal training, typically under an hour, and is well suited for rapidly prototyping and creating expressive motion. A real-time motion-capture framework records the user's motions for simultaneous analysis and playback on a large screen. The animator's real-world, expressive motions are mapped into the character's virtual world. Visual feedback maintains a tight coupling between the animator and character. Complex motion is created by layering multiple passes of acting. We also introduce a novel motion-editing technique, which derives implicit relationships between the animator and character. The animator mimics some aspect of the character motion, and the system infers the association between features of the animator's motion and those of the character. The animator modifies the mimic by acting again, and the system maps the changes onto the character. We demonstrate our system with several examples and present the results from informal user studies with expert and novice animators.

Keywords: Character Animation, Motion Editing, Statistical Analysis, 3D User Interfaces, Motion Transformation

1 Introduction

Animation should come from the heart, not from the head. Current modeling and animation packages favor control over ease of use and demand a level of expertise from the user that is out of reach for all but a few highly skilled animators. Direct motion capture allows expert actors to animate human characters, but is fraught with a different set of difficulties, particularly when animating non-human characters and when attempting to edit existing motion. We present an interface that supports the many benefits of performance animation yet allows for the mapping betweeen the animator and character to be established in ways that are both flexible and easy to understand.

We had several goals in mind when designing our system. We wanted our system to have an easy-to-use and efficient interface appropriate for a novice with little training. We wanted to give the user control and the ability to create motions that have a sense of personality, often lacking in 3-d animation. Our results and user studies indicate that we have made progress towards achieving these goals.

We introduce a system that allows users to create and edit character animation by acting. A motion capture system and a set of reflective props, or widgets, provide the connection between the actor and the character, as shown in Figure 1). The motion capture system frees the animator from the confines of a mouse, a keyboard, and a limited workspace. The user animates by acting while watching a large display for instant feedback. The system requires minimal training, typically under an hour, because it innately captures users' expressiveness from their acting.

The key contributions of our work lie in how the animator's motions are mapped to those of the character, and how animation can be built upon layer by layer. The mapping from actor to character is created by a combination of explicit and implicit controls. Explicit control is used for rapid prototyping of the initial animation. In each layer, multiple related degrees of freedom (DOFs) of the character are modified simultaneously. Layering allows the animator to focus on one aspect of the animation at a time, whereas working with all aspects at once can be overwhelming. When editing, implicit control can be created by automatically inferring which DOFs the animator is trying to control. The system implicitly creates the mapping of real-world motion to that of the character for editing; the mapping may involve both spatial and temporal transformations.

The rest of the paper proceeds as follows. First, we discuss relevant related work and give an overview of the system and the interface. We then provide details of the algorithms for motion creation, motion editing with explicit transformations, and inferred motion editing with implicit transformations. Finally, we conclude with results, user studies, and future work.

2 Related Work

The research presented in this paper was inspired by previous contributions to interactive and easy-to-learn interfaces, especially those intended for novices. Existing work includes sketching outlines into 3-d shapes [Igarashi et al. 1999], sculpting surfaces, [Schkolne et al. 2001], and using plush toys for interaction with virtual environments [Johnson et al. 1999].

Many animators rely on explicit keyframing, where at a specific instant in time, they have full control of a character. The difficulty comes when animators want to express dynamic, fluid motion, particularly when multiple degrees of freedom should be manipulated in unison. We allow the user to edit multiple DOFs over a period of time, and because time is inherent in the process of acting, rather than in an axis of a graph, we retain the intuition that is lacking from keyframes and graph editors.

Motion capture has a long history in computer animation as a mechanism for transferring live motion to the digital domain. Motion capture can record the nuances that give feeling and expressiveness to motion; however, it also has several limitations. The computer-generated character may not have the same physique or dimensions as the human actor. The desired motion may be too dangerous to act and capture in real life or may involve impossible motions requiring superheroic speed or strength. We rely on motion-capture technology to support our work, but we do not tie the character's motion directly to that of the actor. This indirect interaction links us closer to the realm of puppeteering.

Puppeteering and performance animation have been shown to be powerful tools for the expert user. An animator uses a specialized input device, ranging from an exoskeleton to a force-feedback stylus, to control a character through some direct mapping of DOFs. Personality is imparted directly onto the character from the nature of the control. Our work follows on this work with the goal of enabling novice users to easily create expressive animation.

Although the earliest computer puppetry work dates back to the late sixties [Sturman 1998], modern performance animation traces back to the Jim Henson Company [Walters 1989] and deGraf and Wharman [de Graf 1989]. Following the debut of Mike the Talking Head at Siggraph 89, Protozoa [de Graf and Yilmaz 1999], Medialab [Benquey and Juppe 1997], and others deployed performance animation systems on television programs, showing the power of using acting for animation. Our work differs in several ways. Whereas existing systems use direct or preassigned mappings, we generalize the mapping between the actor and character, granting flexibility with the character and the motion-capture rig. We animate parts of the character individually using separate motion-capture takes, whereas traditionally, most parts of a character are animated simultaneously. Unlike existing performance animation systems, ours requires minimal calibration, making our tool useful for rapid prototyping and accessible to novices.

Several researchers have investigated computer puppetry. Shin, et al. developed a computer puppetry system allowing a performer to animate an entire character online [Shin et al. 2001]. They infer context from the motion, which they use to guide inverse kinematics. Our goals differ, as they want a fully markered puppeteer to create instant motion for a retargeted character, but we want an animator to operate a widget to create and edit motion in layers for an arbitrary character. Gildfind recognized the impracticality of tailoring puppetry controls for each device, character, and user [Gildfind et al. 2000]. His system creates the controls through iterations of a genetic algorithm with user-provided fitness feedback. We differ from Gildfind in that we require minimal training.

Snibbe and Levin [Snibbe and Levin 2000] explored layering of motion in the setting of 2-d abstract animation, focusing on using human movement as an interface for dynamic abstract systems. Oore et al. [Oore et al. 2002] used motion layering in their animation system, which combined direct manipulation of DOFs with physical controls as guides for more natural joint motion. We extended their layering to include additive layering, which allows for refinement of existing motion. In addition, our system allows not only direct manipulation of DOFs but also indirect manipulation of relative trajectories through the use of inverse kinematics. We feel that the layering and indirect interaction make our system easier for users, especially novices.

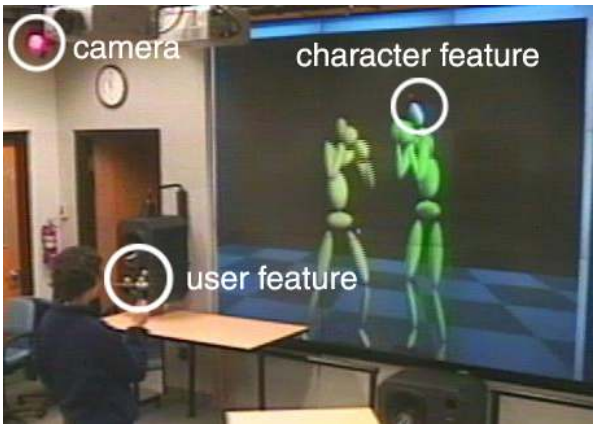

Figure 2: An animator is interacting with our system. The real-time motion-capture system tracks the widget in her hand (the user feature). The widget's motion, $X(t)$, maps to a character feature, $Y(t)$, and the user's edits are instantly reflected on the screen.

There are other related interactive interfaces for animation. Popović, et al. create precise animations of rigid bodies [Popović et al. 2000]. Perlin has a simulated actor that responds based on human intervention [Perlin 1995]. Laszlo et al. directly involve the animator in the control of interactive physics-based simulations of walking, jumping, and climbing using mixes of continuous (mouse-based) and discrete (keypress-based) control [Laszlo et al. 2000]. Donald and Henle use haptics to construct new motion by editing low-dimensional representations of existing motion [Donald and Henle 2000]. Several systems use gestures to control a performance, including the Personal Orchestra [Borchers et al. 2001] and Andy Wilson's Seagull from the 1996 Siggraph Digital Bayou.

3 Overview of System

In this section, we give an overview of the system from the hardware and software perspectives.

Hardware Our system uses an 8-camera optical motion-capture system, a wireless mouse, a wall display screen, and any number of animation widgets (see Figure 2). The system interprets each widget abstractly as a user feature. We gather motion-capture data at 120 fps and compute the translational and rotational values of each widget in real time. Each widget is built from Tinker Toys™ and includes a number of motion-capture markers. Different numbers of markers on each widget helps the system differentiate between them. A video wall displays the virtual scene and the user interface, which is manipulated with a wireless mouse. This hardware setup (see Figure 2) allows the user to move freely around and concentrate on interacting with the characters instead of the interface. Although we use a full motion-capture studio, this machinery is not essential for our system. The framework we describe can be applied to any real-time input device, including a mouse; however, interacting with multiple DOFs can be more expressive and efficient.

Space and Time Interaction Spatial correspondence between the real and virtual worlds is critical for the interactive process. Animators must be able to immerse themselves in the virtual world and forget about their physical surroundings. We create this spatial correspondence by using camera interaction, ground calibration, and explicit motion scaling.

To provide for straightforward user interaction, all user motion is transformed to the camera reference frame in the virtual world. The user can rotate and translate the camera to any viewpoint and maintain an interaction pattern that can be an exact replica or mirror image of her actions. In addition to position and rotation, camera zoom helps make the corresponding virtual motion easier to understand. For example, if the user moves closer to an object, the motion is scaled down to provide finer control over the character. As the user moves farther away from the object, the motion is scaled up, providing coarser control for sketching trajectories. We describe camera view and scale controls in more detail in Section 4.

Ground calibration serves as an absolute reference between the real and virtual worlds. Using the widgets, the animator can specify a mapping between a plane in the real world, such as a table top, and the ground in the virtual scene. Using the defined plane (table), the user can constrain a character to the ground in the virtual world.

One of the advantages of interactive animation is the user's complete control over timing. The animator observes the existing motion and reacts to it, creating synchronized motion much more efficiently. To facilitate the animator in this process, we added timing controls that include speed and interval control over the playback. Animators can isolate the region they are editing and slow down the playback speed for more precise control. To aid the user in reacting to existing motion, the system provides anticipation guides, such as ghost renderings of the character's upcoming motion and the projections of trajectory streamers onto the ground plane.

Character Interaction An animator interacts with characters through user and character features. Widgets are used to control the user features, or DOFs, which map to character features. The character feature set includes joints and points on the body. The user interacts with features through direct, indirect, and inferred mappings, which will be discussed in Sections 5 and 6. For the remainder of the paper, we use X to refer to a user feature over time and Y to refer to a character feature over time. They can be thought of as functions that take time as input and produce a vector as output, such as coordinates in space or a quaternion. The notation $X(t)$ refers to a user feature at time t.

Synthesis and Editing with Layers To create a motion using the direct or indirect mappings, the animator selects a feature on the character and then controls that feature with a widget. The widget's motion is transformed onto the character, and the changes are reflected instantaneously. The animator can then animate another feature while watching all previously animated features. This concept of layering gives the user control over timing. Different features may require different modes of transformation. We provide three modes: absolute, trajectory-relative, and additive. The animator can select the mode of transformation to find the one that feels and works best. In Section 5, we describe the different interaction modes in detail.

We edit motion in a similar fashion. A feature is selected, and the user's acting modifies the motion already present. This modification is another form of layering. For example, an animator can first define the path over which a creature will travel. The animator can then edit that path to include local sways or bounces. Performing these actions simultaneously instead can be more challenging.

Whereas the above-mentioned motion edits involve the explicit specification of the features to be modified, we also allow the user to edit a motion with implicit transformations. Instead of selecting a feature, the animator mimics some aspect of the motion while watching it, and the system infers which features to edit. The system discovers spatial and temporal mappings to associate the mimic with the features. The animator then acts again, and the edits are transformed to the character by these implicit mappings. The algorithms behind the implicit editing are explained in Section 6.

4 Interaction

To make our system appealing to novices, we aim to make the character interaction easy and flexible. To maintain the intuition of direction regardless of how a user views the scene, we transform all motion to the camera reference frame. We adjust motion scale to grant the user freedom in interacting with the system.

4.1 Camera View Mapping

We make character interaction straightforward by mapping the reference frame of the widget motion to the reference frame of the camera view. To compute the camera transformation $\mathbf{C}$ necessary for the reference frame mapping, we use the rotational component of the camera viewing transformation. As motion-capture data is received, the system transforms it by $\mathbf{C}$. Transforming the motion to the exact viewing transformation is not always useful. Sometimes the users want to map their vertical motion to vertical motion in the scene, even if they are viewing the scene with a slight downwards angle or with an overhead view, as is the case when animating the legs of a spider. We provide this flexibility by ignoring the camera roll, resulting in a direct mapping between the vertical motion in user and virtual space.

We use the animation widgets not only for character interaction but also camera manipulation. The user can control the camera and move around the scene just as easily as animating a character. To make the camera motion fluid, we apply damped springs to the widget motions. The user can also attach the camera to a character and animate while moving with the existing motion, which is helpful if the character is moving across large distances.

4.2 Scale Adjustment

To make the interface more intuitive, we automatically scale motion based on camera parameters. In addition to the standard camera transformation, we use c_z to represent camera distance. $\mathbf{T}(c_z)$ is an extra translation in the local camera z axis (the direction of camera view), resulting in a combined camera transformation, $\mathbf{CT}(c_z)$. As the user's view (controlled by the second widget) gets closer to the character, the camera distance c_z automatically decreases, causing a linear decrease in motion scale. Similarly, as the user steps back from the character, the motion scale increases. We scale by a constant amount $\mathbf{S}(k)$ to allow for differences in user motion and display size; it is manually set and can be calibrated to map the user's comfortable range of motion to the display screen size. The final camera transformation is $\mathbf{C}' = \mathbf{S}(k)\mathbf{CT}(c_z)$.

Motion can also be scaled explicitly. Often an animator may animate the first layer of an animation by using the whole room, but then may want to switch to using a table as a reference ground plane for the following layers. To explicitly define scale, we ask the users to perform the existing animation in the space they plan to use. We then define the new scale as the mapping of the bounding box of the existing motion to the bounding box of the sketched motion. This scaling technique allows for different scale factors along the different axes, which can be very helpful for motion that varies primarily along one axis.

5 Performance Animation and Editing

Animators create and edit motion by controlling user features X that interact with character features Y, as described in the overview. The character feature set includes DOFs and relative handles. A handle is a point on a character's skeleton, traditionally used for controlling a character through inverse kinematics. The point is some place P on a bone, and its world position is the composition of the transformation matrices in the skeletal chain starting from the root, $\mathbf{T}_0\mathbf{T}_1\ldots\mathbf{T}_{n-1}\mathbf{T}_nP$. A relative handle is a point on a bone specified in the k-th coordinate frame, namely the frame of a parent bone in the hierarchy. The location is then $\mathbf{T}_k\mathbf{T}_{k+1}\ldots\mathbf{T}_{n-1}\mathbf{T}_nP$. An example of a relative handle is the fingertip of the right index finger with respect to the right shoulder's coordinate frame. In our system, the handles are prespecified in the character model.

Complex motion is created by layering animation trials, also known as motion-capture takes. We approach motion layering as a technique used over multiple features or over a single feature. When layering across multiple features, the animator layers motion by animating different body parts separately. For example, first, the user defines the trajectory of the character and then individually adds leg and arm motion. The user can also perform layering over one feature by preserving the underlying motion and adjusting it to add characteristics or style. Typically, the user starts with a rough sketch and refines it iteratively, adding detail. Because the animator is watching the existing motion while animating, the new motion is synchronized with the existing motion. Throughout the layering process, an animator may require different feature-interaction paradigms. Interaction can be absolute, trajectory-relative, or additive.

5.1 Absolute Mapping

When using the absolute mapping, a feature moves in the world coordinate system (see Figure 1). This interaction is useful for animating body parts that interact with objects in the scene, such as the legs of a character that is walking, running, or jumping. When a feature is selected for interaction, its initial position Y_0 and rotation Y_0^R are stored along with the widget's position X_0 and rotation X_0^R. The new position $Y'(t)$ of a feature at time t is the sum of the initial position Y_0 and the change in translation of the incoming data over time interval $[0,t]$, $X(t)-X_0$.

$$Y'(t) = Y_0 + \mathbf{C}'(X(t)-X_0) \qquad Y^{R\prime}(t) = \mathbf{C}'X^R(t)(X_0^R)^{-1}Y_0^R$$

$\mathbf{C}'$ is the camera matrix that orients the motion so the user maintains a consistent interaction pattern throughout the animation process. The new rotation $Y^{R\prime}(t)$ of a feature is the product of the initial rotation Y_0^R and the difference in rotation over time t, $X^R(t)(X_0^R)^{-1}$. When using absolute features, the user can interact with the specified ground plane, allowing for easy animation of feet placement. If $X(t)$ lies on the prespecified ground plane, $Y'(t)$ is snapped to the virtual ground plane via a penalty objective.

5.2 Trajectory-Relative Mapping

This mapping allows for trajectory-relative motion of a selected feature. This paradigm is useful when the user wants to animate a feature, such as the head, with reference to some existing parent motion. The new position of each feature $Y'(t)$ is the sum of the initial position Y_0 and the transformation $\mathbf{K}(t)$ of the change in the widget's translation over time interval $[0,t]$, $X(t)-X_0$.

$$Y'(t) = Y_0 + \mathbf{K}(t)\mathbf{C}'(X(t)-X_0)$$

$\mathbf{K}(t)$ is the composition of the transformation matrices of a prespecified parent hierarchy at time t. This hierarchy can include just the character root or any additional joint transformations, such as the torso. The user still interacts naturally with handles, while keeping the hierarchical formulation common to joint angles.

5.3 Additive Mapping

The additive mapping is useful when the animator wants to preserve the underlying animation and add more detail or personality. This interaction paradigm provides explicit editing controls for existing motion. The technique adds differences in widget motion to the existing animation. If the animator does not move, then the object motion does not change. The new handle position $Y'(t)$ is the sum of the existing animation value $Y(t)$ and the change in the widget's translation at time t, $X(t)-X(t-dt)$.

$$Y'(t) = Y(t) + \mathbf{C}'(X(t)-X(t-dt))$$

Because we aim to retain the underlying animation, the additive mapping is most useful when the animator wants to make an animation more expressive or add slight changes to the existing motion.

6 Implicit Editing

In this section, we introduce a means for editing animations using an implicit mapping. We use the term implicit, or inferred, because nowhere do the users tell the computer what they desires to edit. It is the computer's responsibility to discover the mapping. Because this problem is harder and ill-posed, we impose restrictions on this form of editing, namely that the edits preserve the semantics of the motion. One should not use this tool to edit a walk into a back flip; rather one uses this tool to change the pace of the walk or the style of the arm swings.

Despite the restrictions, the implicit mapping has several advantages over explicit edits. Instead of manually selecting features to control, the animator mimics the motion to be edited, and the system discovers the features (relative handles or orientations), perhaps several. The features may even be shifted in time. In such a case, when the animator edits those features, the edits are shifted in time accordingly. The animator does not need to reproduce the coordinate frame of the original creator or guess the spatial scale of the 3-d animation. The system discovers any translations, rotations, shears, and scales necessary to map the animator's motions to the features. Latency and nonlinear temporal errors such as anticipation or slow reaction can be detected and corrected. Motions can be edited not only in space but also in time, such as accelerating the motion or adjusting the timing of footsteps.

An edit happens in two stages, each prompted by the interface. First the user mimics some aspect of the character motion. Features X and Y are built from the user and character DOFs. The system constructs spatial and temporal mappings from user features to character features and infers what the user wants to control. In the second stage, the user acts again, and the system remaps the changes from user features X' to character features Y'. The system updates constraints as needed and performs a constrained optimization to recover the character DOFs best satisfying the features Y'. The remainder of this section addresses these details.

A linear transform L maps the user's mimic X to the character motion Y with some error, $Y = L(X) + \xi(X)$. Then the user acts an edit X'. Instead of calculating the new motion $Y' = L(X') + \xi(X')$, we calculate the difference between the edit and the mimic and add the mapped difference to the character, derived by linearity:

$$\begin{aligned} Y' &= L(X') + \xi(X') + Y - L(X) - \xi(X) \\ &= Y + L(X'-X) + \xi(X') - \xi(X). \end{aligned}$$

Note that if errors in X' and X are correlated (the user made the same quirky interpretation twice), they will cancel. Also note that the user's data needs to be filtered to avoid the additive effects of high-frequency noise. Because the user's edit is augmenting the existing motion rather than replacing it, we feel that the edit will also likely preserve some aspects of the personality of the original motion. Whenever features are manipulated, they need to be aligned in time; that is, the system calculates timewarp $\mathscr{T}$ to align X with Y and timewarp $\mathscr{S}$ to align X' with $\mathscr{T}(X)$:

$$Y = \mathscr{T}(X), \qquad Y' = \mathscr{S}(Y) + L(X' - \mathscr{S}(\mathscr{T}(X))).$$

We will retain this notation for the remainder of the section.

6.1 Creating and Ranking a Mapping

Our system constructs the Cartesian products of user features X with character features Y (relative handles and orientations). Orientations are represented in a Euclidean space, discussed in Appendix A. Each pair consists of a user feature X (m–dimensional data over k time samples) and a character feature Y (n–dimensional data over k time samples). For each pair, we use the technique of Canonical Correlation Analysis (CCA) to discover a spatial mapping between the two features. We smooth the features for better performance [Ramsay and Silverman 1997]. CCA is a multidimensional generalization of the correlation coefficient. It finds affine transformations so that the transformed features have maximal correlation. Colloquially, how can two features be rotated, scaled, sheared and translated such that they match most closely? CCA takes features X and Y, zero-means them, and returns linear transformations A and B and canonical coefficients $\sqrt{\Lambda}$. A coefficient of 1 indicates perfect correlation and 0 represents no correlation. The corresponding rows in A and B are the canonical vectors [Mardia et al. 2000]. Appendix B shows how CCA is computed.

CCA has several properties that we exploit. The two inputs to CCA do not need to be the same dimension or the same rank. The user does not have to be in a certain orientation or know the differences in scales with the character. The behavior of CCA is invariant to any affine transformation. This property is invaluable because users can act motions in any coordinate frame they prefer.

We use CCA to map a user feature stream X ($m \times k$) to a character feature stream Y ($n \times k$). CCA zero-means the data and calculates linear transformations A ($m \times m$) and B ($n \times m$) such that

$$\sqrt{\Lambda} A^\top (X - \mu_X) \approx B^\top (Y - \mu_Y).$$

Given the differences $X' - X$ between the two user motions (recall that X' and X need to be temporally aligned before differencing), we can solve for the changes to the character features that were distilled,

$$Y' = \mathscr{S}(Y) + B^\top \backslash \left(\sqrt{\Lambda} A^\top (\mathscr{S}(X' - \mathscr{T}(X))) \right).^1$$

The original feature Y and the remapped differences added together yield the new character feature Y'. When the errors in aligning X' with $\mathscr{T}(X)$ are too large, we do not difference and directly calculate

$$Y' = \mu_Y + B^\top \backslash \left(\sqrt{\Lambda} A^\top (X' - \mu_X) \right).$$

6.2 Distilling the Mappings

Our algorithm takes the user features and the character features and runs CCA on all pairs, resulting in hundreds of potential mappings, each ranked between 0 and 1. Because of spatial and temporal errors from both the user and the input devices, the canonical coefficients of the features may vary over repeated experiments. In addition, sometimes the user may want to control multiple DOFs, such as two limbs, with just one input. We need a way of analyzing the many possible mappings from the Cartesian products and distilling the information from the best ones, with some degree of consistency over multiple trials.

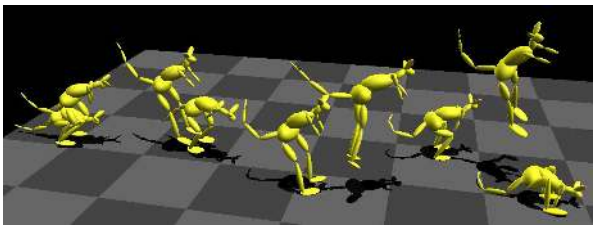

Figure 3: This kangaroo animation was created with six layers and was completed in twenty minutes. The first layer specified the kangaroo's trajectory. The second layer added the bending of the legs. Subsequent layers animated the torso, head, arms, and tail.

First, we eliminate mappings that are worse than a threshold (ten percent) from the best mapping. We distill the remaining mappings by eliminating features that are highly dependent on each other. Relative handles sharing kinematic subchains reduce to a single feature by favoring a handle towards the end-effector and a coordinate frame near the root. For example, the hand in the shoulder frame is preferred, and the forearm in the shoulder frame discarded. The hand in the shoulder frame is preferred, and the hand in the elbow frame discarded. The world coordinate frame is treated specially because the root translation does not coincide with any natural joint; only the best mapping with a world coordinate frame is retained.

Although these heuristics are far from robust, they work well in practice. The motion mappings are solved before the user edits the motion, and the active limbs on the character are highlighted. The user can choose to redo the mimicking X if not satisfied with the inferred mappings. Noise might saturate the subtle frequencies that discern one coarticulated feature from another. A poor mimic X can result in a poor mapping $Y \approx L(X)$ with much error.

6.3 Motion-Curve Subspaces

While testing our system, we discovered that when a user's motion is primarily 1-d or 2-d (or twisting rather than tumbling), the data for the underused dimensions can have non-Gaussian noise, which can lead to poor mappings. We calculate the principal components of the user features and remove the subspaces that contribute minimally to the data. When remapping an edit, the same subspaces are likewise removed. Not only does this projection improve the performance of the system, but it grants the animator the ability to selectively edit on a subspace (by no means axis-aligned) of a feature; e.g., the animator could ignore horizontal motion and only mimic and edit vertical motion. When mapping a lower-dimensional edit to a higher-dimensional feature, the system uses projections of the original higher-dimensional feature to fill the missing dimensions.

For the differencing and direct equations presented in Subsection 6.1, if B is rank-deficient, there is no unique solution to the nonhomogeneous linear equation, because adding any vector y in the nullspace ($B^\top y = 0$) to an existing solution will also be a solution. However, this setback is actually a gain. We calculate the rowspace projector $R = B(B^\top B)^{-1} B^\top$ and the nullspace projector $I - R$. Using these projectors, the edits from the user are remapped onto the rowspace, and the aligned character data $\mathscr{S}(Y)$ fills the nullspace. The projectors enable the user to act in a subspace while the system hallucinates sensible data for the remaining dimensions.

[1] We borrow the notation $M\backslash y$: find x given the linear system $Mx = y$.

6.4 Timewarping

CCA solves for a spatial mapping, but temporal mappings may be necessary as well. One could design a functional optimization that solves for affine spatial mappings and smooth temporal mappings simultaneously, but it would take too long, and we require an interactive system. First we calculate CCA on phase-shifted features to account for a constant latency. Because of the closed-form solution, running CCA for all arrangements of features and phase shifts takes under a second. We then take the top-ranked mapping and solve for the timewarp $\mathscr{T}$ that maximizes the correlation between $\mathscr{T}(X)$ and Y. This timewarp aligns constraints and accounts for higher-order errors such as anticipation or slow reaction, discussed in Appendix C. The user features are timewarped, and the spatial mappings are recalculated. We also calculate the timewarp $\mathscr{S}$ for aligning the semantics of the aligned mimic $\mathscr{T}(X)$ to the edit X'. If the user's feature corresponds to multiple character features, the remaps are shifted according to the phase offsets detected at first.

6.5 Constraints and Solving for the New Motion

We desire our edits to preserve positional constraints, such as to keep feet from sliding. Such a constraint pins a handle to a location for some interval in time. When the user performs an edit and does not change the semantics of the motion, the number of constraints remains the same, but they may move in both space and time. If we allowed the number of constraints to change, it would be harder, likely indeterminable, to deduce correspondences between the constraints. The system first decides if constraints are affected by the edit. If the user is manipulating the arms, there is no need to worry about constraints on the feet. The timewarp $\mathscr{S}$ between the aligned mimic $\mathscr{T}(X)$ and the user's edit X' warps the positional constraints in time. For each edited constraint, we sample its handle over the active interval. Inverse kinematics on the edited features determines the handle's values, and the constraint is pinned to their mean.

Our system takes the user and character DOFs and constructs features. It discovers mappings from user features to character features and remaps edits of user features to the character features, modifying constraints as needed. The final step is to solve for the character DOFs that match the edited character features, while maintaining constraints. Quaternions are remapped directly. When features and constraints are remapped by a world-frame mapping, the root translations are adjusted to minimize error with the constraints and the desired features. In some cases, these changes alone may be enough to generate good motion because the semantics of the motion get preserved. These manipulations help because nonlinear solvers are often sensitive to initial conditions. We then solve the inverse kinematics as a constrained optimization[Popović and Witkin 1999]. The constraints include positional constraints and joint bounds. The objective includes desires to match the new character features, to match the original motion, and to be smooth.

7 Results

We used a variety of characters with arbitrary rigs to evaluate the versatility and effectiveness of our framework. The animators started by roughly sketching the trajectories of the motions and then added details in subsequent layers. The playback speed was usually slowed down to give the animator more control over the character. To be effective, animators had to be aware of the speed of the animation to avoid moving too quickly and creating spastic and unrealistic motion. The duration of the animation process depended on the speed of the playback for the addition of layers and the number of times a layer was performed before the animator was satisfied.

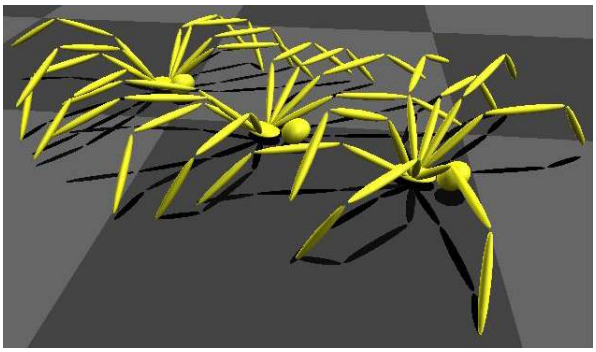

Figure 4: This spider animation was created with six layers and was completed in twenty minutes. Pairs of legs were animated as separate layers. Finally, the user added torso and head motion.

Synthesis Using a combination of the performance animation and explicit editing techniques described in Section 5, we created animations of a bird, kangaroo, and a spider. The bird was created in two layers, and the animation process lasted only two minutes. First, the trajectory of the bird was defined using the absolute mapping. Then, the user animated the wings with the trajectory-relative mapping. The camera was attached to the bird trajectory, allowing the user to animate while flying with the bird. This view was useful when the bird was difficult to animate from a static view.

We animated a kangaroo jumping in a zig-zag pattern using several interaction paradigms. The animation consisted of six layers and was produced in twenty minutes. The animator used the absolute mapping to sketch the trajectory of the kangaroo in a jumping zig-zag pattern, as shown in Figure 1. In the second layer, the animator bent the legs for the jumping motion. A table was used as a reference for the ground plane. The iterative process continued with the tail, arms, and head. The torso was modified using the additive mapping to add anticipation and follow-through to the jumping. Figure 3 shows the final kangaroo motion. This pipeline is typical of how a user would animate a character with our system.

The spider animation also consisted of six layers and was created in twenty minutes. Again, the user started by defining the trajectory with the absolute mapping. Next, two of the legs were animated simultaneously with two widgets. The rest of the legs were animated in a similar manner. The choice of what legs to animate together was left to the user, and there was no restriction on the number of legs animated at once. Finally, the animator added head motion using the trajectory-relative mapping to give the spider more personality. Using the additive mapping, the animator added bounce and sway to the torso of the spider while still maintaining the walk cycle. The spider animation shows how our system can be used to create complex motion efficiently. The animator performed the motion of all eight legs with just two widgets, but her variability in step size and timing produced an appealing spider walk (Figure 4).

Explicit Editing Using our explicit editing technique, the user can also edit previously captured motion data. Using a motion-capture sequence of a broad jump, the animator exaggerated the jump and added a back flip. Both transformations were done in just one motion, taking under a minute to edit.

We also edited a two-character boxing scenario that required synchronizing two unrelated motion-capture sequences. The twelve-second animation consisted of three layers and was completed after five minutes. The animation sequence started with one boxer punching and the other bouncing in place. The animator edited the bouncing boxer to react to the punches of the other. In the first layer, the user added torso motion for absorbing stomach

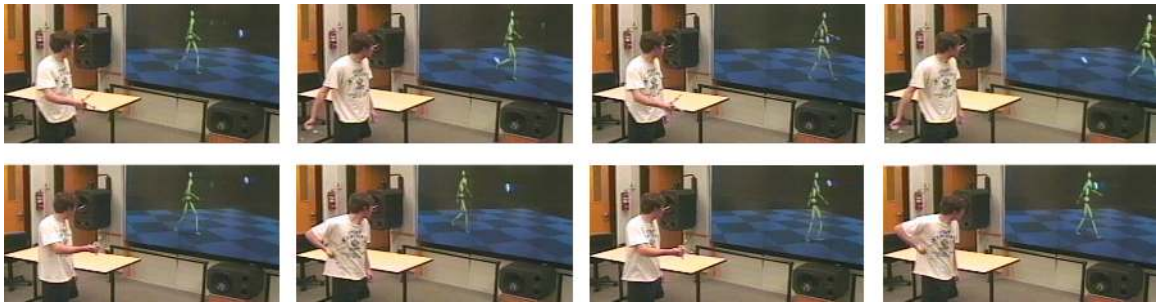

Figure 5: Top: The animator mimics the arm swing of the walking motion. Bottom: He edits by bending his elbow. The system infers that the user is controlling both arms of the character with just one arm. The motion is changed to make both arms swing with bent elbows.

punches. Next, the user made the boxer's head react to the hits. Finally, the hips were pulled back to make the stomach punches more realistic. This animation showed us how easily and effectively an animator can edit animations that require timing adjustments.

Implicit Editing We used the implicit editing technique to change the style of arm swinging in a walk sequence. First, the animator mimicked the motion seen on the screen by swinging his right arm. Then, he performed the motion differently, bending the elbow (Figure 5). The system discovered the mappings from the widget to the character's left *and* right arms and remapped the arms accordingly. The user input was treated as only 2-d because the subspace algorithm deemed the third dimension inconsequential.

We edited the world position and orientation of a character in a broad jump and a spinning jump sequence. The system mapped the user input to the character's left foot in the world frame. The feet were pinned, and the edits preserved these constraints by transforming them both in space and in time. The system calculated timewarps between the user mimic and the character features, as well as the user edit and the user mimic, to align the semantics of all the motions. Although the timing of the jumps changed, characteristics such as the arm swings were timewarped to preserve the semantics. In the spinning jump edit, the animator changed the direction of the character spin. With just one widget, the animator simultaneously controlled both translation and orientation, six DOFs, with ease.

We also edited the spider walk described above. The animator mimicked one leg with respect to the spider body. The system discovered that the animator controlled not just one leg, but *all* legs by determining the phase shifts relating the leg motions. In this example, the leg movement had enough error that the system reverted to directly solving for Y' from X' rather than using $X' - \mathscr{S}(\mathscr{T}(X))$. The mimic and edit were only 1-d, but the nullspaces of the spider's leg features reproduced the remaining two dimensions of the original spider motion. The legs were remapped using the detected phase shifts, enabling the animator to preserve the gait of the spider but reanimate all legs by just editing one.

Informal Experiences with Users A professional and novice animator tested the system we have described. The results and feedback were exciting and generated interesting discussions.

The professional animator was eager to try our system. Although he was a classical animator, he had used motion capture prior to our session and was comfortable with the technology. His main concern with the system was the lack of precision it provided compared to more traditional tools. He was excited, however, about the possibility of using our system for the initial staging, directing, and prototyping of the desired motion. The imprecision and fluidity of the system made these initial tasks of positioning and generating rough trajectories simpler than using a system that always demands exactness. He had similar positive remarks about roughing out camera positions and camera moves through a scene.

The novice animator had no prior animation experience of any kind, but found it intuitive to act the desired behavior. She had little trouble making a mental mapping between her physical surroundings and the virtual scene. However, she found it difficult to slow down and adjust her acting to slower motion on the screen. Unlike the author, who likes to animate at slow playback speeds to ensure control over the character, the novice was more comfortable animating at close to real-time speed. After a forty-minute introduction to the system, she took twenty minutes to produce a fairly successful, although stylized, spider walk, seen in the video.

We had other users interact with the system, and their experiences suggest that acting is an easy-to-learn interface for animation. In a production setting, the ability to sketch rough character and camera paths quickly can make the initial stages of storyboarding and staging faster. Directors often have trouble directing virtual scenes because it is difficult to move the camera interactively. Using our system, these tasks become faster and more natural.

8 Discussion and Future Work

We had several goals in mind when designing our system. We imagined an easy-to-use system that would also grant the animator great control. The system would require little training and would allow the user to create motions with personality. Some of our goals were conflicting; generally, one sacrifices control for the expert by making the system approachable for the beginner. As the initial users, we often felt this conflict when making decisions in designing and using the system. It was encouraging, however, that a novice could create an interesting animation sequence after very little experience with the system. We were pleased that the system could offer benefits, albeit of a different sort, to the professional.

Many questions remain on how we could improve the system. Currently, the animator uses a wireless mouse on buttons and sliders to control speed, motion-editing sections, camera view, and character selection. These operations often slow down users and distract them from interacting with the virtual scene. We hope to minimize mouse interaction to keep the user interface within the acting paradigm. Animations requiring interactions with physical objects, such as the ground, proved more difficult because of the tug-of-war between satisfying constraints and giving the user complete control. The system as it stands is tailored neither for the absolute beginner nor for the expert; we expect that further customization for either purpose would increase the applicability of our framework.

We found that animating complex motions required longer training and greater animation skills. As an example, we attempted animating a quadruped (dog) walk, and although the leg motion was constrained and the paws were not sliding on the floor, the walk did not exhibit the required physical likeness that is necessary for a good dog animation. Creating such an animation requires more training on how to make a quadruped look expressive and realistic.

Editing through inferring mappings still presents many open questions. The initial system we have presented for editing works well in many cases. However, we feel we have presented as much an open problem as a complete set of solutions. We tried to make as few assumptions as possible yet still had to require that edits preserve the semantics of the motion.

Distilling the large set of possible mappings to the essence of what the animator wants continues to be a hard problem. Whereas CCA degrades smoothly, distilling degrades discretely. The use of thresholds makes the problem less stable than it should be. We are working towards overcoming these issues while also expanding the system's capabilities. Although the mappings explored so far were from motion to motion, we plan to investigate higher derivatives or frequencies [Pullen and Bregler 2000]. Moving the other way, we plan to explore whether imposing simple constraints such as always mapping y-up to y-up would provide more stability without imposing too much on the user's sense of freedom. We also want to leverage that sometimes the system does something reasonable but not what was intended; e.g. with features $X(t) = \sin(t)$ and $Y(t) = -\sin(t)$, both $Y(t) = -X(t)$ and $Y(t) = X(t-\pi)$ are valid.

The reader may have wondered why we would design a system for novices that requires an expensive motion capture system. Although our approach currently relies on a full motion-capture studio, this machinery is not essential for our system. To capture six DOFs, one can use a data glove, stereo vision, or any other motion capture device [Oore et al. 2002]. We feel that it is reasonably likely for cheap 3-d input devices to become available in the near future.

Acknowledgements We thank David Hunt, Maya Rodrig, and Steve Capell for spending time with the system and providing us with valuable feedback. Special thanks go to Emre Yilmaz and Michael Cohen for the invaluable discussions on existing work and future ideas. We also thank Michael Cohen for helping us with the text. We thank Karen Liu and Keith Grochow for mocap data and Eugene Hsu for real-time IK. We thank Werner Stuetzle for suggesting a statistical approach to implicit mapping. Funding and research facilities were provided by the UW Animation Research Labs, NSF grants CCR-0092970 and EIA-0121326, two NSF graduate research fellowships, and gifts from EA, Sony, and MSR.

A Quaternions and Affine Transformations

To operate on orientations, we need them in a Euclidean space, the log map of quaternions. We also use the map to smooth orientations [Lee and Shin 2002], which are noisy. Note how rotations and scales in R^3 map to quaternions by the exp map. Scaling changes twist but not the axis of rotation; rotating changes the axis but not the twist.

B CCA

Canonical Correlation Analysis finds transformations A and B such that the correlation of transformed, zero-meaned features $A^\top X$ and $B^\top Y$ is maximized:

$$\arg\max_{A,B} \frac{A^\top \Sigma_{XY} B}{(A^\top \Sigma_{XX} A B^\top \Sigma_{YY} B)^{1/2}},$$

where Σ_{XX}, Σ_{YY}, and Σ_{XY} are the co- and cross-variances. CCA can be posed as a generalized eigenvalue problem to find A and B quickly [Mardia et al. 2000].

$$[S, \Lambda] = \text{eig}\left(\Sigma_{XX}^{-1}\Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{XY}^\top\right)$$
$$A = S, \qquad B = \Sigma_{YY}^{-1}\Sigma_{XY}^\top A$$

C Timewarping

The system extracts positional constraints from the user's initial motion and the user's editing motion. These constraints are intervals in time when the user feature is still. Because the user is not changing the semantics of the motion, there are the same number of constraints in both of the user's motions. We solve for a piecewise-linear timewarp that exactly aligns the endpoints of the constraint intervals and minimizes an objective.

The motivation for the objective is that in an attempt to faithfully reproduce a feature of the original motion, the user may have errors both in space (the user twitched) and in time (the user lagged or anticipated). We make the assumption that temporal errors are low frequency, so by using a sparse number of control points in the timewarp, we minimize our interference with the spatial component of the motion. We select a small number of control points t_i by analyzing the speed of the feature curve. We choose "peaks," local maxima that are a standard deviation above their surroundings. We want to find the best timewarp consisting of these control points that minimizes the error between the two features. The error is defined piecewise (as edges), and the problem can be formulated as a shortest-path problem for an efficient global solution. We optimize T_i such that the control points $t_i \to T_i$ minimize the error

$$E = \sum_{i=0}^{n-1} \int_{T_i}^{T_{i+1}} \left| u\left(\frac{(T_{i+1}-T)t_i + (T-T_i)t_{i+1}}{T_{i+1}-T_i}\right) - v(T)\right|^2 dT,$$

where u and v are the spatially aligned, dominant subspaces of the two features. Because the curves are sampled, one must evaluate E with care to avoid aliasing.

References

BENQUEY, V., AND JUPPE, L. 1997. The use of real-time performance animation in the production process. In *ACM SIGGRAPH 97: Course Notes*.

BORCHERS, J. O., SAMMINGER, W., AND M UHL AUSER, M. 2001. Conducting a realistic electronic orchestra. In *ACM UIST 2001*.

DE GRAF, B., AND YILMAZ, E. 1999. Puppetology: Science or cult? *Animation World 3*, 11.

DE GRAF, B. 1989. Notes on human facial animation. In *ACM SIGGRAPH 89: Course Notes*.

DONALD, B. R., AND HENLE, F. 2000. Using haptic vector fields for animation motion control. In *Proceedings of IEEE Int. Conf. on Robotics and Automation*.

GILDFIND, A., GIGANTE, M. A., AND AL-QAIMARI, G. 2000. Evolving performance control systems for digital puppetry. *The Journal of Visualization and Computer Animation 11*, 4, 169–183.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3d freeform design. *Proceedings of SIGGRAPH 99*, 409–416.

JOHNSON, M. P., WILSON, A., KLINE, C., BLUMBERG, B., AND BOBICK, A. 1999. Sympathetic interfaces: Using a plush toy to direct synthetic characters. In *Proceedings of CHI*, 152–158.

LASZLO, J., VAN DE PANNE, M., AND FIUME, E. L. 2000. Interactive control for physically-based animation. *Proceedings of SIGGRAPH 2000*, 201–208.

LEE, J., AND SHIN, S. Y. 2002. General construction of time-domain filters for orientation data. *IEEE Transactions on Visualization and Computer Graphics 8*, 2, 119–128.

MARDIA, K. V., KENT, J. T., AND BIBBY, J. M. 2000. *Multivariate Analysis*. Academic Press.

OORE, S., TERZOPOULOS, D., AND HINTON, G. 2002. Local physical models for interactive character animation. *Computer Graphics Forum 21*, 3, 337–346.

PERLIN, K. 1995. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics 1*, 1, 5–15.

POPOVIĆ, Z., AND WITKIN, A. P. 1999. Physically based motion transformation. *Proceedings of SIGGRAPH 99*, 11–20.

POPOVIĆ, J., SEITZ, S. M., ERDMANN, M., POPOVIĆ, Z., AND WITKIN, A. P. 2000. Interactive manipulation of rigid body simulations. *Proceedings of SIGGRAPH 2000*, 209–218.

PULLEN, K., AND BREGLER, C. 2000. Animating by multi-level sampling. *Computer Animation 2000*, 36–42.

RAMSAY, J. O., AND SILVERMAN, B. W. 1997. *Functional Data Analysis*. Springer.

SCHKOLNE, S., PRUETT, M., AND SCHRÖDER, P. 2001. Surface drawing: Creating organic 3d shapes with the hand and tangible tools. In *Proceedings of CHI 2001*, 261–268.

SHIN, H. J., LEE, J., GLEICHER, M., AND SHIN, S. Y. 2001. Computer puppetry: An importance-based approach. *ACM Transactions on Graphics 20*, 2, 67–94.

SNIBBE, S., AND LEVIN, G. 2000. Interactive dynamic abstraction. In *Proceedings of NPAR*, 21–29.

STURMAN, D. J. 1998. Computer puppetry. *Computer Graphics and Applications 18*, 1, 38–45.

WALTERS, G. 1989. The story of waldo c.graphic. In *ACM SIGGRAPH 89: Course Notes*.