Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2005)
K. Anjyo, P. Faloutsos (Editors)

AER: Aesthetic Exploration and Refinement for Expressive Character Animation

Michael Neff† and Eugene Fiume

Department of Computer Science, University of Toronto

Abstract

Our progress in the problem of making animated characters move expressively has been slow, and it persists in being among the most challenging in computer graphics. Simply attending to the low-level motion control problem, particularly for physically based models, is very difficult. Providing an animator with the tools to imbue character motion with broad expressive qualities is even more ambitious, but it is clear it is a goal to which we must aspire. Part of the problem is simply finding the right language in which to express qualities of motion. Another important issue is that expressive animation often involves many disparate parts of the body, which thwarts bottom-up controller synthesis. We demonstrate progress in this direction through the specification of directed, expressive animation over a limited range of standing movements. A key contribution is that through the use of high-level concepts such as character sketches, actions and properties, which impose different modalities of character behaviour, we are able to create many different animated interpretations of the same script. These tools support both rapid exploration of the aesthetic space and detailed refinement. Basic character actions and properties are distilled from an extensive search in the performing arts literature. We demonstrate how all high-level constructions for expressive animation can be given a precise semantics that translate into a low-level motion specification that is then simulated either physically or kinematically. Our language and system can act as a bridge across artistic and technical communities to resolve ambiguities regarding the language of motion. We demonstrate our results through an implementation and various examples.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Animation

1. Introduction

Compelling characters are at the heart of most animated productions, yet it remains a significant challenge for animators to create rich, nuanced characters with unique personalities that will engage audiences. There are few tools that directly support the creation of expressive character movement – those aspects of movement that convey personality and mood. This paper presents an animation framework designed to fill that need, along with an initial implementation. The focus is on exploring deeper conceptual design issues, rather than a particular interface.

Our system is designed to support the authoring of expressive skeletal character motion. In order to do this effectively, the system attempts to achieve three subgoals. It seeks:

- to allow animators to control aesthetically meaningful aspects of motion directly,
- to allow animators to work at various levels of abstraction, supporting both exploration and refinement,
- to support the creation of *specific* characters.

Two themes arise from these goals: the need for *customization* and the need for *explicit representations*. Customization arises from the desire to create specific characters. A specific character shows emotion in a specific way. Systems that create a generic “sad" or “happy" movement are likely to be insufficient for production work. We believe customization is best achieved through direct animator control. Thus from a computer science perspective, we are faced with the problem of balancing the conflicting needs for generic, reusable

† {neff|elf}@dgp.toronto.edu

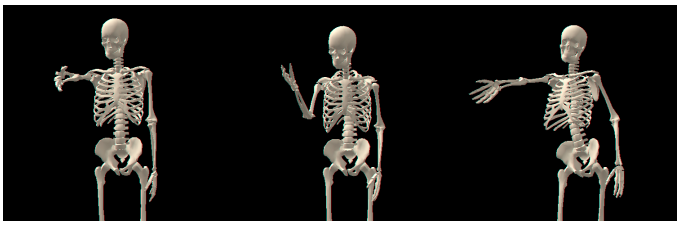

Figure 1: *Three frames from an animation in which a character shows an object to the audience. Left: from the base animation. Middle: an "old man" character sketch has been applied. Right: beauty shape edits and an "energetic" character sketch have been applied.*

classes of characters and behaviours, and for customization and specificity.

The second theme is the use of *explicit representations*. If movement properties are explicitly represented in a comprehensible manner, their definition can be made precise, and they can be customized, edited and refined. Building a high-level tool out of well defined low-level building blocks allows both the high-level and low-level interface to be exposed appropriately to the animator, supporting rapid, exploratory, high-level adjustments early in the process and low-level refinements as needed.

Movement properties, or *properties* for short, are the building blocks of our system. To understand the aesthetic nature of movement, we turned to the arts literature and distilled a list of key movement properties. Many of these properties have been implemented, along with a software architecture that allows these properties to be combined and applied to movements. Together, the properties and architecture form the core of our system and provide an appealing workflow.

Animators create animation sequences by applying and then adjusting properties attached to actions in a script. Because properties directly vary aesthetically important aspects of movement, they are more expedient to use than a keyframe system, while still providing fine control at an appropriate level. Properties also range from high-level effects that vary a large set of motion parameters, such as *CreateRecoil*, to specific low-level controls, like *SetDOFValue*. High-level properties derive from low-level properties, and all properties in our system are procedural.

The animator's workflow supports both exploration and refinement. An artist begins by listing a set of actions in a script which define an initial animation sequence. She can make global changes to the animation by applying different *character sketches* to the script. Sketches, written using properties, outline basic character features such as a slow tempo, hunched back and tense shoulders. By switching sketches, an animator can quickly experiment with various broad characterizations. She can then refine the sequence by adding more properties and adjusting the action descriptions, character sketch and existing properties as needed.

To make our discussion more concrete, throughout the paper we will refer to how a simple animation sequence is specified and refined. There are three simple movements in the sequence: a character reaches for an object, brings it close to him to inspect it, and shows it to the audience.

Our current implementation focuses on the creation of realistic, skeletal, humanoid character motion over a limited range of standing movements, including gestures, posture changes and balance adjustments. Our system generates both kinematic and dynamic motion based on forward simulation. Although limited, this range of movement demonstrates considerable expressive variation while also avoiding dynamically challenging movements.

2. Background

The literature describes many previous animation systems, often with a procedural component. Early systems, such as *Scripts and Actors* [Rey82], provided a language for defining modeling and animation. Independent *actors* control different visible elements in a scene and are invoked every time step. *MENV* [RLO90] controls animation by changing the value of *avars*, articulated variables which control various aspects of the model. *Improv* [PG96] combines two procedural components: a Behavior Engine for deciding what actions a character should perform, and an Animation Engine for controlling these movements. Actions can be layered and movement transitions are normally specified using either sinusoids or different frequency noise functions [Per95]. Blumberg and Gaylean [BG95] present a system for directing autonomous creatures, but the emphasis is on action selection, not the expressive aspects of how the motion is performed. Our work is distinguished by its focus on direct control of aesthetically important qualities of animated characters, an iterative tool designed to support exploration and refinement, and the integration of dynamic simulation.

A significant body of work shares our emphasis on expressive aspects of motion. *EMOTE* [CCZB00], like our work, develops explicit models based on the arts literature, using Laban's Effort-Shape model to vary the manner in which a movement is performed. In previous work, we have presented various tools for modifying specific aspects of motion that are the building blocks for this system [NF02, NF03, NF04]. Work on expressive transforms ([UAT95, ABC96]) attempts to extract emotional content

from a piece of captured motion into a transform that can be applied to different motion. Style Machines [BH00] support high level editing of a motion by learning a style from captured motion, such as ballet or modern dance, and applying this style to other sequences. Pullen and Bregler [PB02] work at a similarly high level, allowing an animator to specify key frames and using a statistical model drawn from motion capture data to texture the key-framed motion. Approaches based on learning or parameter extraction cannot guarantee that learned animation will remain physically consistent. Rose et al. [RCB98] present a data-driven system in which a given action (*verb*) is captured whilst performed several different ways (*adverbs*); new motion can be generated by interpolating these samples. Bruderlin and Williams [BW95] adjust captured motion by treating movement as a signal and adjusting the gains of various frequency bands.

Two main approaches have been used for generating physically based character animation. Spacetime constraints take the laws of Newtonian mechanics as constraints and solve for motion using an optimization process (see for example [WK88]). Forward dynamic simulation generates forces at each time step, integrating Newtonian laws of motion to update the character state. Our work takes the latter approach, building on previous efforts in hand tuned control for force generation (see for example [HWBO95, FvdPT01]).

The prototype system described here was built on the freely available Dynamic Animation aNd Control Environment (DANCE) [NTHF]. Our work makes considerable use of kinematic and dynamic simulation and control techniques.

3. Arts Background

A broad literature survey in theatre, animation and movement theory was conducted to better understand what aspects of motion are expressively salient. A set of aesthetic movement properties was developed based on this analysis. The properties were grouped into three broad categories: those that affect character *shape* or pose; those that relate to *timing*; and those that relate to *transitions*, or how a character moves from one pose to another. The full taxonomy is available in [Nef05] and is summarized below.

Shape deals with the poses a character strikes over time. *Balance* adjustments are a very important expressive aspect of shape. They should be amplified in a performance setting and act to increase the intensity of movements [Bar91b]. According to Laban, the three principal components of trunk movement are rotational movement about the length of the spine; “pincer-like” curling from one or both ends of the trunk and “bulge-like” shifting of the central area of the trunk out of its regular position [Lab88]. The “beauty line", a large S-curve that snakes through a character’s body and lends a very sensual appearance, is also important [Bar91a]. Forward and backward collar movement can open or close the chest. Upward movement adds intensity to a pose and downward movement can suggest relaxation, dejection or exhaustion [Sha63]. *Extent* or extension refers to how far an action or gesture takes place from a character’s body.

Transitions deal with how a character moves from shape to shape, as well as transient aspects of movement, such as the interplay of tension and relaxation. Following [Las87], we use transition curves to warp the timing of a motion, also referred to as the *motion envelope*. Tension and relaxation is widely reported [Lab88, Sha63, Bar91a] as an important movement property. When doing physical simulation, we use tension changes as an intuitive way to warp the motion envelope, to control end effects like overshoot and pendular motion, and to vary the effect of external forces.

The two main components of timing are *tempo* and *rhythm* [Lab88, Moo84, Sta49]. Rhythm refers to the overall beat structure of a set of movements. A sample rhythm could be long, long, short, repeat. Tempo refers to the speed at which motions are completed; the speed of the beat. Rhythm and tempo are independent and can have a strong effect on inner mood, ranging from excitement to melancholy [Sta49]. *Successions* deal with how a movement passes through the body and are another crucial timing property. They help generate a sense of flow [Sta49].

Our list of movement properties is extensible and does not rely on one particular movement theory. An animator is free to employ whichever properties she finds most useful. It is worth noting that psychological research suggests that people have “style factors”, such as a certain tempo, which are consistent across all the normal activities in which they engage [Gal92]. The character sketch builds on this idea.

4. Workflow Overview

User interaction in the system follows a basic Perform-Review-Update iteration cycle. The system is given a set of instructions and generates an animation sequence (Perform). The animator then reviews the animation and considers possible adjustments (Review). The animator can then issue additional instructions and generates a new animation sequence (Update). This sequence is repeated until the animator converges on the desired movement sequence. Early in the process, an animator will normally make broad changes as he explores different possibilities and later make fine adjustments as he converges on a final animation.

An animator progresses through the following steps in order to generate an animation sequence:

1. The animator provides a motion script, character sketch and additional edits to specify the animation. (Section 6)
2. The system develops an executable motion plan based on these instructions. (Section 7)
3. A dynamic or kinematic simulator executes the motion plan in order to generate an animation. (Section 8)
4. The animator reviews the animation and refines the script, iterating until he is satisfied with the result.

5. Basic Entities: Actions and Properties

5.1. Actions

Movement specification begins in a familiar way: with poses. Actions are the basic unit of movement in the system, and are defined hierarchically with the following action levels: *Action*, *Cycle*, *Pose*, and *DOFEntry*. A *DOFEntry* controls an individual degree of freedom of a character as a function of time. A *pose* is a set of DOFEntries. A *cycle* is a sequence of poses that can be repeated. An *action* is a sequence of cycles. Actions can control any subset of a character's DOFs. Multiple actions controlling different parts of the body can be superposed.

An action defines the structure but not the content of a movement. Specifically, an action defines which DOFs make up a pose, how many poses are part of an action, and their sequencing. The angles that define the pose, transition time, tension changes etc. are all defined by attaching properties to the action. Actions provide animators with a handle for a particular component of the motion sequence. In our example animation, each base action consists simply of a single pose to which properties are applied that define a world space constraint on the wrist, initial timing and an initial transition function.

5.2. Properties

Properties are designed to encapsulate a particular aesthetically meaningful aspect of movement. Properties provide an interpretation for actions. We have identified several classes of properties. *Base properties* directly modify low-level attributes exposed by the system. They operate by invoking commands exposed through the Base Representation and Movement Property Integrator described below. *Composite properties* combine several base properties. They provide for higher level interaction. *Generator properties* introduce new actions into the script, such as idle motions, nervous ticks, or movement sequences that are more easily defined procedurally.

An action can be viewed as a structured container for properties, which can be applied anywhere in the action hierarchy. By default, properties will propagate to lower levels in the action description. If they encounter a property of the same type at a lower level, it will be given precedence and can choose to override the higher level property or blend with it. For instance, a transition curve applied at the Pose level will cascade to all DOFEntries that are part of that pose, but if a DOFEntry already has a transition curve property applied to it, this will take precedence.

Properties are defined procedurally. Our goal is not to prescribe an exhaustive set of properties. Rather, the property set is designed to be extensible as needed by an animator or technical director. Properties can also be modified, so the manner in which a particular aspect of movement is varied can be customized for a given character. Seen in this light, developing specialized composite properties is a part of character design, much like rigging. The development of repositories of properties will facilitate re-use in future productions.

Property	*Description*
SetDOFValue	Specifies the value for a DOF
SetTension	Adjusts the amount of joint tension during a motion.
Synchronize	Sets a timing relationship between two actions.
SetDuration	Varies the transition time to a pose.
SetTransitionCurve	Varies the transition envelope.
VarySuccession	Adjusts the relative timing of joints in a motion.
VaryExtent	Adjusts the amount of space a pose occupies.
VaryAmplitude	Adjusts the angle range covered by a motion.
GenerateWeight Shifts	Generates idle weight shifting behaviour.
SetReachShape	Varies posture during a reach.
SetPosture	Varies character posture.
AdjustBalance	Alters the character's balance point.
SetRhythm	Coordinates actions to occur with a certain rhythm.
CreateRecoil	Varies all aspects of movement to recoil from point.

Table 1: *A few of the properties defined in the system.*

A representative set of properties is shown in Table 1. Note the wide range of granularity at which properties act. *SetDOFValue* will specify the desired DOF value. Properties like *VaryExtent* or *SetPosture* vary several DOFEntries within a single pose. Synchronization and Rhythm properties create relationships between multiple actions. *CreateRecoil* is a high level property that supports a range of different forms of recoil and will change a character's pose, warp the transition function, and increase or decrease his tension and tempo.

Properties take parameters that define both how they behave and how they should be combined with other properties. *SetDuration* for instance, takes a *value* and a *flag*. If the flag is set to ABS, the property will override previous *SetDuration* properties attached to the action level and set the item's duration to be *value*. If the flag is REL, the property acts in a relative manner and multiplies the action level's current du-

Channel	*Summary*
Script	Specifies a time series of actions.
Action Description	Defines the structure of a movement, generally with an initial set of properties.
Character Sketch	Generates a global change to the movement sequence.
Animator Edits	Used to fine tune an animation sequence.
Property Definitions	Provide interaction handles.

Table 2: *Interaction channels.*

ration by *value*. Each property has a precise semantics, but for the purposes of this paper, their names will hopefully be sufficiently illustrative of their function.

6. User Input

There are several channels by which an animator can interact with the system, as summarized in Table 2. They may be used at different points in the production process. New property definitions may be developed during character set up or existing properties might be sufficient. A script must always be provided to define the actions a character performs. Character sketches define properties that are typical of a particular character type, make global changes to the movement sequence and are normally applied in the early stages when an animator is exploring different approaches. Changing character sketches allows an animator to rapidly vary broad characterizations. Animator edits allow the animator to directly change, add or fine-tune properties at any point in the action hierarchy for any action in the script and can also be applied globally to all actions.

Various forms of input have been used in the example animation. Animator edits were used to refine the base motion sequence, and create a more feminine and languid sequence. An “energetic" character sketch was layered on top to vary the feel. In other examples, “old man" and “dejected" sketches were applied to the base motion sequence.

7. Generating a Motion Plan

The system uses input from all the channels described above to generate a motion plan. The motion plan is stored in a low-level *Base Representation* or *BRep*, from which a kinematic or dynamic animation can be generated. The *Movement Property Integrator* or *MPI* is the central intelligence of the system, and responsible for mapping the various forms of input into the BRep. The stages involved in this process are shown in Figure 2 and described below. In a single iteration, the system transforms the user’s input script to an animation sequence. The system *always* generates an animation.

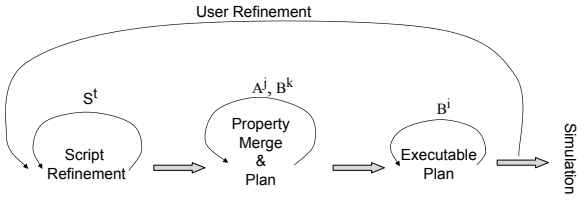

Figure 2: *Generation of an executable movement plan, showing the three stages: script refinement, property resolution and BRep refinement.*

We introduce some notation in this section to make our concepts clearer, omitting the detailed semantics (see [Nef05]). The crucial point of the notation is that it captures the essentially iterative nature of the construction of an animation.

7.1. Script Refinement

The *Script*, denoted S^i where i denotes the iteration count, defines the movements a character is to perform. It consists of a set of time ordered tracks. Each track is populated with actions. The animator defines the initial script and can directly edit it during any iteration. In our example animation, the initial script consists of the three actions: reach, inspect, display. It can also be modified by *generator properties*.

In building the motion plan, the MPI begins by applying all the generator properties which add new actions to the script. This can be represented very simply as

$$MPI : S^i \rightarrow S^{i+1} .$$

Generator properties can either be included as part of the character sketch or specified as animator edits.

The remaining edits in the character sketch and animator edit list act to attach properties to actions in the script. The MPI next performs this binding. This can be represented as:

$$MPI : S^{i+1}(\mathbf{A}^j), CharacterSketch, AnimatorEdits \rightarrow S^{i+1}(\mathbf{A}^{j+1}) ,$$

where $\mathbf{A}$ is the initial set of actions and $\mathbf{A}^{j+1}$ is the evolving set of actions as they are augmented with further properties. As an example, the old man character sketch specifies a reduced movement duration. The property that triggers this is attached to all the actions in the script at this point.

7.2. Property Resolution

Once the script has been completed and all properties have been attached, the MPI develops an executable Base Representation. We will first define the BRep and then explain how the script, actions and properties are mapped to it.

7.2.1. Base Representation

The BRep contains a track for every DOF in the skeleton. In addition, the BRep contains *signal parameter* tracks that act as blackboard space. These are used to specify continuous variation in higher level properties controlled by the system, such as a centre of mass offset for balance control or the amount of pelvic twist. Each track is time ordered

and can thus be seen as defining a time series of activities for every degree of freedom in the animation. A track $T = \langle a_0, a_1, \cdots, a_i, \cdots, a_n \rangle$ is populated with a sequence of *Transition Elements* or *TElements* denoted a_i. As discussed below, each a_i indicates (at minimum) an interval of time over which an activity operates on the specific DOF to which a given T is bound. The intervals are (currently) presumed to be disjoint, but their union may leave gaps in the timeline for that DOF, which indicates that no actions are active in that interval. We define $\mathbb{T}$ to be the set of all such sequences T.

Any instance B of a *base representation* is an m-tuple containing at minimum one track for each DOF. Thus $B \in \mathbb{T}^m$, $B = (T_1, \cdots, T_m)$. Each T_i is bound to a specific DOF in the character's state. Let $\mathbb{B}$ denote the space of all representations B.

There is currently a one-to-one correspondence between Transition Elements in the BRep and DOFEntries in the action description. TElements are labelled based on the actions that generate them, facilitating query and edit operations. TElements contain all the information necessary to generate a final animation. A Transition Element has three pieces of timing data: start time, duration and hold time, with a TElement becoming active at the *start time* and transitioning the DOF to its desired value over *duration* sec., then holding this value for *hold time* seconds. Other basic information TElements hold includes a desired value, a transition curve that specifies how the interpolation from the current value to the desired value should proceed, and data related to dynamic simulation, such as tension values. They can also be tagged with data that is used in later stages of the planning process, but is not directly used to create the final animation.

The BRep is iteratively constructed during a motion specification process. This creates a sequence of refinements of valid BReps $\langle B^0, \cdots, B^i, \cdots, B^n \rangle$, where $B^i \in \mathbb{B}$ and i indicates the i^{th} iteration on the BRep. Any such B^i may well be executable by the simulator, but B^n should be seen as a "converged" base representation. The initial base representation, B^0, is the null operation, and is defined to be an m-tuple of empty DOF track sequences. modified in the representation.

7.2.2. Property Merging and Application

The MPI generates a set of operators which regulate how the properties and actions defined in the script are mapped to the BRep. Algebraically, any such operator $\mathbf{M}$ is a mapping from the base representation together with information about the activity to be performed, to a base representation. More formally, the operator $\mathbf{M} : \mathbb{B} \times \mathbb{A} \times \mathbb{P} \rightarrow \mathbb{B}$, where $\mathbb{P}$ is the set of animation properties and $\mathbb{A}$ is the set of actions that are modified by these properties. The MPI performs a sequence of mapping operations: $\mathbf{M} = \langle M_1, M_2, \ldots, M_n \rangle$. The ordering of these operators is based on the type of property being applied as some property types need to be applied ahead of others. The current mapping order is as follows:

1. Generate signal parameters.
2. Apply shape calculator properties.
3. Apply other shape properties.
4. Apply timing properties.
5. Apply transition properties.
6. Apply properties that need to query the BRep.

Each operator normally consists of a *merge* phase followed by an *apply* phase. Composite properties are handled somewhat differently. The merge phase is the same as for other properties, but the apply phase acts to tag the actions with a new set of low-level properties rather than writing directly into the BRep. Composite properties are processed first so that the low-level properties they generate can be merged with other active properties of the same type.

7.2.3. Merge

In general, an individual action will be tagged with multiple instances of a property, such as *SetDuration*. These different versions of the properties come from the different input sources: the initial action description, the character sketch and animator edits. It is important that all the applied properties can potentially affect the final motion, rather than one property replacing all earlier properties of a given type. This allows later edits to *refine* earlier *exploratory* edits. For instance, the character sketch *adjusts* an existing movement sequence, rather than *defining* it.

Before a property is written into the Base Representation, all properties of a given type acting on a specific action level are merged. This resolution process effectively leaves a single property of each type attached to an action level. This merge process involves three steps. First, properties are pushed down to the lowest level of the action hierarchy at which they can act. This ensures that all properties of a given type will be at the same node when the merge operation is performed. Second, properties are sorted based on priority level. Priority level is based on the property source (default order: action description first, character sketch, animator edits) and the level in the hierarchy the property is applied at (higher levels being used first). Third, the sorted property list is merged. Prototype merge functions are available which support merges that overwrite, scale by a factor, average or add parameters, and properties can also define their own merge semantics. This allows the property designer to decide how best to merge a property. The *SetDuration* property supports absolute and relative duration specifications, the latter acting to scale a lower priority absolute duration. Returning to our example animation, the first action initially contains a *SetDuration* property that specifies a 1.1 sec. duration. The old man character sketch specifies a *SetDuration* property that requests scale of 1.8. After the merge, one property remains that sets the duration to be 1.98 sec.

7.2.4. Apply

Once the properties have been merged, the application process is quite straightforward. Each property defines an *apply*

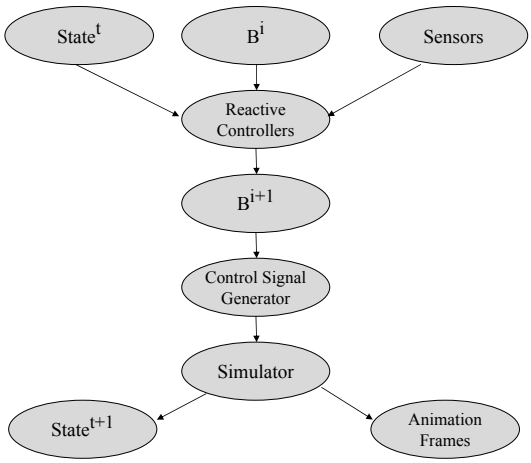

Figure 3: *Data flow for one time step of simulation.*

method. This is called for the one property of each type that is still active at an action node. The method can query the BRep, perform any necessary calculations and then writes the resultant data either into the BRep or a data store associated with one of the solvers described below. Once data has been committed to the BRep, it can be queried by future properties and filters.

7.3. BRep Refinement

Once the action-based properties have been applied, the BRep can be further refined through filtering or post-processing. In a manner similar to above, a filter $F : B^i \rightarrow B^{i+1}$. These processes no longer make any reference back to the script, working solely from the information contained in the BRep. Examples from this stage include a filter that smoothes transition curves or the Time Planner described below which enforces the timeline semantics. This is a powerful notion as it admits the full extent of signal processing to be incorporated into the framework.

8. Executing the Motion Plan

The process of executing the motion plan for one time step is shown in Figure 3. *Reactive controllers* allow the character to adjust its behaviour based on its state at the beginning of the time step. This is particularly important in dynamic simulation, where the effect of an action is not completely known ahead of time and adjustments may be needed in order to ensure that a motion completes successfully. The main reactive controller in our system provides balance control.

Reactive controllers receive the system state, the current BRep, and sensor information as input. *State* is a vector of position and velocity values for every DOF in the character. Reactive controllers can update the BRep to attempt to better achieve the requested motion. As an example, it is the balance controller that works to achieve balance adjustments, knee bends and pelvic twists, as specified in the signal parameter tracks of the BRep. Reactive controllers update the BRep by inserting short elements into the DOF tracks that change the control parameters for the current time step.

The *Control Signal Generator* takes the DOF tracks of the updated BRep as input and produces the control information required by the current simulator. For the kinematic simulator, this information consists of the value of each DOF. The dynamic simulator requires a torque for each DOF. The torque values are generated by an actuator positioned at each DOF. These actuators can either be proportional derivative controllers, such as described in [HWBO95] or an antagonistic controller variation based on [NF02]. When calculating muscle activation levels (i.e., spring gains) for each DOF, the system performs gravity compensation to ensure that positional errors are small, even when the character is moving in low tension. Gains are calculated at each pose and activation curves are stored in the BRep.

The simulator takes the input from the control signal generator and updates the character state, producing animation frames. The dynamic simulator is based on code generated by the commercial software SD/Fast [HRS94].

8.1. Balance Control

Movements near their balance limits have strong expressive impact [Lab88], making balance control particularly difficult. Two different, feedback based balance algorithms are used. The idea behind both is to calculate an error term between the location of the projection of the COM on the ground plane and the desired location for this value and use this error to adjust the ankle angles. The first balance controller is based on the algorithm of Wooten [Woo98] and provides true balance control, but has a limited stability range and supports limited lower body adjustments. The second is described in [NF04, NF05] and provides a wider range of lower body movement. To maintain stability in dynamic simulation with this controller, however, we must employ "sticky ground"; simulated springs to help hold the feet in position, thus making the balance control task easier.

9. Planners and Solvers

When designing properties, there is a trade-off between maximizing encapsulation by embedding all the functionality required to achieve a particular effect within a particular property versus encouraging code reuse by creating a common code base for functionality that will be shared by multiple properties. Both approaches are used in our prototype. Corresponding to the categorization of movement properties, the system supports a transition planner, shape solver and time planner. The latter two are described below.

9.1. Shape Solver

Solving for a character's pose can be made easier through the use of inverse kinematics and balance correction tools. It

is impractical to include such complex code within an individual property. Instead, properties should be used to dictate how these solvers behave when determining a pose.

The system includes a *body shape solver* described in [NF04, NF05] which combines balance adjustments, world space constraints and aesthetic constraints in a single hybrid IK system. We have extended the system to allow shape properties from different input sources to be blended. This allows default shape properties to be specified in the character sketch which are combined with the shape properties that are used to define an action. The solver parameters are controlled through the application of properties. Composite properties are used to encapsulate *shape sets* that describe a particular type of pose and provide an animator with a small number of parameters.

Shape calculator properties are associated with a particular pose in the action. Once they are merged, their application sets values in a data store that is accessed by the shape calculator. The calculator solves for a set of joint angles that meet the constraints and stores these angles as SetDOFValue properties attached to the pose's DOFEntries. The calculator may also add signal parameters to the action description.

9.2. Time Planner

Time planning benefits from a global view of the timeline. Properties operating on individual actions may request synchronization constraints, scaling of TElement duration and hold times, and shifting of transition elements that cascade through the timeline. A *Time Planner* implements a timeline semantics that works directly on the BRep as a post-process, making use of tag data added to the TElements by properties. All time effects are ultimately achieved by adjusting the three time properties in the TElements: start time, duration and hold time.

Scaling edits can be applied to adjust the hold time, duration or both and can act at any level in the action hierarchy. The edits are pushed down to the DOFEntry level before being applied. By default, the time planner will adjust the start time of DOFs within an action to maintain the same relative timing, providing a local scale. This can be overridden.

Relative timing between actions may also need to be maintained. For instance, a punch might need to start at the same point relative to the start of a lunge. This can be enforced by creating a link between the two actions, which instructs the time planner to maintain the same relative timing.

Placement edits shift the position of TElements in the time-indexed BRep tracks and can be applied at arbitrary action hierarchy levels. *Succession* edits are a specialized placement edit that adjust the start time of joints involved in an action as one moves outward from the base of the spine. Placement edits are applied before synchronization edits.

Synchronization edits act to either synchronize actions to each other or synchronize them to particular points in time. Three forms of action-action synchronization are supported: TElements in different actions can be set to end or start at the same time, or to maintain the same relative spacing.

The time planner implements elastic behaviour for the timeline where no TElemnts can overlap and there can be no gaps in the timeline. Once all time adjustments have been applied, if a TElement overlaps another, all TElements that are part of the overlapping pose and all TElements that occur after the end of the overlap will be moved forward in time by the amount of the overlap. If there is a gap, all future actions are shifted back so that the first TElement to start after the gap now starts at the beginning of the gap. Gaps can be created by inserting *spacer* TElements on a special track in the BRep. Spacers prevent the timeline from contracting and can be effected by time edits, but have no effect on the final animation.

10. Sample Results

Several simple animations have been produced to illustrate system functionality and are available online [Nef]. We use a skeleton for our sample animations as this focuses the viewer's attention on what can be achieved by varying skeletal motion - what we control - and avoids the distractions of varying body types and clothing.

A composite property such as *CreateRecoil* encapsulates a full set of movement properties, and is suitable for rapidly exploring a movement idea. It takes as input the location of the object to react to, a parameter that indicates one of several styles of reaction and an intensity value that indicates the vehemence of the reaction. The property will adjust the motion based on the location and alter pose, timing, and transition features based on the intensity value. The animation shows three different style recoils.

Approaches that view animation as a time series of body state vectors enforce an unnecessary correlation between all joints. In our system, actions may use a subset of the character's DOFs and be overlapped in time, allowing an animator to independently modify the movement of subsets of joints. This different form of retargeting is illustrated with two versions of simple twisting dance. The first is correlated with a slow tempo, 4/4 rhythm. Both the lower body twist and upper body twist and arm swings are aligned with each beat. The second sequence is based on a faster tempo 3/3 rhythm. The lower body twist is still aligned with each beat. The upper body movement from the first sequence is used on the third beat of each measure and its motion envelope is warped to punctuate the beat. New actions are inserted on beats 1 and 2 that simply feature a small elbow bend and head twist. The result is two very different sequences, based on the same action descriptions. Such retargeting differs significantly from what can be achieved by motion warping a state vector.

Our example animation illustrates how properties are layered together to generate a final sequence. An initial version of the animation has the basic timing and defines the reach constraints. To each pose, we then add the *beauty line* reach property which takes the reach constraint and an intensity parameter and triggers the shape calculator to determine the pose. The tempo is reduced to give the animation a more languid feel and forward successions are added to increase the sense of flow. Transition warps are used to give the animation a bit more pop, and finally, it is dynamically rendered.

A character sketch can be used to prototype different approaches to a character. We take a simple animation sequence that shows a character greeting a friend, beckoning him over and then showing him an object to the right. Two different character sketches are applied, one for an old man, one for a more energetic character. They produce two very different sequences, allowing the animator to quickly play with different styles. Due to the global nature of the character sketch, some of the adjustments may not be appropriate for some of the actions. In a subsequent pass, the animator can add additional edits to fine tune the motion, supporting refinement.

The character sketch for the old man is as follows:

```
#Scale duration
SetDuration {*} REL 1.8
#Scale hold time
SetHoldTime {*} REL 3
#provide default shape settings
SetCharShapeParam {*} 1 load oldMan.shp
#flatten the transition functions
SetTransitionFunction {*} AVG 0 0 0 0
#reduce extent
VaryExtent {*} 0.8
#add a normal succession
VarySuccession {*} normal 0.01
#reduce neck rotations
SetDOFAngle {* * * neck_y} REL 0.5
#add some shake to the forearms
SetShake {* * * forearm_x} ABS 0.017 7
```

On any row, the first term is the property to apply, the curly braces specify the recipient action level (* indicates a global edit) and the rest of the entries are parameters. The extent variation is applied post-burn in, after a pose has been solved for that may involve a world space constraint. It should not be used on actions with hard touch constraints as it will not maintain them. The rest of the properties will not interfere with constraint satisfaction. By using a different scaling on the hold and duration times, both the character's rhythm and tempo are adjusted. The old man takes longer to prepare for a movement than the default character.

Edits and sketches can be freely layered together. For instance, we combine the beauty line shape edits with the "energetic" character sketch to generate the third frame in Figure 1. The first frame shows the same pose from the base animation and the middle frames shows the "old man" sketch applied to the base animation. The default postures used with these sketches and an additional "dejected" sketch are shown in Figure 4.

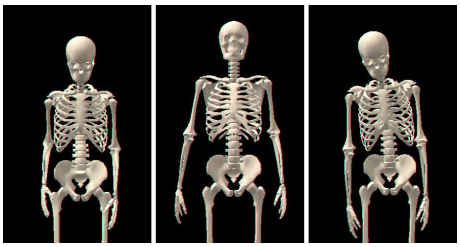

Figure 4: *Default postures for the "old man", "energetic" and "dejected" character sketches, respectively.*

11. Discussion

We have presented an approach and prototype system for generating expressive character motion. Procedurally defined properties provide direct handles on aesthetically important aspects of motion. High level interfaces like the character sketch and composite properties support rapid exploration, while animator edits and low-level properties allow for detailed refinement. Furthermore, high level properties, such as shape sets, are built out of low-level properties, allowing an animator to switch between high and low-level interfaces as needed. These mechanisms combine to meet our second sub-goal of supporting both exploration and refinement. Finally, since there is an explicit, well-defined and editable representation for every movement property and input channel in the system, these can be customized. We feel that this is important for supporting the creation of specific characters, our third sub-goal. The system is validated in two ways: first, by grounding the ideas in the field that traditionally studies expressive movement, the performing arts. Second, by demonstrating how these ideas can be implemented, how they alter animation sequences and how they can be combined for expressive effect.

The system is designed to be open and extensible. The main mechanism for extension is adding new properties, which may be desired for three reasons: to implement new ideas from the arts literature or experience; to customize control for a particular animation style; to create or adjust high level properties that are specialized for a specific character. Actions and character sketches are also reusable animation resources that can be augmented over time, leading to resource libraries. New sources for these resources based on motion capture and machine learning can also be explored.

Production work will require a larger range of motions. Better dynamic control algorithms remain a difficult problem and limit the range of tasks that can be simulated using forward dynamics.

Knowledge representation could be used to greatly expand the capabilities of the character sketch by allowing the sketch

to vary the edits it performs based on both the action that it is editing and the current mood of the character. In a similar vein, our framework could be used as a low-level subsystem for a high level AI character controller as it provides an appropriate set of handles for varying the style and content of a character's movements.

Finally, properties represent a step towards a *language* for movement, with the ultimate goal being the ability to provide concrete definitions for terms like "graceful". Such terms will in general not have a single definition as they encompass a range of movement nuances. Having concrete representations that cover a good portion of this range, within an extensible system, would be of immense value.

References

[ABC96] AMAYA K., BRUDERLIN A., CALVERT T.: Emotion from motion. *Graphics Interface '96* (May 1996), 222–229. 2

[Bar91a] BARBA E.: Dilated body. In *A Dictionary of Theatre Anthropology: The Secret Art of The Performer*, Barba E., Savarese N., (Eds.). Routledge, London, 1991. 3

[Bar91b] BARBA E.: Theatre anthropolgoy. In *A Dictionary of Theatre Anthropology: The Secret Art of The Performer*, Barba E., Savarese N., (Eds.). Routledge, London, 1991. 3

[BG95] BLUMBERG B. M., GALYEAN T. A.: Multi-level direction of autonomous creatures for real-time virtual environments. *Proceedings of SIGGRAPH 95* (August 1995), 47–54. 2

[BH00] BRAND M., HERTZMANN A.: Style machines. *Proceedings of SIGGRAPH 2000* (July 2000), 183–192. 3

[BW95] BRUDERLIN A., WILLIAMS L.: Motion signal processing. *Proceedings of SIGGRAPH 95* (August 1995), 97–104. 3

[CCZB00] CHI D. M., COSTA M., ZHAO L., BADLER N. I.: The emote model for effort and shape. *Proceedings of SIGGRAPH 2000* (July 2000), 173–182. 2

[FvdPT01] FALOUTSOS P., VAN DE PANNE M., TERZOPOULOS D.: Composable controllers for physics-based character animation. *Proceedings of SIGGRAPH 2001* (August 2001), 251–260. ISBN 1-58113-292-1. 3

[Gal92] GALLAHER P. E.: Individual differences in nonverbal behavior: Dimensions of style. *Journal of Personality and Social Psychology 63*, 1 (1992), 133–145. 3

[HRS94] HOLLARS M. G., ROSENTHAL D. E., SHERMAN M. A.: *SD/FAST User's Manual*. Symbolic Dynamics Inc., 1994. 7

[HWBO95] HODGINS J. K., WOOTEN W. L., BROGAN D. C., O'BRIEN J. F.: Animating human athletics. *Proceedings of SIGGRAPH 95* (August 1995), 71–78. 3, 7

[Lab88] LABAN R.: *The Mastery of Movement*, fourth ed. Northcote House, London, 1988. Revised by Lisa Ullman. 3, 7

[Las87] LASSETER J.: Principles of traditional animation applied to 3d computer animation. *Proceedings of SIGGRAPH 87 21*, 4 (July 1987), 35–44. 3

[Moo84] MOORE S.: *The Stanislavski System: The Professional Training of an Actor*. Penguin Books, 1984. 3

[Nef] NEFF M.:. http://www.dgp.toronto.edu/people/neff. 8

[Nef05] NEFF M.: *A Framework for Expressive Character Animation*. Ph.D. dissertation. Department of Computer Science, University of Toronto, 2005 (expected). 3, 5

[NF02] NEFF M., FIUME E.: Modeling tension and relaxation for computer animation. In *ACM SIGGRAPH Symposium on Computer Animation* (July 2002), pp. 81–88. 2, 7

[NF03] NEFF M., FIUME E.: Aesthetic edits for character animation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (July 2003), pp. 239–244. 2

[NF04] NEFF M., FIUME E.: Methods for exploring expressive stance. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aug. 2004), pp. 49–58. 2, 7, 8

[NF05] NEFF M., FIUME E.: Methods for exploring expressive stance. *Graphical Models* (2005). to appear. 7, 8

[NTHF] NG-THOW-HING V., FALOUTSOS P.: Dynamic animation and control environment. http://www.cs.ucla.edu/magix/projects/dance/index.html. 3

[PB02] PULLEN K., BREGLER C.: Motion capture assisted animation: Texturing and synthesis. *ACM Transactions on Graphics 21*, 3 (July 2002), 501–508. 3

[Per95] PERLIN K.: Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics 1*, 1 (March 1995), 5–15. 2

[PG96] PERLIN K., GOLDBERG A.: Improv: A system for scripting interactive actors in virtual worlds. *Proceedings of SIGGRAPH 96* (August 1996), 205–216. 2

[RCB98] ROSE C., COHEN M. F., BODENHEIMER B.: Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications 18*, 5 (September - October 1998), 32–40. 3

[Rey82] REYNOLDS C. W.: Computer animation with scripts and actors. vol. 16, pp. 289–296. 2

[RLO90] REEVES W. T., LEFFLER S. J., OSTBY E. F.: The menv modelling and animation environment. *Journal of Visualization and Computer Animation 1*, 1 (August 1990), 33–40. 2

[Sha63] SHAWN T.: *Every Little Movement: A Book about Francois Delsarte*, second revised ed. Dance Horizons, Inc., New York, 1963. 3

[Sta49] STANISLAVSKI C.: *Building a Character*. Theatre Arts Books, 1949. Translated by Elizabeth Reynolds Hapgood. 3

[UAT95] UNUMA M., ANJYO K., TAKEUCHI R.: Fourier principles for emotion-based human figure animation. *Proceedings of SIGGRAPH 95* (August 1995), 91–96. 2

[WK88] WITKIN A., KASS M.: Spacetime constraints. *Proceedings of SIGGRAPH 88 22*, 4 (August 1988), 159–168. 3

[Woo98] WOOTEN W. L.: *Simulation of Leaping, Tumbling, Landing, and Balancing Humans*. Ph.D. dissertation., Georgia Institute of Technology, 1998. 7