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

Spatial Keyframing for Performance-driven Animation

T. Igarashi [1,3], T. Moscovich [2], and J. F. Hughes [2]

[1] The University of Tokyo   [2] Brown University   [3] PRESTO, JST

Abstract

This paper introduces spatial keyframing, a technique for performance-driven character animation. In traditional temporal keyframing, key poses are defined at specific points in time: i.e., we define a map from a set of key times to the configuration space of the character and then extend this map to the entire timeline by interpolation. By contrast, in spatial keyframing key poses are defined at specific key positions in a 3D space where the character lives; the mapping from the 3D space to the configuration space is again defined by interpolation. The user controls a character by adjusting the position of a control cursor in the 3D space; the pose of the character is given as a blend of nearby key poses. The user thus can make expressive motion in real time and the resulting motion can be recorded and interpreted as an animation sequence. Although similar ideas are present in previous systems, our system is unique in that the designer can quickly design a new set of keyframes from scratch, and make an animation without motion capture data or special input devices. Our technique is especially useful for imaginary characters other than human figures because we do not rely on motion-capture data. We also introduce several applications of the basic idea and give examples showing the expressiveness of the approach.

1. Introduction

The most popular approach to character animation is keyframing, where the designer manually specifies the pose of a character as a discrete set of frames (keyframes) and the computer synthesizes the poses in the remaining frames. However, novice users have difficulty in creating fluid motion using this approach and it is very labor-intensive work. Other approaches such as motion capture and physically based simulation are available, but they are expensive to use and are not suitable for designing expressive imaginary motions. Furthermore, motion capture is mainly designed for human figures and is not directly applicable to imaginary characters.

We describe here a method that lets novice users create lively animations for arbitrary 3D characters quickly and easily using a standard input device such as a mouse. The basic idea is to directly record the user's performance or actions, that is, the user's direct manipulation of the character. We believe that this is much faster and more intuitive than traditional temporal keyframing, because the user need not mentally translate static keyframes to temporal motion during design. In performance-driven animation, what you perform and see on the screen during recording is what you obtain as the final result.

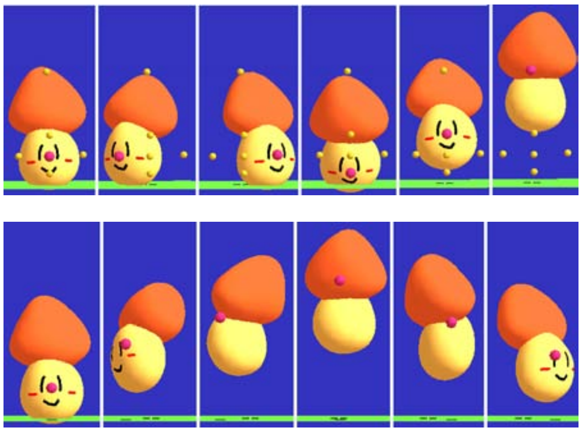

Figure 1: *Spatial keyframing with six key poses (top) and an example animation sequence using it (bottom). The user associates each pose with a location in space (yellow markers) in a preparation phase. During performance, the user moves the control cursor (red sphere) and the system synthesizes an animation sequence by blending nearby poses.*

The problem is that it is difficult to control complicated motion of a character in real time using standard input devices. A character may have many degrees of freedom (DOFs), including position, orientation, and angles at many joints, while a typical input device has only two DOFs. To control many DOFs in real time with input devices with limited DOFs, our system takes several predefined key poses and blends them in real time during performance. In a preparation phase, the user creates several key poses and associates them with specific positions in the 3D space (which we call spatial keyframes). During performance, the user moves a control cursor in the 3D space and the system synthesizes a corresponding pose by blending nearby spatial keyframes (Figure 1).

Synthesizing new poses by blending predefined poses is already common practice, and making animations from real-time performance is not new. The main contribution of this paper is to combine these methods in a practical system for making lively character animations from scratch without motion-capture data or special input devices. This paper also describes specific methods for blending poses to obtain reasonable results with very sparse key poses, as well as several extensions to the basic idea. We hope that this paper encourages the use of performance-driven approaches for character animation. The resulting animations are very different from those created using traditional methods, as they make apparent the animator's natural sense of timing. It is also much easier and more fun to create animations this way.

2. Related work

Performance-driven animation and puppeteering usually rely on a specialized input device or a motion-capture system [Stu98]. The direct mapping they assign between the character's DOFs and the device's DOFs requires a significant amount of training and expertise to control fluently. Shin et al. [SLG01] introduced methods for retargeting a motion-captured performer's full body motion to characters of different sizes. Dontcheva et al. [DYP03] proposed layered acting, where the user designs complicated motions by multiple acting passes. Our goal is to make performance-driven animation more accessible to casual users who lack expensive devices.

Synthesizing new poses by blending predefined poses is already done in many animation systems, but most of them are designed for large amounts of motion-capture data. Wiley and Hahn [WH97] associated key poses obtained by motion capture to a dense grid of points in space and linearly interpolated them. Rose et al. [RBC98] interpolated motion-capture sequences using radial basis functions to express the character's emotions. Kovar and Gleicher's system [KG04] automatically constructs a parameterized space of motions by analyzing large amounts of motion data and synthesizes a new pose by weighed interpolation of nearby poses. Our system is designed to work with very few key poses and does not require a large motion data set.

One work closely related to ours is the artist-directed inverse kinematics of Rose et al. [RSC01]. They also used radial basis functions to interpolate sparse examples in space. Our goal differs from theirs in that we focus on performance-driven animation authoring from scratch, while their focus was on controlling pre-authored motion data.

There are other related interactive systems for animation. Ngo et al. [NCD*00] used linear interpolation for manipulating 2D vector graphics. Key poses are embedded in a special structure called a simplicial configuration complex. Rademacher [Rad99] used interpolation for controlling view-dependent geometry. Key geometries are associated with specific view directions and are blended according to the current view direction. Laszlo et al. [LvPF00] combined interactive character control with physics-based simulations. They showed an example in which the horizontal and vertical motions of the mouse were directly mapped to the character's individual DOFs. The "motion doodle" system lets the user sketch the intended motion path; the system then synthesizes an appropriate motion by combining pre-authored keyframe animations [TBvP04]. Terra and Metoyer used performance for timing pre-authored key frame animation [TM04]. Donald and Henle proposed to use a haptic input device to manipulate motion capture data [DH00].

3. The User Interface

Our system consists of two subsystems. One is for designing spatial keyframes and the other is for making animation using these keyframes.

3.1 Designing spatial keyframes

The user's first task is to design a set of spatial keyframes, that is, to set poses of a character and associate them with positions in the 3D space. The user first imports a 3D articulated model into the system. We provide a standard direct-manipulation interface for the 3D model. The user can change the position of the character by dragging it within the space and change its pose by rotating its parts; the object can also be moved parallel to the ground by dragging its shadow [HZR*92].

Once the user is satisfied with a pose, the next task is to mark it as a new spatial keyframe by associating it with a position in the 3D space. To do so, the user moves the red control cursor to the target position and presses the "set" button. A small yellow ball now appears at the location of the control cursor that indicates the existence of the spatial keyframe. A spatial keyframe consists of two elements: (1) a character pose (e.g. joint angles) and (2) the xyz cursor position that corresponds to that pose. The user defines a set of these spatial keyframes by repeating the above process and these keyframes define a mapping from the control space to the set of the character's poses via an interpolation procedure described in the next section. The user can examine this mapping at any time by dragging the control cursor with the right mouse button down: the system blends the neighboring spatial keyframes around the control cursor and continuously displays the result. This synthesis is not done when the left mouse button is used; the left button is reserved for setting new poses.

The important feature of the system is that the user can set spatial keyframes at arbitrary positions in the 3D space, and that the user can start testing interesting motions with very few spatial keyframes. With only three keyframes, the user can make interesting full body motion, as shown in Figure 2. This is in contrast to linear interpolation systems that require many keyframes specified in a grid structure [WH97; NCD*00].

Note that the spatial keyframes are overlayed in the same 3D space inhabited by the character. This is in contrast to Ngo et al.'s system [NCD*00] in which keys are placed in a special configuration space. This makes it possible to establish an intuitive correspondence between the location of a spatial keyfmrame and the associated pose. For example, the keyframe for "look left" is likely to be on the left side of the screen and "look right" on the right (as in Figure 2). This intuitive mapping may be difficult to achieve when using automatic mapping such as the method introduced in [GMH04].

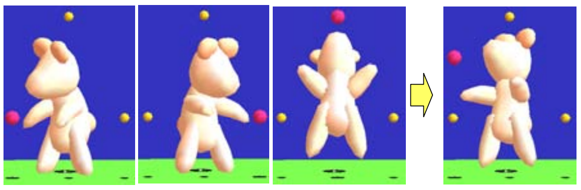

Figure 2: *A simple example with three key poses. Three key poses are associated with the three yellow balls (left). As the user drags the red control cursor with the right mouse button down, the system synthesizes a new pose by blending the three (right). Note that many joint angles as well as the character's position are controlled together.*

3.2 Making animation by performance

Having set the necessary spatial keyframes, the animator can use them to begin performing animations. In this phase, the character's pose can no longer be adjusted directly. The user moves the control cursor and the system shows the synthesized pose on the screen (Figure 3). Recording starts when the user starts dragging the control cursor after pressing the "record" button and finishes when the mouse button is released. The user can watch the resulting motion immediately by pressing the "play" button, and can watch it from any direction by changing the camera position.

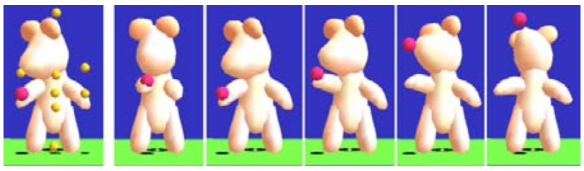

Figure 3: *Juggling. The user first sets the nine key poses as shown on the left. As the user drags the control cursor, the character performs a smooth motion as shown on the right.*

3.3 Discussion

In the current system a 2D mouse is used to control the position of the 3D control cursor and the control cursor moves parallel to the screen during dragging, so the motion of the 3D control cursor is actually 2D motion. Although interesting animations can be designed with this setup, we plan to investigate the possibility of using 3D input devices. Three-dimensionally distributed key poses may also be helpful for scripting purposes when specifying the 3D motion of the control cursor (see Section 5.3).

Designing animations in this way is really intuitive and fast. The spatial keyframe examples in this paper took only 10 to 20 minutes to design. This includes several iterations to adjust the resulting motion. After setting the keyframes, the only thing the user needs to do is to drag the control cursor. There is no need to directly edit each frame or repeat performance, as is required in layered acting [DYP03]. The time necessary to make an animation sequence is the same as the time to play it. In addition, the resulting motion is very lively because the user's direct hand motion is (indirectly) present in the animated motion. This is in contrast to the unnatural, robotic motion designed by novice users using standard keyframing. This idea is similar in spirit to the technique introduced by Terra and Metoyer [TM04], but they used performance to adjust only the timing of a predefined keyframe animation while our system allows the user to control timing and pose simultaneously.

A possible concern is that each mapping is usually specific to a single motion and thus not very reusable. This is true to some extent; the mapping defined for juggling makes little sense for other motions. However, our method allows the user to easily experiment with and design a wide variety of motions within a specific class of motion, e.g. in juggling the user can move the ball fast or slowly, high or low, clockwise or anti-clockwise. This flexibility is missing in traditional temporal keyframing methods and is critical for designing convincing motions.

4. Algorithm

This section describes the algorithm we use in the current implementation. Note that the main purpose of the following description is to provide the necessary information to implement the system, not to propose a better algorithm for motion blending. The blending of motion is essentially a difficult problem. Many techniques have been proposed, such as quaterenions and the exponential map, each with its own strengths and weaknesses. We chose this particular method because it is easy to implement, works well in practice, and satisfies certain reasonable user expectations It is our future work to implement and compare other approaches in detail.

The system takes the xyz-coordinates of the control cursor and a set of spatial keyframes (the xyz-coordinates of anchors and associated character poses) as input and returns a blended character pose. A pose is defined as a set of local transformations of the body parts. We currently do not allow translation for any part except the root, so we have 3×3 rotation matrices for all parts and one xyz translation vector for the root part. We interpolate each entry of the matrix using a radial basis function and orthonormalize the resulting matrix.

4.1 Interpolation using a radial basis function

Radial basis interpolation is useful for scattered data interpolation [Powell87]. We use the interpolation method described by Turk and O'Brien [TO02]. Each entry of each matrix is treated as a real-valued function on the control space, expressed in the form

$$f(\boldsymbol{x}) = \sum_{j=1}^{j=n} d_j \Phi(\boldsymbol{x} - c_j) + P(\boldsymbol{x})$$

where *F(x)* is a radial basis function, c_j are the marker positions, d_j are the weights, and *P(x)* is a degree-one polynomial. We currently use *F(x)* = | *x* | as a basis function. We chose this function empirically after several experiments, largely because the interpolation result follows the control cursor most faithfully. Other functions are smoother but show some oscillation effects.

The system solves for values d_j such that *f*(*x*) represents the given pose at the marker locations: supposing $h_j=f(c_j)$, the constraint is represented as

$$h_i = \sum_{j=1}^{j=n} d_j \Phi(c_i - c_j) + P(c_i)$$

Since this equation is linear with respect to the unknowns, d_j and the coefficients of *P*(*x*), it can be formulated as the following linear system:

$$\begin{bmatrix} \Phi_{11} & \cdots & \Phi_{1n} & 1 & c_1^x & c_1^y & c_1^z \\ \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \Phi_{n1} & \cdots & \Phi_{nn} & 1 & c_n^x & c_n^y & c_n^z \\ 1 & \cdots & 1 & 0 & 0 & 0 & 0 \\ c_1^x & \cdots & c_n^x & 0 & 0 & 0 & 0 \\ c_1^y & \cdots & c_n^y & 0 & 0 & 0 & 0 \\ c_1^z & \cdots & c_n^z & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ \vdots \\ d_{1n} \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ \vdots \\ h_n \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

where $c_i = (c_i^x, c_i^y, c_i^z)$, $\Phi_{ij} = \Phi(c_i - c_j)$,

$$P(\boldsymbol{x}) = p_0 + p_1 x + p_2 y + p_3 z \cdot$$

We obtain the interpolation function *f*(*x*) by solving the above linear system. We need to solve the linear system for each entry of the 3×3 rotation matrix. However, the large coefficient matrix on the left hand side of the above equation is identical for all nine entries, so we only need to invert the large matrix once for each joint. For the root part, we also compute each entry of the translation vector using this method.

Special care must be taken when there are fewer than four spatial keyframes and when the spatial distribution of the markers is degenerate (linearly or two-dimensionally distributed). In these cases, we apply the interpolation in a space of reduced dimensions by mapping the marker locations to the reduced space before applying the above procedure. The dimension of *P*(*x*) is also reduced accordingly. The choice of *F*(*x*) must also change to obtain true thin-plate interpolation [TO02], but we currently use *F*(*x*) = | *x* | for all dimensions and it works well.

4.2 Orthonormalization

The interpolated matrix obtained by the above procedure is not in general an orthonormal rotation matrix; we need to orthonormalize it. In some methods for orthonormalization such as Gram-Schmidt, the result is not necessarily close to the original matrix. We currently use the following iterative refinement method to orthonormalize the matrix by maintaining the balance between the three axes (Figure 4).

Suppose we have three basis vectors $\vec{x}_0, \vec{y}_0, \vec{z}_0$ and want to orthonormalize them. We first normalize them. We then compute $\vec{u}_0 = \vec{y}_0 \times \vec{z}_0$, $\vec{v}_0 = \vec{z}_0 \times \vec{x}_0$, $\vec{w}_0 = \vec{x}_0 \times \vec{y}_0$ and normalize these. Then we compute $\vec{x}_1 = (\vec{x}_0 + \vec{u}_0)/2$, $\vec{y}_1 = (\vec{y}_0 + \vec{v}_0)/2$, $\vec{z}_1 = (\vec{z}_0 + \vec{w}_0)/2$ and normalize them. We repeat the above procedure until the residual $r = (\vec{x}_n \cdot \vec{y}_n)^2 + (\vec{y}_n \cdot \vec{z}_n)^2 + (\vec{z}_n \cdot \vec{x}_n)^2$ is below a threshold or the number of repetition exceeds a predefined count. We currently use 0.000001 for the residual threshold and 10 for the count. It usually takes fewer than 3 iterations to obtain visually pleasing results. The maximum number, 10, is sufficient to detect a degenerate case.

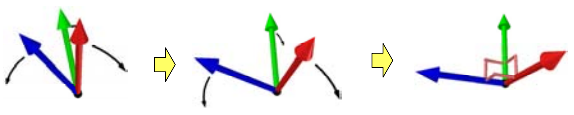
Figure 4: *Orthonormalization process. The system gradually makes the basis vectors perpendicular to each other.*

The above method is empirically designed with the goal of obtaining reasonable results robustly and quickly with a simple implementation. The outcome is satisfactory in our experience. It does not work for degenerate cases such as $\vec{x}_0 = \vec{y}_0 = \vec{z}_0 = \vec{0}$ that can arise, for example, when the control cursor is in the middle of two key poses that face completely opposite directions. In that case, the convergence fails and a skewed result is shown on the screen. Degenerate cases like this are unavoidable when creating a sufficiently nice mapping from 2-space or 3-space to the rotation group ("Sufficiently nice" in this case means that if two control points correspond to nearby rotations, then the line segment between them must correspond to a path near the geodesic path between these two rotations, which is meant to match user expectations). However, the answer is not well defined anyway in such cases from the user's point of view. The user naturally understands the existence of the degeneracy and avoids it during performance.

The method also has the nice mathematical property that if R is a rotation and M is a small perturbation to R that's orthogonal to the manifold of rotation matrices, considered as a submanifold of the manifold of all 3×3 matrices, then our process, applied to $R + tM$, yields a matrix that agrees with R to second order in t, i.e., it's esentially an orthogonal projection onto the rotation group, a property not shared by Gram-Schmidt, for instance, as can be seen by perturbing the identity by a small multiple of $\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$.

4.3 Why not angular parameterization?

One might ask why we do not use angular parametrizatoins such as euler angles, quaternions [Sho85], or the exponential map [Gra98] for the interpolation. Our short answer is that our domain is three-dimensional space and not sequential time, as is often the case when the typical interpolation happens. We describe two example issues that arise in this domain. Note that we only claim here that straightforward application of other approaches does not work well in our target domain. It may be possible to obtain similar results to ours by elaborating angular parameterizations [PSS02][BF01]. We leave further discussion to future publications.

The first issue is that simple angular parameterization does not behave as expected for extrapolation in our system (Figure 5). Suppose we have the two keyframes on the left. As we move the control cursor to the right, the head continues to turn if we use angular parameterization. In contrast, our method naturally keeps the head looking at the cursor. This is a design issue rather than a theoretical problem, but the basic idea behind spatial keyframing is to associate the pose with a position in space and angular parameterization breaks the natural mapping.

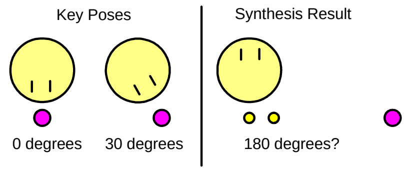

Figure 5: *An example synthesis result with straightforward angular parameterization. As the user drags the control cursor, the character rotates continuously. With our approach the character appropriately looks at the control cursor.*

The second issue is that there is a discontinuity when a part rotates 360 degrees. When using euler angles, the discontinuity is apparent. Even when using quaternions, the pose at 0 degrees and 360 degrees are located at opposite poles of the unit sphere in 4D space. This is not a problem when using a linear parameter space such as time, but we are working in two- or three-dimensional continuous parameter space. Figure 6 shows what happens when we apply angular parameterization naively. Suppose we have the four spatial keyframes shown on the left. If we move the control cursor between the first and last marker, the resulting pose is something between the second and third key pose on the left, because there is a discontinuity in angular parameter space between the first and last spatial keyframe. It might be possible to design methods that avoid this problem by elaborating on angular parameterization, but we believe that our approach (directly interpolating the rotation matrix) is more straightforward and easier to implement for our particular application domain.

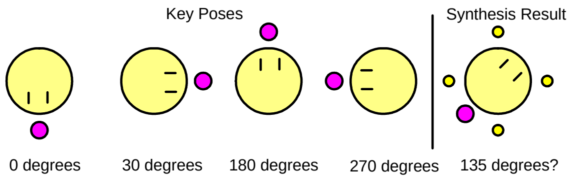

Figure 6: *Another synthesis result when using straightforward angular parameterization. If we have the four key poses shown on the left and places the control cursor at the place as shown on the right, the result is the blend of the four angles as shown on the right. Our system returns natural results by interpolating each component independently.*

Previous pose interpolation systems used angular parameterization [WH97; RSC01]; this was a reasonable decision because these systems were designed primarily to blend existing motions and the problems described above do not arise. However, our goal is the creation of a new motion from scratch by performance and it is crucial to be able to support dynamic behavior such as that shown in Figure 1.

5. Extensions

This section describes some extensions to the basic framework. These extensions are applications of existing ideas for spatial keyframing and we do not claim significant novelty here. We describe them here in order to show the possibilities of our technique and to inspire further exploration.

5.1 Inverse kinematics

Inverse kinematics (IK) is the process of determining the joint configuration required to place a particular part of an articulated character at a particular location in space. The most popular approach is to incrementally update the joint angles to satisfy the given constraints using Jacobian iteration [G M85; WW92]. In other words, the system gradually pulls the grabbed part to the target location. This means that the resulting pose is dependent on the previous pose, which can easily lead to very unnatural poses. Many solutions to this problem have been proposed, for instance using biomechanics knowledge [GGL96], constraint solving [YN03], and example-based optimization [GMH04]. However, they require manual encoding of various low-level constraints or large motion-capture datasets. Furthermore, it is difficult to include artistic control in the process.

Our spatial keyframing can be useful in adding artistic control to the inverse kinematics process. The algorithm is very simple. Instead of starting Jacobian-based refinement from the previous frame, we start the process from the synthesis result using spatial keyframing (Figure 7). This makes the resulting pose very stable. Regardless of the pose in the previous frame, the resulting pose is always the same for a given control cursor position. Our method can be seen as a subset of the one presented in Rose et al. [RSC01]. A similar technique is also used in [YKH04].

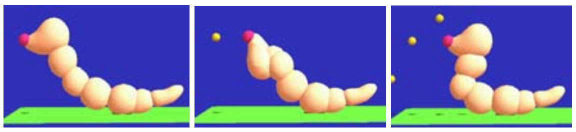

Figure 7: *Initial pose (left), standard IK result (middle) and IK with spatial keyframing (right). Standard IK can produce very strange poses after continuous operation. In contrast, IK with spatial keyframing returns stable results regardless of previous pose.*

5.2 Locomotion

Basic spatial keyframing is designed for controlling the character's pose at a fixed base position, and does not work well for animation involving travel or locomotion. The user can certainly represent a small positional change by setting the character in different places as independent spatial keyframes (as in Figure 1), but a long traveling sequence, such as walking, requires too many spatial keyframes.

One way to address this issue is to automatically change the character's position with respect to its body motion. We were inspired by interactive character control by Laszlo et al. [LvPF00], in which the user controls the character's limbs and the locomotion is generated as a result of a physically-based simulation. We would like to test similar physically-based simulation in the future, but currently use a simple rule to generate horizontal position change from the character's poses; at each point in time, the lowest point of the character is fixed relative to the ground, and the character's base position slides to satisfy the constraint [OTH02] (as it is too time-consuming to check all vertices, we manually mark the tip of each toe beforehand and use these marks for computing locomotion). When the lowest part is above the ground, the base position travels according to inertia; the system remembers the horizontal traveling speed just before the lowest point leaves the ground and continues to slide the ground with the same speed until another point touches the ground. The camera is fixed relative to the character's base position during recording.

It is possible that some point on the free leg dips lower than the supporting leg. In this case, the contact point suddenly switch, causing the character to start moving backwards. This problem can be serious if we consider all vertices of the mesh as possible contact points, but we can avoid most of the problem by using manually marked vertices only. In practice, we do experience some "waddling" motion when creating various walking motions, but the result is acceptable for novice users to quickly create simple animations. It is also very easy to interactively fix the problem by adjusting key poses and cursor trajectory when a problem occurs.

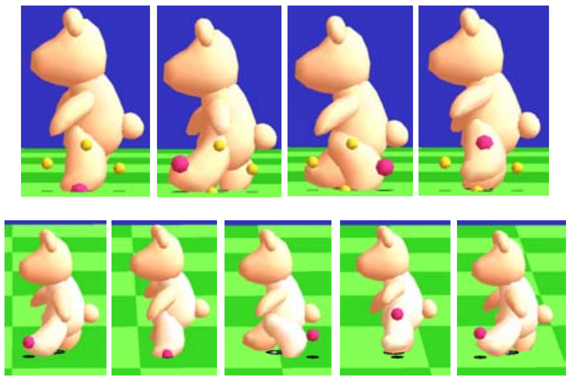

Figure 8: *Walking. Four key poses (top) and a walking animation using them (bottom). Observe that the ground slides along with the foot on the ground.*

Figure 8 shows an example. It has four key poses that represent a walking cycle. As the user moves the control cursor counterclockwise, the character makes a walking motion and the ground slides with the lower foot. The faster the user moves the cursor, the faster the ground slides, ensuring that foot-skating artifacts do not occur. Figure 9 shows another example. The three key poses represent a galloping motion. The system slides the ground to simulate inertia when the character is in the air.

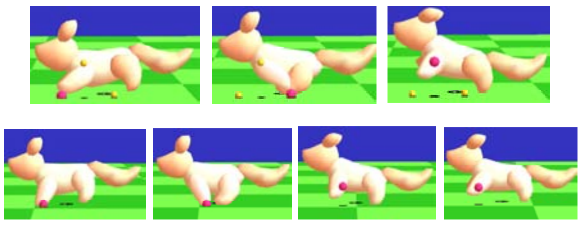

Figure 9: *Galloping. Three key poses (top) and a galloping animation using them (bottom). Observe that the ground slides along with the foot on the ground.*

5.3 Scripting

We developed spatial keyframing primarily for interactive puppetry and animation authoring by direct performance. However, the central idea behind it is to create a compact, low-DOF representation for high-DOF character poses, which should be useful for applications other than direct manipulation via a control cursor. One possibility is to use simple scripting for animation authoring. When using scripts to control a standard articulated character, individual joint angles must be specified explicitly. But using spatial keyframing, one can control rich character movement just by specifying the behavior of the control cursor in a script. Scripting with spatial keyframing is also useful for controlling mutually interacting characters.

We imagine that a set of spatial keyframes would be designed for each character, and that they would be packaged together (like the model and "rigging" of characters in animation studios). Then, the script authors would import the character with the spatial keyframe set and start writing scripts that select appropriate spatial keyframes and control the control cursor. In traditional scripting systems authors usually directly specify each joint angle [CDP00], so the spatial keyframe technique can significantly lower the bar and enrich the resulting animations. This is similar to a blend-shape interface where a character model is shipped with many adjustable control parameters, but our spatial keyframing is unique in that the control cursor lives in the same space as the character.

6. Implementation and Results

The current prototype system is implemented in Java (JDK1.4) and uses DirectX8 for 3D rendering. It currently uses articulated 3D models consisting of multiple rigid parts embedded in a hierarchical structure. Freeform surfaces guided by embedded bones are not currently supported, but it is straightforward to apply spatial keyframing to such bone structures. We use an extension to the Teddy system [IMT99] as a primary 3D modeling system, in which the user can design a painted articulated 3D character very rapidly (~10 minutes). Figure 1 and Figure 10 show example animations designed by the author. Acting with spatial keyframing is useful for expressing the characters' rich emotions in these simple actions.

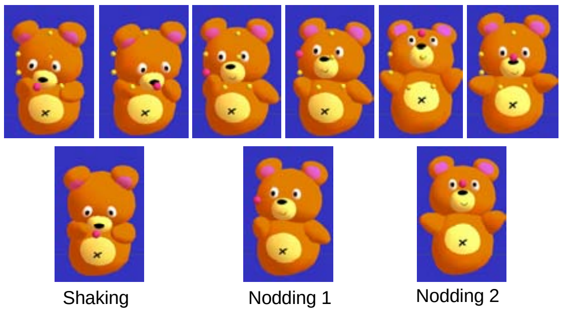

Figure 10: *An example animation. Using the six key poses (top), one can design an animation sequence in which the bear shakes his head, makes a small nod, and makes a large nod in turn seamlessly.*

Figure 11 shows an example of a highly articulated character. We experimented with various motions and found that our algorithm works well for these kinds of characters especially when the target animation is a general whole body motion such as dancing and gesturing. If the target animation requires precise placement of end-effectors, it might be better to interpolate the position of the end-effectors first and then apply inverse kinematics as in [YKH04]. It is our future work to implement this and compare the results.

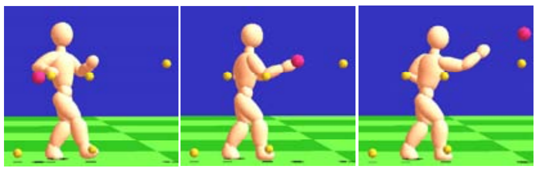

Figure 11: *An example of a highly articulated character.*

We have asked two professional artists to try the prototype system, one with extensive experience in 3D graphics and the other mainly in 2D animations. They both understood the concept quickly and started creating animation within 30 minutes. Examples are shown in Figure 12 and 13. They commented that the system was fun to use and the experience was very novel. However, the current implementation is still preliminary and the test revealed its limitations. The 2D artist had difficulty in setting individual poses with a mouse. Both wanted a mechanism to prepare multiple sets of spatial keyframes for different motions and switch from one motion to another to create meaningful stories. They noted that the system might not be immediately useful for professional production because they need precise control for each frame. They suggested that the system could be useful for real-time performance in front of audiences and for novice users.

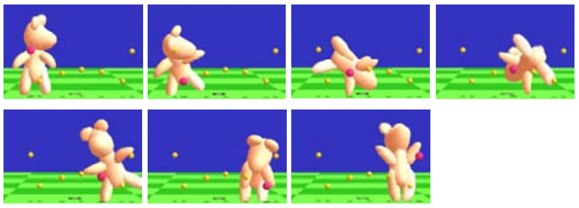

Figure 12: *An example spatial keyframes designed by a 3D expert with experience using standard keyframing. He found that spatial keyframing is much more fun to use, and that the resulting motion is very different from those created using existing methods.*

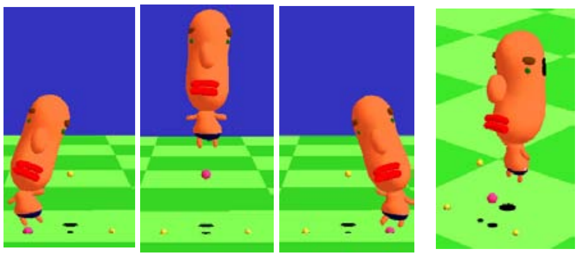

Figure 13: *An example animation created by a 2D artist. He found the system very fun to play with and inspiring but also found that it is still difficult to specify individual 3D poses with a mouse.*

7. Limitations and Future Work

The main limitation of our technique is that spatial keyframing is not directly applicable to some kinds of motions. It is very natural and effective for motions that are semantically associated with specific points in space, such as gazing and object manipulation, but is difficult to apply to more complicated motions such as speaking in sign language. Another problem is that spatial keyframing can represent only one type of motion at a time. We found that reasonably interesting animations can be designed with a single set of spatial keyframes by carefully distributing the key poses in space, but there certainly is a limit. To address these problems, we plan to investigate mechanisms for combining multiple spatial keyframe sets and achieving smooth transitions between them. How well a typical user can remember the different mappings is still an open question which we hope to answer in future research.

Spatial keyframing can easily be combined with existing methods for animation authoring. One can design more complicated motion by using spatial keyframing in layered acting [DYP03]. It is also straightforward to combine it with interactive physically based simulation to generate realistic motion automatically [LvPF00]. Motion doodles can be used to specify the trajectory of the character's locomotion while using spatial keyframing to control its pose [TBvP04].

Spatial keyframing can be seen as supplemental information added to a rigged character; skilled designers design a 3D character with predefined spatial keyframing and end users quickly author their own motion with it. We plan to develop tools to support widespread use of this framework. Examples include plug-ins for commercial 3D modeling and animation systems, 3D animation players that supports spatial keyframing, and a library of 3D articulated characters with pre-authored spatial keyframes.

References

[BF01] BUSS, S.R., FILLMORE, J.: Spherical Averages and Applications to Spherical Splines and Interpolation, ACM Transactions on Graphics, 20, 2, (2001), 95-126.

[CDP00] COOPER S., DANN W., PAUSCH R.: Alice: A 3-D Tool for Introductory Programming Concepts. *Journal of Computing Sciences in Colleges, 15,* 5 (2000), 107-116.

[DH00] DONALD, B. R., HENLE, F.: Using haptic vector fields for animation motion control. In Proceedings of IEEE International Conference on Robotics and Automation, (2000).

[DYP03] DONTCHEVA M., YNGVE G., POPOVIC' Z.: Layered Acting for Character Animation. ACM Transactions on Graphics, 22, 3 (2003), 409-416.

[GGL96] GULLAPALLI V., GELFAND J. J., LANE S. H.: Synergy-based Learning of Hybrid Position/Force Control for Redundant Manipulators. In *Proceedings of IEEE Robotics and Automation Conference,* (1996), 3526-3531.

[GM85] GIRARD M., MACIEJEWSKI A. A.: Computational Modeling for the Computer Animation of Legged Figures. In *Computer Graphics (Proceedings of ACM SIGGRAPH 85), 19,* 3 (1985), 263-270.

[GMH04] GROCHOW K., MARTIN S. L., HERTZMANN A. POPOVIC' Z.: Style-based Inverse Kinematics. *ACM Transactions on Graphics, 23,* 3 (2004), 522-531.

[Gra98] GRASSIA, F. S.: Practical parameterization of rotations using the exponential map, Journal of Graphics Tools archive, 3, 3, (1998), 29-48.

[HZR*92] HERNDON K. P., ZELEZNIK R. C., ROBBINS, D. C., CONNER, D. B. SNIBBE, S. S., VAN DAM A.: Interactive Shadows. In *Proceedings of UIST '92*, (1992), 1-6.

[IMT99] IGARASHI T., MATSUOKA S., TANAKA, H.: Teddy: A Sketching Interface for 3D Freeform Design. In *Proceedings of ACM SIGGRAPH 1999*, ACM Press / ACM SIGGRAPH, Los Angeles, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, (1999), 409-416.

[KG04] KOVAR L., GLEICHER M.: Automated Extraction and Parameterization of Motions in Large Data Sets. *ACM Transactions on Graphics*, *23*, 3 (2004), 559-568.

[LvPF00] LASZLO J., VAN DE PANNE, M., FIUME, E.: Interactive Control for Physically-based Animation. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 2000, 201-208.

[NCD*00] NGO T., CUTRELL D., DANA J., DONALD B., LOEB L., ZHU S.: Accessible Animation and Customizable Graphics via Simplicial Configuration Modeling. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 2000. 403-410.

[OTH02] OORE, S., TERZOPOULOS, D., HINTON, G.: A desktop input device and interface for interactive 3D character animation, *Proceedings of Graphics Interface 2002*, (2002), 133-140.

[Pow87] POWELL M. J. D.: Radial Basis Functions for Multivariable Interpolation: A Review. In *Algorithms for Approximation*, J. C. Mason and M. G. Cox, Eds. Oxford University Press, Oxford, UK, (1987), 143-167.

[PSS02] PARK, S. I., SHIN, H. J., SHIN, S. Y.: On-line Locomotion Generation Based on Motion Blending, In Proceedings of Symposium on Computer Animation, (2002), 105-111.

[Rad99] RADEMACHER P.: View-Dependent Geometry. In *Proceedings of ACM SIGGRAPH 1999*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, (1999), 439-446.

[RBC98] ROSE C., BODENHEIMER B., COHEN M.: Verbs and Adverbs: Multidimensional Motion Interpolation Using Radial Basis Functions. *IEEE Computer Graphics and Applications 18*, 5 (1998), 32-40.

[RSC01] ROSE III C. F., SLOAN P.-P. J., COHEN M. F.: Artist-Directed Inverse Kinematics Using Radial Basis Function, Interpolation. *Computer Graphics Forum*, *20*, 3 (2001), 239-250.

[SLGS01] SHIN H. J., LEE J., GLEICHER M., SHIN, S. Y.: Computer Puppetry: An Importance-Based Approach. *ACM Transactions on Graphics*, *20*, 2 (2001), 67-94.

[Sho85] SHOEMAKE, K.: Animating Rotations with Quaternion Curves. In *Computer Graphics (Proceedings of ACM SIGGRAPH 85)*, *19*, 3 (1985), 245-254.

[Stu98] STURMAN, D. J.: Computer Puppetry. *IEEE Computer Graphics and Applications*, *18*, 1 (1998), 38-45.

[TBvP04] THORNE M., BURKE, D., VAN DE PANNE M.: Motion Doodles: An Interface for Sketching Character Motion. *ACM Transactions on Graphics*, *21*, 3 (2004), 424-431.

[TM04] TERRA S.C.L., METOYER R.A.: Performance timing for keyframe animation. In *Proceedings of SCA 2004*, (2004), 253 - 258.

[TO02] TURK G., O'BRIEN J. F.: Modelling with Implicit Surfaces That Interpolate. *ACM Transactions on Graphics*, *21*, 4 (2002), 855-873.

[WH97] WILEY D.J., HAHN J.K.: Interpolation Synthesis of Articulated Figure Motion. *IEEE Computer Graphic and Applications*, *17*, 6 (1997), 39-45.

[WW92] WATT A., WATT M.: *Advanced Animation and Rendering Techniques: Theory and Practice*. Addison-Wesley, 1992.

[YKH04] YAMANE, L., KUFFNER, J. J., HODGINS, J. K.: Synthesizing animations of human manipulation tasks. *ACM Transactions on Graphics*, *23*, 3 (2004), 532-539.

[YN03] YAMANE K., NAKAMURA Y.: Natural Motion Animation Through Constraining and Deconstraining at Will. *IEEE Transaction on Visualization and Computer Graphics*, *9*, 3 (2003), 352-360.