A performance-based technique for timing keyframe animations

Sílvia César Lizana Terra, Ronald Anthony Metoyer *

Oregon State University, School of EECS, Kelley Engineering Center, Corvallis, OR 97331, USA

Received 23 January 2006; received in revised form 12 July 2006; accepted 22 September 2006
Available online 21 November 2006

Abstract

From our experience observing novice animators, it is clear that setting keyframe spatial values is straightforward while specifying the keyframe timing is difficult and often time consuming. This paper presents a modified approach to the keyframing paradigm, performance timing, that lets the user focus on the timing of keyframes separately from the spatial values. In performance timing, the user “acts-out” the timing information using a simple 2D input device such as a mouse or pen-tablet. The user’s input is analyzed and features of the user’s input are mapped to the spatial features of the keyframed motion. The keyframes are then distributed in time according to the timing of the user’s input path. We demonstrate the approach on several scenes and discuss the situations in which it is most and least effective. We present the results of a user study of over 20 subjects in which we compare accuracy using performance timing to accuracy using a standard animation package for specifying keyframe timing.

© 2006 Elsevier Inc. All rights reserved.

Keywords: Performance animation; Keyframing; Sketching

1. Introduction

Animation has its roots in the beginning of the twentieth century. As a new field, animation has gone through a number of changes, from the invention of cels—which allowed background and foreground to be drawn separately—to much more recently, the use of computers for generating images. However, the basic animation process remains essentially unmodified. The lead animator specifies the primary or key poses for the character, and someone else (or something else, in the case of computer animation) draws the motion in between those poses. The keyframes determine the important poses of the animation, while the in-between frames take the viewer from key pose to key pose. The number of frames between each keyframe determines the timing of the animation. A small number of frames results in rapid motion while a large number of frames results in slow motion. By controlling the spacing of keyframes, one controls the timing of the animation. Unfortunately, correctly placing keyframes in time can be very tedious and time consuming, especially for the novice animator.

As 3D computer animation tools become more accessible to novice users, interaction techniques should address the needs of these users. We chose to explore keyframing because it is one of the most fundamental motion generation techniques. Within the context of keyframing, the talented animators...
of Walt Disney developed the 12 principles of animation for creating compelling motion [23]. While these principles were created for hand-drawn animation, they have been applied to 3D animation [15,14,5]. Several of these principles deal directly with timing for animation. According to John Lasseter: “... it gives meaning to the movement—the speed of an action defines how well the idea behind the action will read to an audience. It reflects the weight and size of an object, and can even carry emotional meaning.”

And to Isaac Kerlow: “The skillful timing of the action in an animated project, or in any visual storytelling for that matter, can have a major positive effect on the audience...”

From our observations of novice students in computer animation classes (in a computer science department), we found that timing an animation is often the most difficult part of the process for novice users. Novice users do not have much difficulty in setting the spatial values of the keyframes using an animation suite such as AliasWavefront’s Maya. They are often quite capable of positioning a character in the desired pose. It is the spacing in between the poses, the timing, that is most difficult to specify. Because they are not trained animators, students struggle to place the key poses in time such that they evoke a particular performance with specific timing characteristics. This is clearly not an issue for professional animators with years of experience. We do not believe that the novice’s inability to create the desired timing is caused by an inability to imagine the timing, but by an inability to convey the timing using the provided interfaces. On the other hand, we found that many people find it fairly intuitive to “perform” or mimic the desired motion with their hands. For example, most of us could probably mimic a bouncing ball motion by tracing the path in mid air with a finger. In fact, this exact behavior was demonstrated (without any urging on our part) by the great majority of the participants in our user study. These ideas and observations are the basis of a method that we call performance timing which we originally presented in [22].

We first capture the user’s timing by requiring her to act out the motion of a single animated object. We then determine the correspondence between the acted motion and the object’s spatial motion path. All of the object’s keyframes are then redistributed in time according to the acted motion, modifying the timing of the animation without changing the motion path itself.

In this paper, we extend this work on two substantial fronts. First, we present alternative methods for computing correspondence between the object motion and the acted-out timing path and discuss the pros and cons of each method (Section 3.3). Second, we perform an extensive user study to compare novice user performance in specifying timing using performance timing and using a traditional animation package interface (Section 4). Our goal was to identify if subjects could truly imitate timing better than they could specify timing with a traditional interface.

2. Related work

Each year, animation moves more within the reach of the novice user—the hobbyist or the student, who may need content for presentations, web sites, or simply entertainment. Most current animation tools were not created with that demographic in mind. Several research groups have explored interfaces for beginners and many of these efforts used sketching as their primary form of input. The most inspirational work for our approach was proposed in 1969 by Baecker and it was called the GENESYS Computer Animation System [3]. The GENESYS system aided animators by automating the process of creating in-between frames and by providing for some rough animation tools. In the GENESYS system, the user sketched the motion path using the desired timing and shape. The motion was then replayed along that path with the user’s exact timing. In effect, this was a form of 2D motion capture.

In 1995, Balaguer and Gobbetti introduced an integrated environment for creating animations using 3D sketching [4,12]. The system was designed to allow for the expressiveness of a real-time motion capture system in a general animation system. The environment allows users to sketch animations using a 3D device, controlling both the spatial and timing aspects. A data reduction step fits a curve to the user data, resulting in a clean animation. Our approach is different in several ways. We intentionally separate spatial specification from timing in order to make use of existing techniques for specifying 3D motion paths through keyframe values. We then provide a method for timing the animation within a standard 2D mouse-based interface. Blum et al. also chose to separate timing and posing in
their ImageTimer interface, however, this interface was designed to help professional animators quickly adjust timing using a traditional paradigm [6].

Dontcheva and her colleagues present the layered acting approach for creating character animations [10]. In this system, the user acts the motion out, in 3D, using a specially designed puppet—a toy augmented with motion capture gear. The user’s actions are captured and mapped to the digital character that the puppet represents. They allow for fine-tuning of the acted motion by permitting the user to re-animate the character several times. The resulting animation is a weighted average of the user’s multiple demonstrations. The motion can also be restricted to certain portions of the character, so that, for example, only the legs are captured during one sequence, while the head is captured in another. The presented system requires a rigged puppet and motion capture equipment for collecting user input. This system also requires that the user demonstrate both spatial and timing aspects together. Our approach is very similar in that we allow users to directly control features of the “character” by capturing motion. However, we have focused on using easily available 2D devices and we consider a technique that modifies timing while leaving the spatial aspects of the motion unchanged.

In 2000, Laszlo et al. presented an entertaining interface for controlling interactive physical simulations by directly mapping mouse motion to the desired joint angles and by providing discrete control actions via keystrokes [16]. The goal was to build on the user’s intuition about how the motion should be performed and to use this intuition to drive the character directly. The user could set the character in motion and ask it to perform actions at certain points in time by setting desired poses for the physical simulation.

Most recently, Thorne and his colleagues presented a sketch-based technique for creating character motion [24]. In this system, the user not only sketches the character to be animated, but also sketches motion curves that are segmented and mapped to a parameterized set of output motion primitives that reflect the timing of the input sketch. The user does not sketch the actual motion path, but rather sketches a symbolic representation of the actions the character will perform. The system then translates the user’s input into a library of pre-generated motion. The generated motion is restricted by their ‘alphabet’ of motions and by their motion library.

Popović et al. discuss an interface for sketching the desired path and timing for rigid body simulations [17]. An optimization step then produces physically plausible motion that reflects the desired path and timing from the sketch.

An approach that is similar to ours in nature but not interface is Sampath’s NUKE plugin for Maya [18]. This plugin allows users to retime keyframed animations by scrubbing the animation time slider to reflect the desired timing. Scrubbing the timeline can cause un-recoverable changes on the spatial portion of the motion, such as two or more key-frames being merged into one. Scrubbing is also not a very intuitive interface for specifying timing.

Since our original performance timing paper [22], Igarashi et al. have presented a novel approach for performance animation called spatial keyframing [13]. In spatial keyframing, key frames are associated with 3D points in space as opposed to points in time. Each point in space is also associated with a specific pose. The user can control the character by moving a 3D cursor in the space of poses. The pose that corresponds to a particular 3D location is computed via interpolation. The timing of the motion comes from the timing of the 3D cursor during the user’s performance.

Real-time motion capture is the ultimate goal in performance animation. Unfortunately, motion capture systems are expensive and often cumbersome, especially for animation of simple characters. A thorough overview of motion capture for animation is presented by Gleicher [11]. Performance timing has different goals from motion capture. With motion capture the user typically wants to record both timing and spatial information. In performance timing we only want to modify the timing.

3. Performance timing procedure

The performance timing approach consists of four main steps. First, the user must define the keyframe spatial values. Next, the user “acts out” the motion for the keyframes demonstrating the desired timing. A correspondence is then computed between the acted-out motion path and the spatial path of the keyframe values. Finally, the timing of the acted-out motion path is then used to redistribute the keyframe values in time to reflect the timing of user’s performance. We describe each step in detail in the following sections.
3.1. Setting the keyframe values

For performance timing, the user must first set the keyframe values using a standard keyframing interface. When setting these keyframes, the initial timing may be arbitrary. This will result in an animation that goes through the correct spatial values but that does so with arbitrary timing. The process of setting keyframes in traditional animation is virtually identical across all platforms and typically follows this format:

1. Move to the correct point in time.
2. Place object in desired pose.
3. Create a keyframe.
4. Play back animation.
   • adjust keyframe if animation does not look right
5. Repeat
   (Workflow A).

By making the initial timing arbitrary, the workflow for performance timing now looks as follows:

1. Move to an arbitrary point in time after previous keyframe.
2. Place object in desired pose.
3. Create a keyframe.
4. Repeat
   (Workflow B).

Performance timing aims to simplify a process that as described in Workflow A can be extremely tedious. Rather than require that the user adjust the timing via trial and error, our performance timing algorithms will adjust the keyframe time after the keys have all been set and the user has performed the motion.

3.2. Acting out the timing

Once all keys have been set, the user is now ready to adjust the keyframes in time by acting out the timing. The first step in our approach requires that the user select an object in the scene to imitate. The target object must contain a spatial component for the user to mimic. This spatial component must exhibit values that vary in at least one of the translation channels (x, y, or z). This is the component that the user will mimic while performing the motion. If there is no variation in this component, then the resulting motion consists of a point in space. A point has no spatial variation with time and therefore makes a poor target for acting out the motion.

The target object might not have an explicit change in the translation channel but may still exhibit spatial motion. For example, consider a rotating object with a center of rotation not located at the object’s center. The translational channel for this object would not show any change. However, because the off-center rotation causes a translational motion, this object would make a fine target for the user. As another example, consider a child object in a hierarchy of linkages. As the parent object rotates, so does the child object, creating a spatial change. If an object presents only scaling or rotation changes, it could still produce a motion path by observing a sub-component or child of the object (Fig. 1).

For most scenes, it is fairly easy to find the correct object to use as the target object. For example, imagine a scene with a bouncing ball. In this scene, the user would select the ball as the target object to imitate. The ball is a good selection because it exhibits clear spatial translation. In a more complex scene, such as a maestro directing an orchestra, the animator could decide that the most expressive element is the tip of the wand or one of the conductor’s hands. The user selects an element that she believes appropriately represents the motion and that she can imitate on a 2D surface. Furthermore, the user is free to select the target object in any orthographic or perspective camera view.

Once the target object is selected, our system visually presents the translation motion path of the target object as a 3D path projected onto the 2D viewplane. The motion path is a representation of the path the object takes through space. It is generated by sampling the object at regular intervals of the animation and is composed of piecewise linear segments (Fig. 2).

Now that the target is selected and the motion path has been presented to the user, the user must choose a view and act out the desired timing for that view. The user can choose any camera view with either a perspective or orthographic projection. The goal is to choose the view in which the target object’s 2D projection is most expressive.

The user then acts out the scene by sketching a path with the desired timing directly on the window containing the motion path of the target object. The sketched path should ideally be the same shape as

1 The sketched path is also referred to as the timing path.
As the user sketches, the system samples the input sketch, resulting in time-stamped path samples. These samples are defined as $p_i(x, y, t, d)$, where $x$ and $y$ are the screen coordinates, $t$ is the time in milliseconds, $i$ is the sample index, and $d$ is the arc distance traveled so far. This is a simple form of two-dimensional motion capture. Fig. 2 shows an example target object along with the sketched timing path as seen in the actual interface. Similar to the sketched path, the object’s motion path is made up of samples defined as $r_j(x, y, d)$.

Unless specifically stated, all references to coordinates and locations in space refer to screen space.

### 3.3. Computing correspondences

Now that we have a motion path and a sketched timing path, we use the timing of the sketched path to redistribute the target object’s keyframes in time. The first step in this process is to compute correspondences between the motion path and the sketched timing path (Fig. 3). A naïve approach...
would be to use the timed samples of the sketch curve to directly represent the motion path for the target object. This approach, however, would modify the spatial properties of the object if the sketched path did not match the motion path exactly. Another disadvantage of this approach is that the resulting motion would be restricted to the 2D plane in which the user sketched the path unless a 3D pointer device such as Flock of Birds or Phantom [2,20] were used. Instead, we compute and match features of the sketched path to features of the target object’s motion path.

To find correspondences, we must first find the features of each path. Feature matching is a common problem in computer vision and therefore many algorithms exist for finding and subsequently matching feature points in images. However, treating our paths as images gives rise to certain complications. First, our paths contain self intersections when projected to the 2D plane, and these should not be counted as features. Second, our paths have a temporal component not present in images. The difference between a feature in an image and in our paths makes some computer vision approaches, such as the Harris corner detector for feature point finding, inappropriate for our needs.

Ideally, the correspondence method will avoid these problems in feature detection and the resulting correspondence will be intuitive to the user. In other words, the matching should be similar to what the user might have created by hand given a manual matching tool. We have experimented with several feature identification and correspondence techniques described in the following sections.

3.3.1. Min–max peak features

We have developed a heuristic method for finding features and correspondences in the interest of maintaining a fast, interactive interface for the user. Our heuristic feature finding approach involves finding the local minimum and maximum points of a path in both the horizontal and vertical directions in screen space. It requires that we find changes in direction along the $x$ and $y$ axes that are larger than a threshold value $u$. This is done by taking the $x$ and $y$ derivatives of the path with respect to time. The zero-crossings of the derivative are marked as temporary features. A feature is eliminated if there is another feature within a distance of $u$. Since we must find the features in both the motion path and the sketched timing path, this technique is $O(n + m)$ where $n$ is the number of samples in the target component motion path and $m$ is the number of samples in the sketched timing path. Once identified, the features are forced into a 1-to-1 correspondence sequentially.

Fig. 3. An example of correspondence matching for the two paths: the target object motion path and the sketched timing path.
As Fig. 4 shows, this method is not rotation independent and if the threshold is not adjusted accordingly, it is not scale independent either.\footnote{If the view is modified so that the path is at a different scale, \( u \) should be scaled by the same factor.} We therefore assume that users will maintain a sketched path orientation that is similar to the original motion path. In practice, we found this to be true for the majority of users in our study (discussed in Section 4). We acknowledge that this may not always be the case, and can be troublesome for those that always sketch at a rotational offset. In practice, this heuristic technique is efficient, produced meaningful features, and easy to implement. However, it relies on the assumption that the input will not be rotated and that features are defined by horizontal and vertical peaks.

### 3.3.2. Curvature analysis

Another feasible option is marking high-curvature areas as features [8]. This approach is scale-dependent, but rotation independent, and is still \( O(n + m) \). For this approach, we define two thresholds, \( u_0 \) and \( u_d \), \( u_0 \) is the angle threshold at which we consider a point in the path to have enough curvature to be considered a feature. \( u_d \) is the arc distance from the current point, in pixels, used as the window for computing curvature. Since \( u_d \) is in pixels, changing the scale of the path changes the significance of this value.

As Fig. 5 shows, the distance at which a path is observed has implications on curvature analysis. The distance itself is not important, but the observed size of the path on the viewplane is important.

For every sample \( i \), we find two other samples, \( i - a \) and \( i + b \), where the distance between \( i \) and each of the two samples is \( u_d \). \( u_d \) is measured in this case not as a straight line, but as the arc length distance on the path. Using these points and the current sample point, we can compute the bend angle at that sample. If it is less than \( u_h \), then the curvature is high enough to mark as a feature. As before, the identified features are forced into a 1-to-1 correspondence in sequential order.

This approach may produce clustered features in high-curvature areas. We select the local minimum of the cluster as the actual feature. One disadvantage of this approach is that if the motion maintains the same level of curvature for long spans, then there will be a very limited number of feature points that are found. Paths resembling a circle or a spiral, for example, could appear featureless.

### 3.3.3. Contour matching

A contour matching approach is a more robust approach to finding correspondences than either of the previously reported approaches [21], but it assumes features or a sampling have already been identified. It is a discrete, dynamic programming solution for curve matching. Unfortunately, it is more computationally expensive and thus less interactive than the previously described algorithms.

For this approach, we first align the two paths in space so that the origin and endpoints match. The timing path is rotated and scaled as necessary to align it with the spatial path. Because the paths are aligned in this way, the cost at the origin and endpoints is zero and they are always matched appropriately (Fig. 6).

Assuming that sample \( m_i \) from the motion path has already been matched to sample \( t_j \) from the timing path, the algorithm will then match \( m_{i+1} \) to \( t_k \) for \( k > j \). The error associated with this match is \( D = B - A \) where \( A = m_i - t_j \) and \( B = m_{i+1} - t_k \) (Fig. 7). The cost is then defined as \( ||D||^2 \). This computation is performed for every possible \( t_k \), storing the costs in a matrix. Once the entire matrix is filled with the costs for each match, a minimum cost search is performed on the matrix. The search results in the optimal correspondence.

This approach favors assigning correspondence to areas with a high degree of similarity. It does not
work well, however, if the shapes of the paths are very dissimilar. Fortunately, in this particular application, the shapes of the paths are usually very similar. This approach is also more computationally expensive than the previously mentioned approaches, having a running time of $O(nm)$. Since it does not produce single features, but considers every sample to be a correspondence point, this approach does not mesh well with a manual editing step such as that described in the following section. This can be overcome, however, by subsampling the set of correspondence points.

3.3.4. Manual feature editing

Regardless of which feature matching algorithm is used, the resulting features may not exactly match the user’s expectations. We therefore provide a manual editing technique for creating and removing feature points. At one extreme, we could start with no feature points and add all of them manually. For most animation sequences, this process could quickly become as tedious as adjusting individual keyframes. Manually setting the correspondence between the motion and timing paths should be seen as a final step in the performance timing process. When the algorithm fails or when the user has a unique vision of where the features should lie, the editing tools are available.

The user clicks on a point reasonably close to the path to place a feature at the corresponding location. Features are eliminated via selection and deletion. To change the position of a feature, the user must eliminate that feature and add it at the desired location.

3.3.5. Comparison of correspondence techniques

In summary, we chose to use the min–max Peak feature approach for several reasons. First, it is the most computationally efficient method. Our goal...
was to design a system for fast prototyping of animated motion and therefore it is important that the user receive fast feedback when performing a timing. Second, the curvature-based approach suffers from the problem of scale-dependence, and more importantly, it suffers from the problem of not being able to distinguish features when the provided motion curve does not contain many curvature changes (such as a spiral). Although the subpixel contour matching approach is a robust correspondence solution it is computationally less efficient than the other two options. Furthermore, this method does not find features, rather it aligns sample points from the two signals. This means that sample points must be provided. In our tests, we sampled the signal at a fairly high rate in order to ensure that important possible features were captured. This approach will generally result in a larger number of features than the previous methods, even when using a small sampling rate. A large number of features does not lend itself well to the manual editing phase. One could subsample the resulting features, keeping every $n$th feature, however, these features would not necessarily correspond to any intuitive notion of a feature for the user. A possible extension would be to compute and use both curvature and min–max peaks as features and then align these features using contour matching, thus avoiding the forced 1-to-1 forced correspondence of those techniques but benefiting from the alignment abilities of the contour matching approach.

Overall, the min–max peak approach performs well, is computationally efficient, and lends itself well to manual editing when the identified features are not correct in terms of the user’s intuitions. We found that the features produced are generally those that are ‘expected’ by the user.

### 3.4. Applying the sketched timing path

After establishing the correspondence between the target object motion path and the sketched timing path, we adjust the target object’s keyframes in time to reflect the user’s timing while preserving the original spatial properties. The paths are broken into $cp - 1$ segments, where $cp$ is the number of correspondence points. Each segment is then parameterized according to arc length. We cycle through all of the keyframes that will be modified and assign them a value $(cp, s)$ such that the keyframe is in segment $cp$ at arc length $s$. This tuple is used to index into the sketched timing path to find the time stamp value that is present at the same cumulative arc length on the timing path (Fig. 8). If there is no sample at this particular arc length, one is interpolated from the two closest neighbors. The sample’s timing is then applied to the keyframe at that particular arc length.
arclength distance in the target object’s motion path.

3.4.1. Adjusting tangency

After redistributing the keys in time, some adjustments may be necessary to maintain the path’s original shape (Fig. 9) to avoid modifications to the spatial component of the motion. In most animation packages the underlying motion curve of any component is represented with a spline where the control points represent each keyframe [1,9,7]. Each control point has two adjustable handles or tangents, allowing for timing adjustments without the addition of extra keyframes. In some animation systems, such as Maya, spline tangents can be ‘fixed’. Unlike a regular tangent, a fixed tangent is not modified automatically if the keyframe is moved in time from its current position, therefore, fixed tangents can cause undesirable side effects when keyframes are moved as demonstrated in Fig. 9. If fixed tangents are used by the animator, the system must modify the handles to guarantee that the spatial properties of the motion remain unmodified.

A keyframe handle must be modified so that it only changes the time the target object takes from one keyframe to the next keeping the motion path unmodified. To do so, we scale the tangent along the time axis (Fig. 10) keeping the value axis unchanged. In Fig. 11, \( k_i \) and \( k_{i+1} \) represent the current keyframe being processed and the next keyframe. Let the original distance (in time) between \( k_i \) and \( k_{i+1} \) be \( l_{\text{old}} \). Assume that the handle to be adjusted forms an angle of \( \theta_{\text{old}} \) with the horizontal axis. We complete the right-angle triangle by computing the height, \( h = l_{\text{old}} \times \tan(\theta_{\text{old}}) \). We then calculate the new timing for \( k_i \) and \( k_{i+1} \) with the process described in Section 3.4. Let the new distance between the keys be \( l_{\text{new}} \). Using \( l_{\text{new}} \) and \( h \) from the original timing, we have the trigonometric relationship: \( \tan(\theta_{\text{new}}) = h/l_{\text{new}} \) and we can now compute the new tangent angle as

![Fig. 9. The top path represents the original animation path for the target object. The horizontal axis is time while the vertical axis is the value of the animation channel (i.e. translation X). In this example, imagine that the user has specified fixed tangents for the spline that represents the animation curve. Path (a) shows the results of moving the keyframes in time without adjusting the tangents—the motion is clearly changed. Path (b) shows the results after we modify the angle of the tangents.](image)

![Fig. 10. When the tangent is scaled only in the time axis, the spatial characteristics of the curve remain the same.](image)
This process is performed for both handles of the keyframe. Keyframe handles can originally be set to be modified independently (broken) or together (joined). If they are joined, we break them temporarily and rejoin them once both handles have been set.

The result of the process is a keyframed animation with a different distribution of keyframes (in time) than the original animation and with no change to the spatial aspects of the motion. This resulting animation can be further modified using performance timing or using traditional keyframe animation methods. The fundamental form of the data is unchanged and therefore all traditional tools and techniques still apply.

4. User study

One of the major contributions of this paper beyond our original work [22] is the user study. We hypothesize that subjects can imitate timing, using our interface, more accurately than they could specify timing with a traditional interface. We therefore performed a user study to determine the effectiveness of performance timing as compared to traditional keyframe timing tools. We intended to study novice users with little or no familiarity with animation. Our subjects’ performance was recorded and their final results saved and analyzed. Our analysis was supplemented with a series of questionnaires. This section covers the study protocol and analysis methods in detail.

4.1. Demographics

The user study was conducted over a period of two weeks and included 21 students. Out of these students, two were female, while 19 were male. Table 1 shows a breakdown of the students by major.

![Diagram](image)

Fig. 11. From the original (top) path, we compute the \( h \) value, and in the retimed (bottom) path, we use the computed \( h \) to compute \( \theta_{new} \). For the sake of simplicity, this image shows only one handle from keyframe \( k_i \). In reality, both \( k_i \) and \( k_{i+1} \) have two handles each. One handle pointing toward each of their neighbors.

\[
\theta_{new} = \arctan \left( \frac{l_{old} \times \tan(\theta_{old})}{l_{new}} \right)
\]

4.2. Protocol

The study was performed with one subject at a time. Each session lasted less than 2 h. At the start of a session the subject was asked to fill out a questionnaire detailing their experience with Maya and animation in general. Next, the subject was given a half-hour tutorial on the basics of Maya. The tutorial explained the interface and covered some basic keyframe animation concepts. We then presented a simple scene and demonstrated how to retime the scene using the traditional method in Maya and then using our performance timing method.

There are two traditional methods for modifying keyframes in Maya: the Dope Sheet and the Graph Editor. We chose the Dope Sheet editor as the traditional method for adjusting the keyframe times because modifications made through the Dope Sheet do not result in any spatial changes to the animation. This is not true of the Graph Editor. Both the Dope Sheet and the Graph Editor are typical components of most animation packages and are used for editing keyframed animation.

The subjects were then presented with four different scenes in random order. Each of the scenes
included keyframed object motion with an arbitrary initial timing. For the sake of this study, we are assuming that the subject (or someone else) has successfully set the spatial values of the keyframes and is now concerned with adjusting timing only. Each of these scenes also had a corresponding video clip that showed the desired timing for the animated objects in the scene. Our goal was to understand how well the subject was able to reproduce this desired timing using the two provided tools. The subject was then asked to retiming the scene, in Maya, to match the timing demonstrated in the movie, first using one method, and then the other. While the order of the scenes was random, the order of the methods was only random for the first scene. For subsequent scenes the order was alternated so that the first method tried was not the same for two scenes in a row.

For each scene, the users were given 10 min to achieve the desired timing as best they could. They were allowed to watch the video of the desired timing as often as they needed to. The results were saved and the subjects were asked to fill out a questionnaire related to their experience using both methods in retiming that particular scene. A final questionnaire was presented to them when all four scenes were completed.

The first scene depicted a bouncing ball where the ball bounced a total of 5 times. The second scene showed the strokes resulting from an animated depiction of someone writing the three letters: O S U. The third scene showed a pencil character looking up, around, and back down. Finally, the last scene required that the user annotate a song with a bouncing circle cursor, much like that often shown in television commercials (Fig. 12).

Fig. 12. Starting with the top left, in clockwise order: bouncing ball, OSU, Jingle Bells, pencil.
4.3. Results

We were interested in learning two things from the user study data: with which method was the subject more accurate, and which method would the subject prefer to use if they had to perform the same task again.

Once a user completed a scene with a given method, the placement of their keyframes was compared to the desired placement. Each set of keyframes was treated as a vector of times. To account for the difficulty in the subject having to match the absolute time of the very first keyframe, we offset the subject’s timing by the difference between the subject’s first keyframe and the desired first keyframe. We then compared keyframes starting at keyframe number 2. We averaged the error over the number of keyframes in a particular scene. Since each of the four scenes had a different number of keyframes, this provided a consistent, meaningful metric across all scenes.

As shown in Table 2, most people achieved better results utilizing performance timing in the bouncing ball and pencil scenarios (18 out of 21 users). In the other two scenarios, the results were more evenly split.

We can hypothesize about why performance timing yielded better results on particular scenes and not on others. The bouncing ball scene involved very natural motion and was easy for the subjects to imitate. The animation was also quite slow and smooth, which allowed the user to react appropriately to the desired timing. The pencil scene included more 3D motion and was composed of faster movements with unpredictable velocity change. However, the motion was clearly deliberate and a result of the character’s intentions and therefore, subjects could rely on being able to mimic the intentions of the character (i.e. look around quickly). The OSU and Jingle Bells scenes were challenging for subjects in two aspects. The OSU scene had a series of discontinuities and the motion was in no way predictable as with the bouncing ball. The motion was also not personified in any way as with the pencil scene. Users struggled with bringing the pointer to the desired position in time due to the discontinuities. The Jingle Bells scene was the most dynamic in terms of speed and velocity changes required to match the desired timing. The precise rhythm of the song was a challenging one to match. However, errors, such as the one shown in the demonstration video, are quickly fixed with the manual feature editing tool.

It is also interesting to note that users performed better with performance timing on the scene that was truly 3D. In the pencil character scene, the motion path was a 3D path while the other three scenes included primarily motion in a 2D plane. The bouncing ball scene was truly 3D and most subjects viewed it as a 3D path, however, the motion could be simplified to a 2D view by rotating the scene appropriately.

We utilized the Signed-Rank Wilcoxon statistical test to analyze our results. We used the Signed-Rank Wilcoxon test to determine if there was a significant difference in subject performance, on a per scene basis, when using the two different timing methods. For the bouncing ball and the pencil scenes, subject performance with performance timing proved significantly better than the traditional approach.\footnote{Within 99\% certainty.} No significance could be found in the other two scenes. We attribute this to the reasons discussed above.

4.3.1. Wilcoxon Signed-Rank test for significance

The Signed Rank Wilcoxon test takes into consideration not only which method the user did better with, but it also provides some estimate of how well they performed compared to other users. Our independent variable was the timing method itself and has only two possible values: traditional or performance timing. Table 3 shows an example of the Signed Rank Wilcoxon test data for the Bouncing Ball scene. The value of $z$ is looked up in a table to determine the significance for each method.\footnote{Computing $z$ and determining what probability it corresponds to is a fairly straightforward method, described in introductory statistics textbooks.}

4.4. Subject comments and response

We also provided questionnaires for a more subjective comparison of the two methods. From our
questionnaire results, we can state that performance timing was preferred over the traditional method in the majority of cases. As Table 4 demonstrates, on average, subjects preferred performance timing on three out of the four scenes provided. The pencil scene produced the lowest user satisfaction rating (at 57%), yet users actually performed better on this scene with performance timing than with the traditional method (Table 2).

Similarly, Table 4 shows that subjects preferred performance timing when editing the OSU and Jingle Bells scenes. However, these are the two scenes on which the subjects performance timing result was not statistically significantly better than using the traditional method. Users generally preferred the method they perceived themselves as having performed the best with. In only a few cases, performance timing was preferred while at the same time being seen as less accurate than the traditional method.

Most subjects agreed that the performance timing interface was natural and intuitive, and that drawing what you see was a definite improvement over the traditional method. This can be determined from reading the subject comments. There was a distinct appreciation that there was no need to worry about the keyframes while drawing, and that the re-arranging was done behind the scenes. According to the subjects, this made it easier for them.

There were aspects of the interface that the subjects found problematic: the feature editing portion was seen as overly cumbersome and subjects were sometimes confused as to how to exit this editing mode. Subjects were also sometimes frustrated with determining how to correct mistakes that were made in the initial timing because there was no way to change the timing of a particular portion without modifying the timing that came afterward. Subjects felt that fine-tuning particular parts of the timing was something better suited to the traditional method because it allows one to shift one key at a time to exactly the desired position.

4.4.1. Suggested changes

The interface was frozen throughout the length of the study. A number of readily implementable changes were often reiterated by our subjects. A popular suggestion was to slow down the animation for timing purposes. This would allow the user to
react with more accuracy to the scene before her. The suggestion that we utilize a tablet PC or some other sort of drawable display (such as the Wacom Cintiq) was also common. Users felt that being able to draw directly on the screen would improve their accuracy and make it easier to draw their intended path. There would be no change in the actual implementation of the technique in this case.

Users considered feature editing the weakest part of the implementation and made suggestions for improvement. Suggestions included the idea of numbering the feature points to make it easier to determine where on the path they lie. Numbering the features would give the user a much more direct view of the correspondence between the two paths.

5. Discussion and future work

Performance timing is a novel approach to retiming keyframe animations via user performance. Performance timing is not meant to be a complete replacement for traditional methods of retiming motion, nor is it meant to be used in timing extremely precise motion. Performance timing relies almost exclusively on a “performance” given by the user and would not be appropriate for retiming elements which move faster than the average human can imitate. It would also not be appropriate in cases that require an irregular rhythm that is difficult for the user to characterize and imitate.

Animations of considerable length5 also pose an interesting problem. In particular, the user should not be required to animate an entire scene in one performance. In our current implementation, the user can time and map several portions of the animation in a piece-wise fashion. In doing so, the user must be sure that she picks appropriate ending points so that there are no apparent discontinuities in the motion. If the user overshoots an ending point or starts before a start point, the unwanted parts of the timing curve can be removed with the feature point editing utility. Another option is to allow the user to specify an overlap window for two parts of an animation and apply a blend, combining the old timing and part of the new timing. This addition is left for future work.

In our current implementation, we require that the user identify a view in which she wants to perform the timing. One possible improvement would be to provide the user with an “optimal” view in which to perform the timing. For example, the best view in which to perform the timing is generally one

Figure 13. In this example, the character’s head is animated by translating an inverse kinematics handle. The user has chosen this handle as the target object for the timing performance.

---

5 Any animation sequence the particular user cannot perform comfortably without making mistakes.
where the target object’s motion is mostly parallel to the viewplane with minimal motion perpendicular to the view plane. We could, therefore, attempt to identify an optimal view plane by computing a plane with the overall smallest distance from the target object’s path (least-squares fit plane). This extension is left as future work.

As mentioned previously, we currently require that an object have an imitable channel that exhibits translation. For example, in Fig. 13, the user is animating the rotation of the character’s head via the translation of a bone’s inverse kinematics handle. The imitable channel in this case is the handle’s translation. This translation requirement makes acting out the change in an attribute, such as color, currently impossible. One extension would be to map such an attribute to an artificial spatial channel, and treat that as a regular motion path. For an RGB color animation, for example, one could map \( x \leftarrow R \), \( y \leftarrow G \), \( z \leftarrow B \), create a motion path, and sketch the timing for the color changes.

We have limited ourselves to utilizing 2D input devices such as a mouse or a pen tablet for this research. We are mostly interested in this domain since we assume that it is a common setup, shared by the majority of users. However, we can also apply our algorithms to 3D timing paths. For example, one could imagine a vision-based system for tracking a single marker to be placed at the end of a timing wand. The user could act out the timing information by manipulating the end of the wand as an extension of her hand, much like a maestro leading an orchestra. This approach is similar to work in the audio synthesis domain [19].

An alternative to acting out the timing by drawing paths is to act out the motion by clicking the input device, such as the mouse, when the features should occur. Although this approach does not appear as intuitive as sketching the timing path, it would provide the relevant information needed to re-distribute the keyframes. However, possible valuable information is lost in this approach. With our current technique, we could analyze the user’s input to understand velocity changes between keyframes and use this information to adjust the animation curves to create effects such as ease-in and ease-out. For example, in a scene with a single sphere and only two keyframes, we can currently only generate a constant velocity path between those two keys. If we were to analyze the velocity profile of the user’s input, we could adjust the tangent handles of the motion curve to reflect that profile between the two keyframes. This would not be possible with the “clicking” approach.

In summary, performance timing is a novel approach for generating timing information for keyframed motion. In this paper, we have presented a more thorough exposition of the details of the approach and extended our original work with additional correspondence computation techniques as well as a user study.

It is important to note that this approach is designed to help a novice user generate animations without necessarily being artistically talented. Skilled animators are not the target audience for this form of animation timing. The user study has confirmed that novice users can retiming animations to their satisfaction on scenes of varying complexity and that in two of the four cases, can perform significantly better than with traditional timing methods.

Acknowledgments

The authors thank all of the subjects that participated in our user study. This work was supported in part by NSF CCF-0237706 and NSF CNS-0423733.

References

[1] AliasWavefront Maya, 1601 Cloverfield Blvd., 2nd Floor, South Tower, Santa Monica, CA 90404.


