The JGoodies Animation Framework
Time-based Real-time Animations

Karsten Lentzsch, February 10, 2003

1 Introduction

The JGoodies Animation framework enables you to produce sophisticated time-based real-time animations in Java. It uses concepts and notions as described by the W3C SMIL specification [1]. This animation framework has been designed for a seemless, flexible and powerful integration with Java, ease-of-use and a small library size. Unlike SMIL we use Java to describe the animations – not XML.

This document shortly introduces time-based animations and describes the framework and its components.

2 Time-based Animations

An animation is defined as a time-based change of some *attribute* of an *animation target* over a specified *duration*. An animation defines an *animation function* f(t), that maps time to values of the target attribute. For example, you can describe that during 10 seconds the width (target attribute) of a rectangle (animation target) grows from 10 to 50 pixels.

The animation function will be evaluated if required to set the result value as target attribute. Since animation functions are continuous over time, a render system can display the animation with different frame rates while maintaining the overall animation appearance.

The values of an animation function can be, for example, discrete or linear (see figures 1 and 2); motion animations often use paths of the host graphics system. Animation functions can be described in a variety of ways: by an algorithm, using random numbers, reading the external *wall clock time*, accessing the mouse position or other application state.

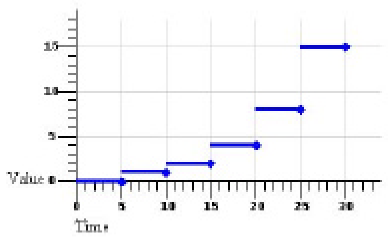

Fig.1: discrete function

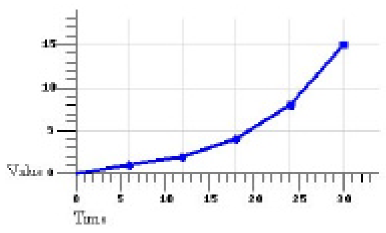

Fig. 2: linear function

A *composed animation* consists of simple animations and applies an effect to multiple targets and target attributes. The *animation effect* F(t) combines all animation functions with all other aspects and timings of a composed animation. It describes all target attribute values of all targets for each non-negative time. At the end of the animation duration the animation effect can be frozen or reset.

For example, you can say that a rectangle grows from 10 to 50 pixels then stays at 50 pixels and that the rectangle's color changes from gray to black during the first 5 seconds of this animation.

3 The Animation Framework

Package com.jgoodies.common.animation consists of all basic classes and interfaces of this framework. The AnimationFunction interface describes animation functions that map long time values to Java objects. The Class AnimationFunctions provides methods to create animation functions and operate on instances of AnimationFunction. For example, #discrete creates and answers a discrete animation function that is defined by a duration and a sequence of values. #linear returns an animation function, that is defined by a duration and a field of values and interpolates values. Both methods accept an optional array of key times; in this case the values won't be equally distributed over the duration, but will use the given key times as anchor points.

The Animation interface describes animations by their duration and an #animate method that applies the animation effect to the target attribute for a given time. In addition, you can register AnimationListener objects for an animation. The abstract superclass AbstractAnimation implements the Animation interface and eases the development of new animations. It manages AnimationListeners and fires AnimationEvents if the animation is started and stopped. And it provides behavior for freezing an animation by implementing the #animate method which in turn invokes the abstract method #applyEffect. A typical animation extends AbstractAnimation and implements #applyEffect. Class Animations operates on animations and provides animation factory methods; it can create pauses, can reverse an animation, and can compose animations either sequentially or parallel.

Class Animator is *a* prebuilt animation container that applies an effect as described by the given Animation at a specified frame rate using a javax.swing.Timer.

The renderer subpackage contains predefined renderers that paint shapes and typographic effects using Java2D. You can use these renderers as visual animation targets – either direct or indirect, for example from AWT or Swing components. To paint faster a basic text renderer caches GlyphVector objects.

The components subpackage contains subclasses of JComponent which integrate the renderers with Swing. The animations subpackage contains sample animations that combine some animation functions to apply a composed animation effect to the prebuilt animation components. Class BasicTextAnimations provides a bunch of factory methods that create text fading animations for a sequence of texts using and combining different effects: color fade, text scaling and glyph spacing, all which you can see live in the JGoodies Metamorphosis intro.

Package com.jgoodies.animation.demo provides a demo application that in turn uses a simple demo animation that just uses a JLabel to display the animation state over the animation duration. The Demo logs messages to the console whenever an AnimationEvent is fired. You can set the demo's frame rate with a command-line argument.

4 Future Directions

The current implementation of the JGoodies Animation framework is a good starting point for your elegant and eye-catching animations in Java. However, it lacks interesting features – most noticably a visual editor. I plan to carefully extend the framework in the future and will add more prebuilt animation functions, renderers, components and animations.

I plan to port animation functions and renderers that I've used in the old animation framework and would like to add animation functions for spline interpolation, sawtooth and trigonometric functions.

SMIL animation can synchronize animations, access the wall clock time and start/stop animations on mouse events. All this is possible with the JGoodies animation framework, now – but it lacks good support for it. Also, I'd like to synchronize visual animations with audio output, for example: click, boom, ssssst.

Your comments and suggestions regarding this framework are welcome and will help me improve this library.

References

[1] SMIL
http://www.w3.org/AudioVideo/