365: Animation

* 1. Overview.  
     Adding animations to a user interface can help visually indicate to a user that the state of the application has changed (in addition to just looking cool).  Judicially applied animations can help enhance a user interface greatly, and are an essential element of 2D or simple 3D games, which one can also build with Titanium.  In this module, learn several techniques for animating UI components.  
       
     In this module, students will:
* Learn to animate properties of UI controls
* Examine cross-platform differences in animation capabilities
* Create 2D matrix animations
* Create 3D matrix animations
* Explore when and why to use animation

1.2 Animation Effects  
An important part of designing a great UI is making the interface interactive and smooth.

Titanium provides built-in Animation capabilities that are optimized for each device. For example, on iOS devices, animations perform on the GPU and provide high performance capabilities.

1.3 Basic Animation: animate()

view.animate();

Any Window, View or UI component (TiViewProxy class) in Titanium can be animated with the **animate** method. This method will perform a set of animation against the view and animation events can be handled to perform complex timelines.

1.3.1 Basic Animation

Animation Properties

Used to implicitly apply an animation transformation on an object.

// animatable properties against what is being animated

zIndex number

left number

right number

top number

bottom number

width number

height number

duration number

center point

backgroundColor color (hex)

color color (hex)

opacity number

opaque number

visible number

// properties that control the animation

curve number

repeat number

autoreverse number

delay number

transform TiProxy

Transition number

1.3.2 Limitations

The animate object provides more fine grain listeners and also provides an easier ability to reuse animations.

1.4 Advanced Animation: 2D / 3D Matrix

What is a matrix? Help users understand the basics of a matrix. At the heart of a matrix is the concept of linear transformations.

Transformations are a fundamental part of computer graphics. Transformations are used to position objects, to shape objects, to change viewing positions, and even to change how something is viewed (e.g. the type of perspective that is used).

Matrices allow arbitrary linear transformations to be represented in a consistent format, suitable for computation. This also allows transformations to be concatenated easily (by multiplying their matrices).

Most common geometric transformations that keep the origin fixed are linear, including rotation, scaling, shearing, reflection, and orthogonal projection.

1.4.1 2D Matrix

Titanium provides a [2D Matrix](http://developer.appcelerator.com/apidoc/mobile/latest/Titanium.UI.2DMatrix-object) class for holding values for an affine transformation matrix. A 2D matrix is used to rotate, scale, translate, skew the objects in a two-dimensional space. A 2D matrix is represented by a 3 by 3 matrix. Because the third column is always (0,0,1), the data structure contains values for only the first two columns.

The [initial matrix](http://en.wikipedia.org/wiki/Identity_matrix) is always the "identity matrix" which is the simplest nontrivial diagonal matrix.

You can apply methods against the resulting matrix object to create a new representation. You must use the returned object.

You can also chain multiple function calls together since the return value is another 2D Matrix object.

1.4.2 3D Matrix

Titanium provides a [3D Matrix](http://developer.appcelerator.com/apidoc/mobile/latest/Titanium.UI.3DMatrix-object) class for holding values for an affine transformation matrix. This 3D matrix is a projection that maps three-dimensional points to a two-dimensional plane.

This 3D matrix is used to rotate, scale, translate, or skew the objects in a three-dimensional space. A 3D matrix is represented by a 4 by 4 matrix. Because the forth column is always (0,0,1), the data structure contains values for only the first three columns.

The 2D Matrix and 3D Matrix are similar in usage.

1.4.3 Transforms

To use the matrix, you simply call the transform property on a View. By setting the transform property, the change takes effect immediately.

1.5 Event Listeners & Callbacks

When performing an animation, you can pass in a function as a callback that will immediately fire upon completion.

view.animate({opacity:1,duration:300},function(){

// do something on complete

});

Or, alternatively, you can attach an event listener to the animation object.

var a = Titanium.UI.createAnimation();

a.addEventListener('start',func);

a.addEventListener('complete',func);

1.6 Opacity

When performing an animation, you will want to spend time on the aesthetics to make the animation seem more life-like. The opacity property is a good property to keep in mind when performing animations where you're changing the visual state between one or more Views.

The opacity property controls the transparency (or opacity) of the View. Typically, you'll bring the opacity from 0.0 to some end state, such as 1.0 during the animation. This will provide a "fade in" effect and make the View seem to "appear" on the screen.

In this example, the view is initially set to 0 - effectively making it fully transparent, or invisible. When the button is clicked, the view simply animates it's opacityproperty from 0.0 to 1.0, making it appear to fade in.

1.7 Cross Platform

1.8 Window Animation (iOS Only)

You can apply an animation to the following functions of a Window.

You can optionally apply animated transitions on child views of an existing window.

1.8.1 Window Transitions

You can also apply a Window transition using the transition property. Transitions provide a transition animation between two windows.

The transition property also applies to an Animation object and the animate function.

The following styles are supported on [Titanium.UI.iPhone.AnimationStyle](http://developer.appcelerator.com/apidoc/mobile/latest/Titanium.UI.iPhone.AnimationStyle-object):

* CURL\_DOWN: Curl downwards during a transition animation
* CURL\_UP: Curl upwards during a transition animation
* FLIP\_FROM\_LEFT: Flip from left to right during a transition animation
* FLIP\_FROM\_RIGHT: Flip from right to left during a transition animation
* NONE: No Animation

1.9 Design Patterns: Animations

1. Window transitions
2. View Opacity (fade in/out)
3. Sliding left/right, up/down
4. Bounce, Grow, Shrink
5. Drag / Drop