Drawing And Animations

Terms: CALayer, NSBezierPath,

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

About Animation

- Animations provide fluid visual transitions between different states of your user interface
- In iOS, animations are used extensively to reposition views, change their size, remove them from view hierarchies, and hide them
- You might use animations to convey feedback to the user or to implement interesting visual effects

About Animation

- In iOS, creating sophisticated animations does not require you to write any drawing code
- All of the animation techniques described in this chapter use the built-in support provided by Core Animation
- All you have to do is trigger the animation and let Core Animation handle the rendering of individual frames
- In iPhone there are various type of animation and their delegate method.

What Can Be Animated?

- Both UIKit and Core Animation provide support for animations, but the level of support provided by each technology varies
- In UIKit, animations are performed using UIView objects
- Views support a basic set of animations that cover many common tasks

UIKit:UIView Animation

- There are two different ways to initiate animations:
  - In iOS 4 and later, use the block-based animation methods. (Recommended)
  - Use the begin/commit animation methods
- The following properties of the UIView class are animatable:
  - frame : The frame rectangle describes the view's location and size in its superview's coordinate system
  - bounds: The bounds rectangle describes the view's location and size in its own coordinate system
  - center: The center of the frame.
  - transform: Specifies the transform applied to the receiver, relative to the center of its bounds.
  - alpha: floating-point number in the range 0.0 to 1.0, where 0.0 represents totally transparent and 1.0 represents totally opaque.
  - backgroundColor:

Animatable UIView Properties

Table 4-1 Animatable `UIView` properties

Property	Changes you can make
`frame`	Modify this property to change the view's size and position relative to its superview's coordinate system. (If the `transform` property does not contain the identity transform, modify the `bounds` or `center` properties instead.)
`bounds`	Modify this property to change the view's size.
`center`	Modify this property to change the view's position relative to its superview's coordinate system.
`transform`	Modify this property to scale, rotate, or translate the view relative to its center point. Transformations using this property are always performed in 2D space. (To perform 3D transformations, you must animate the view's layer object using Core Animation.)
`alpha`	Modify this property to gradually change the transparency of the view.
`backgroundColor`	Modify this property to change the view's background color.
`contentStretch`	Modify this property to change the way the view's contents are stretched to fill the available space.

Methods For Animation

Table 4-2 Methods for configuring animation blocks

Method	Usage
`setAnimationStartDate:` `setAnimationDelay:`	Use either of these methods to specify when the executions should begin executing. If the specified start date is in the past (or the delay is 0), the animations begin as soon as possible.
`setAnimationDuration:`	Use this method to set the period of time over which to execute the animations.
`setAnimationCurve:`	Use this method to set the timing curve of the animations. This controls whether animations execute linearly or change speed at certain times.
`setAnimationRepeatCount:` `setAnimationRepeatAutoreverses:`	Use these methods to set the number of times the animation repeats and whether the animation runs in reverse at the end of each complete cycle. For more information about using these methods, see "Implementing Animations That Reverse Themselves."

`setAnimationDelegate:`
`setAnimationWillStartSelector:`
`setAnimationDidStopSelector:`
	Use these methods to execute code immediately before or after the animations. For more information about using a delegate, see "Configuring an Animation Delegate."

`setAnimationBeginsFromCurrentState:`	Use this method to stop all previous animations immediately and start the new animations from the stopping point. If you pass `NO` to this method, instead of `YES`, the new animations do not begin executing until the previous animations stop.

HOW TO ANIMATE WITH A UIVIEW

```
[UIView beginAnimations:nil context:NULL];

[UIView setAnimationDuration:1.0];
[UIView setAnimationRepeatCount:10];
[UIView setAnimationRepeatAutoreverses:YES];

CGPoint position = myObject.center;
position.y = 200.0f;
position.x = 150.0f;
myObject.center = position;

[UIView commitAnimations];
```

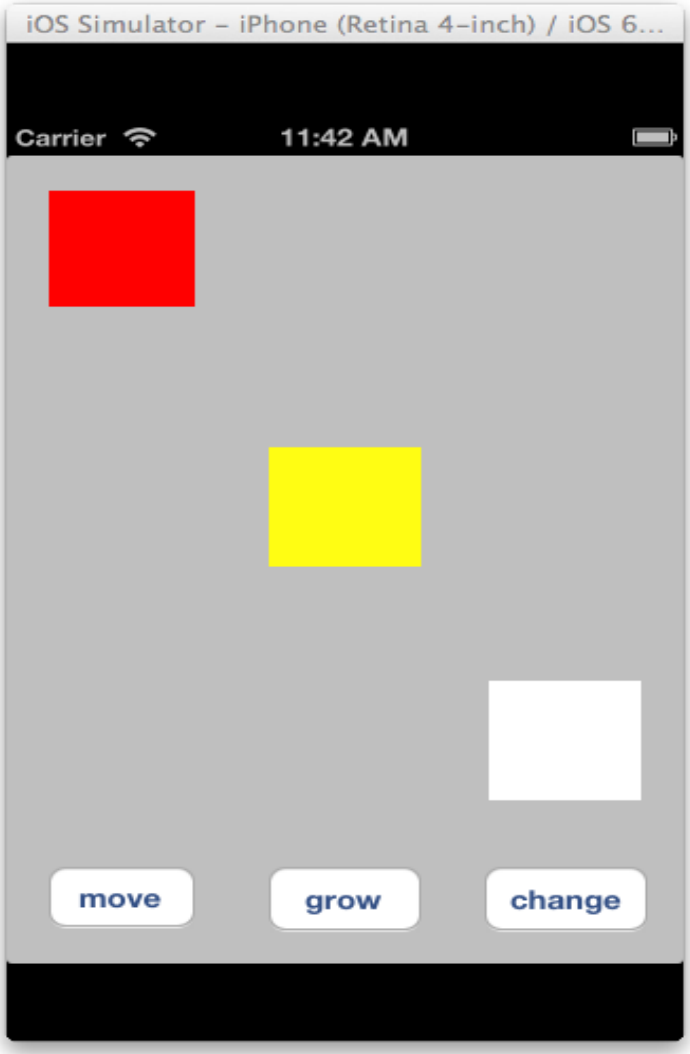
Animaton Block

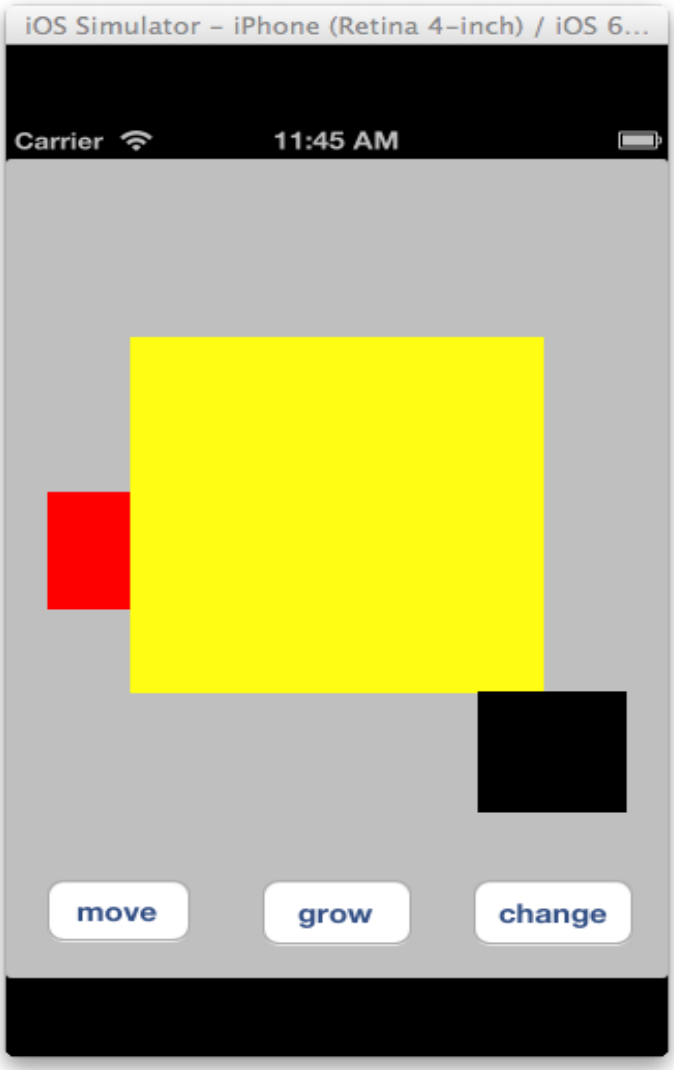
UIView Animation: Block Based Methods

- Block-based methods are class method and offer different levels of configuration for the animation block. These methods are:
- animateWithDuration:animations:
- animateWithDuration:animations:completion:
- animateWithDuration:delay:options:animations:completion:

```
[UIView animateWithDuration:1.0
        delay: 0.0
        options: UIViewAnimationOptionCurveEaseIn
        animations:^{
            firstView.alpha = 0.0;
            secondView.alpha = 1.0;}];
```

View Based Animation





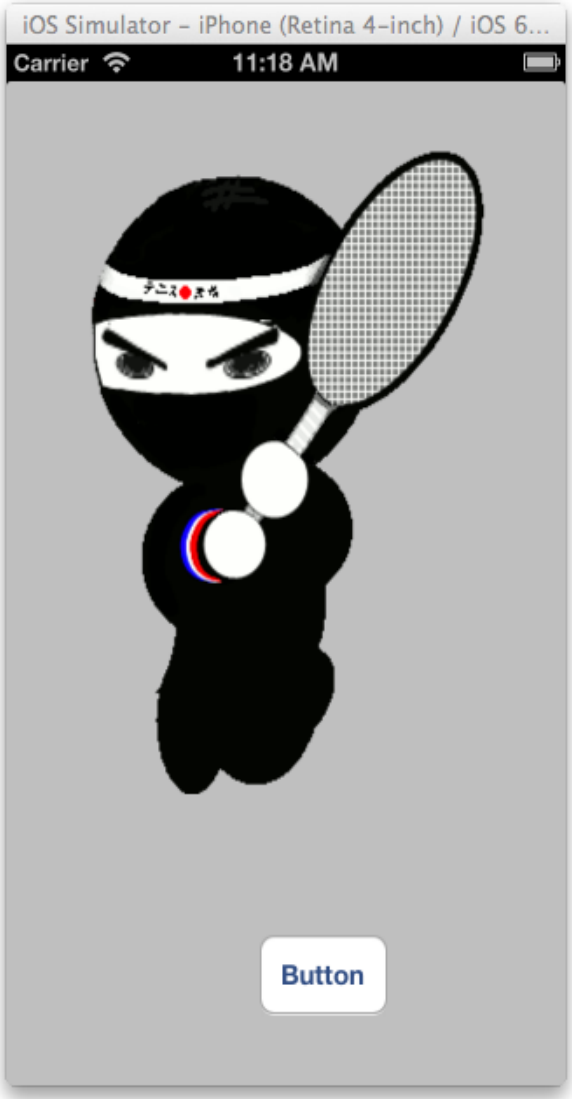
Frame By Frame Animation

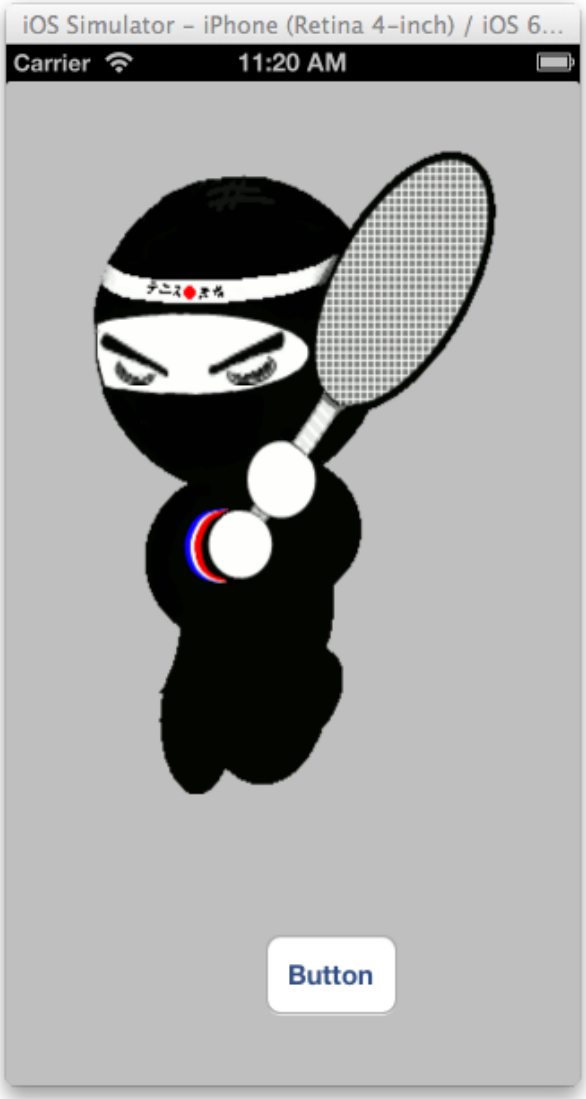
- Frame by frame animation is UIImage View based animation
- One UIImage View will contain array of images
- Images in UIImage View will be added by add animationimages property
- Frame by Frame animation can be startd by sending message(startAnimating) to UIImage View

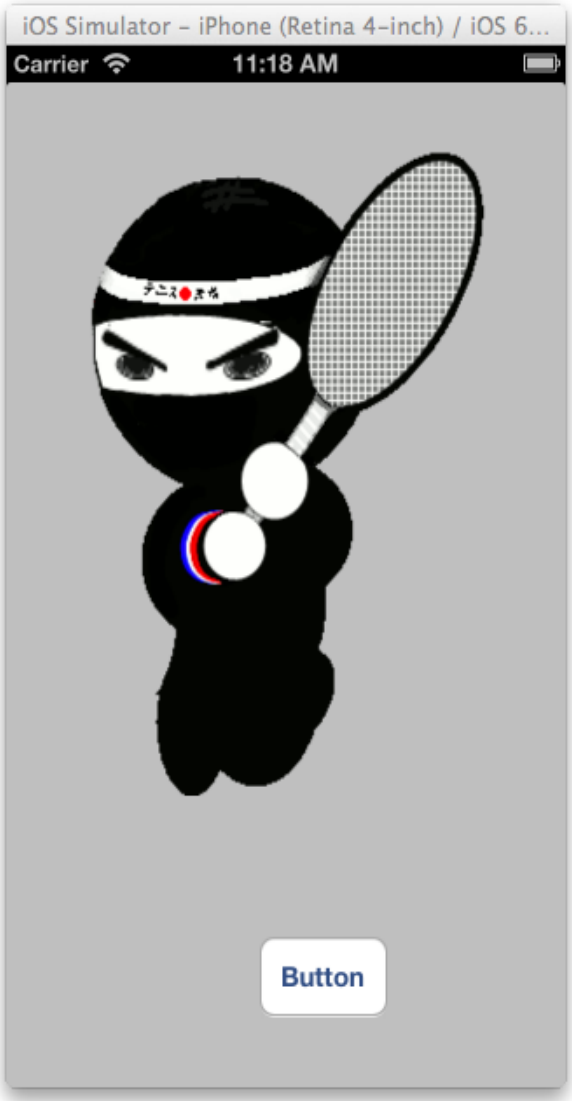
Frame By Frame Animation


```
// Load images
NSArray *imageNames = @[@"win_1.png", @"win_2.png", @"win_3.png", @"win_4.png", @"win_5.png", @"win_6.png", @"win_7.png", @"win_8.png",];

NSMutableArray *images = [[NSMutableArray alloc] init];
for (int i = 0; i < imageNames.count; i++) {
    [images addObject:[UIImage imageNamed:[imageNames objectAtIndex:i]]];
}
// Normal Animation
UIImageView *animationImageView = [[UIImageView alloc] initWithFrame:CGRectMake(60, 95, 86, 193)];
animationImageView.animationImages = images;
animationImageView.animationDuration = 0.5;
[self.view addSubview:animationImageView];
[animationImageView startAnimating];
```

Frame By Frame Animation



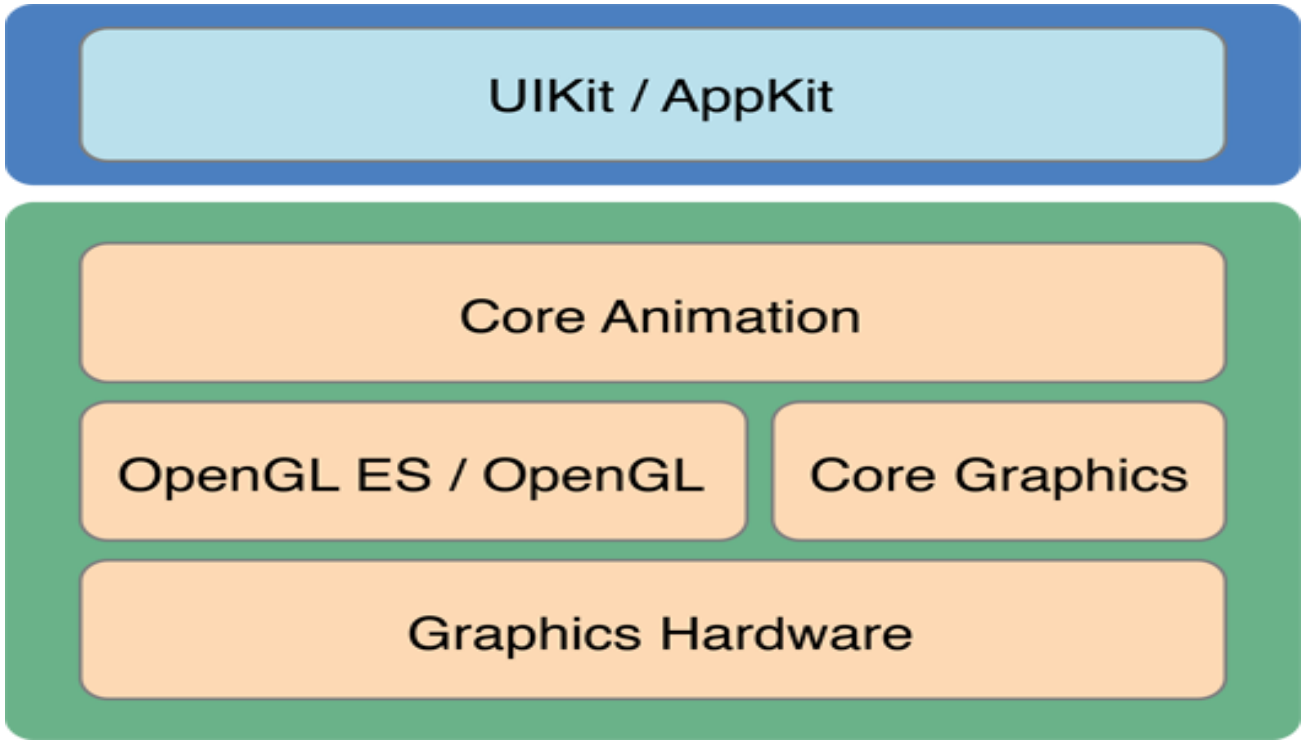




Core Animation

- Core Animation provides a general purpose system for animating views and other visual elements of your app
- Core Animation is not a replacement for your app's views
- Instead, it is a technology that integrates with views to provide better performance and support for animating their content
- It achieves this behavior by caching the contents of views into bitmaps that can be manipulated directly by the graphics hardware

Core Animation…



- There are two classes that make Core Animation work:

CALayer and CAAnimation

- To use any part of Core Animation, you need to add the

QuartzCore framework to your project

CALayer

- The CALayer class manages image-based content and allows you to perform animations on that content
- Layers are often used to provide the backing store for views but can also be used without a view to display content
- A layer's main job is to manage the visual content that you provide but the layer itself has visual attributes that can be set, such as a background color, border, and shadow

CALayer…

- The neat thing about the CALayer class is that it contains a lot of properties that you can set on it to affect the visual appearance, such as:
  - The size and position of the layer
  - The layer's background color
  - The contents of the layer (an image, or something drawn with core graphics

  For more on CALayer visit:

  http://www.raywenderlich.com/2502/calayers-tutorial-for-ios-introduction-to-calayers-tutorial

CALayer Animatable Properties

- Opacity
- Position
- Transform
- Backgroundcolor
- Bounds
- Frame
- For more visit link:

https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CoreAnimation_guide/AnimatableProperties/AnimatableProperties.html

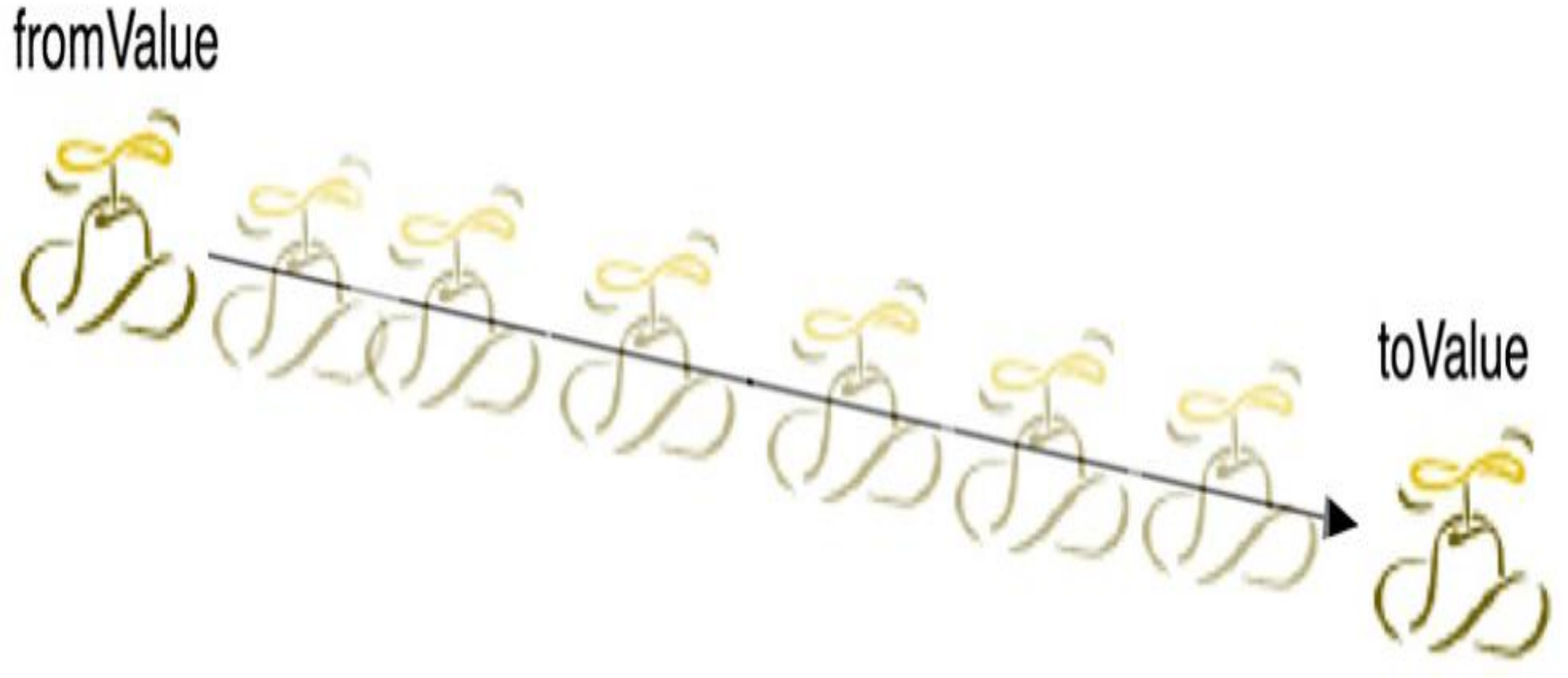
CABasicAnimation

- It has two properties: fromValue and toValue, and it inherits CAAnimation's duration property
- When a basic animation is added to a layer, the property to be animated is set to the value in fromValue
- Over the time specified by duration, the value of the property is interpolated linearly from fromValue to toValue

CABasicAnimation

```
CABasicAnimation* fadeAnim = [CABasicAnimation animationWithKeyPath:@"opacity »] ;
 fadeAnim.fromValue = [NSNumber numberWithFloat:1.0];
 fadeAnim.toValue = [NSNumber numberWithFloat:0.0];
 fadeAnim.duration = 1.0;
fadeAnim.repeatCount=20 ;

//    CALayer *theLayer=self.button10.layer ;

 CALayer *theLayer=self.view.layer ;
 [theLayer addAnimation:fadeAnim forKey:@"opacity"];

 // Change the actual data value in the layer to the final value.

 theLayer.opacity = 1.0;
```

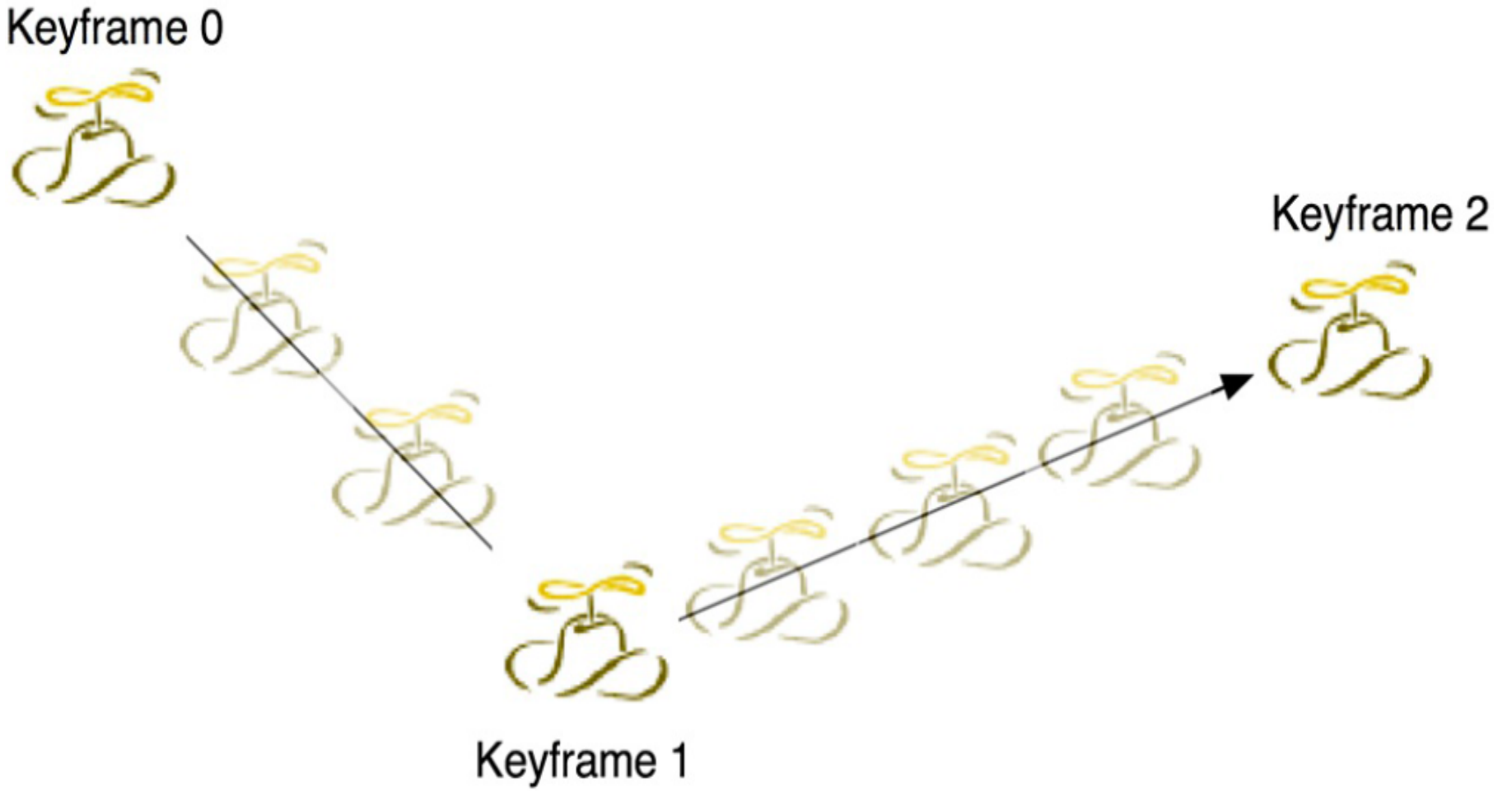
CABasicAnimation



CAKeyframeAnimation

- The difference between CABasicAnimation and CAKeyframeAnimation is that a basic animation only interpolates two values while a keyframe animation can interpolate as many values as you give it
- These values are put into an NSArray in the order in which they are to occur
- This array is then set as the values property of a CAKeyframeAnimation instance

CAKeyframeAnimation



Other Animations

- Group Animation
  - Various animation can be grouped together
- Transform Animation
  - Can be added to basic as well to keyframe animation

For Key Path Support for Structures visit link:

https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CoreAnimation_guide/Key-ValueCodingExtensions/Key-ValueCodingExtensions.html

Drawing

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

Custom Drawing

- High-quality graphics are an important part of a well-designed application.
- High-quality graphics is one of the things that sets Mac OS X apart from many other operating systems.
- Mac OS X uses color, transparency, and its advanced compositing system to give programs a more fluid and inviting appearance.

Custom Drawing....

- Drawing is only one step in the process of creating a fully functional Cocoa view.
- Drawing is a fundamental part of most Cocoa applications.
- Cocoa does maximum parts of the drawing for you.

Cocoa Drawing Support

- The Cocoa drawing environment is available to all applications.
- There are numerous classes and functions for drawing everything from primitive shapes to complex images and text.
- The Cocoa drawing environment is compatible with all of the other drawing technologies in Mac OS X

The Painter's Model

- Cocoa drawing uses the painter's model for imaging.
- In the painter's model successive drawing operation applies a layer of "paint" to an output "canvas.
- This model allows you to construct extremely sophisticated images from a small number of powerful primitives.

The Graphics Context

- A graphics context encapsulates all of the information needed to draw to an underlying canvas.
- graphics contexts are represented by the NSGraphicsContext class.
- The most common drawing destination is your application's windows.
- Most drawing occurs on your application's main thread.

The Graphics Context…..

- Graphics context objects make it possible to draw from secondary threads as well.
- You can also create graphics contexts explicitly.
- There also some are other ways to create graphics context objects explicitly.

Color and Color Spaces

- Color is an important part of drawing.
- Before drawing any element, you must choose the colors.
- Cocoa provides complete support for specifying color information.
- There are several different color spaces.

Basic Drawing Elements

- The creation of complex graphics often has a simple beginning.
- In Cocoa, everything you draw is derived from a set of basic elements.
- These elements are fundamental to all drawing operations and are described in the following sections.

Shape Primitives

- Cocoa provides support for drawing shape primitives with the NSBezierPath class.
- You can use this class to create several shapes.
- Lines
- Rectangles
- Ovals and circles
- Arcs
- Bezier cubic curves
- For more on NSBezierPath Class Reference Visit link:

https://developer.apple.com/library/mac/documentation/Cocoa/Reference/ApplicationKit/Classes/NSBezierPath_Class/Reference/Reference.html