Swift Mac OS Animation using Core Animation

ddas.tech/swift-mac-os-animation-using-core-animation/

November 20, 2022

Created By: Debasis Das (20-Nov-2022)

Table of Contents

Introduction to Mac OS Animation

Animation in Mac OSX has a long history and there are more than one option for achieving an animated behavior in a Mac Application.

We can achieve animation in a Mac Application using

- **Simple View Animation**
- **Using Animation Proxy or**
- **Using Core Animation on CALayer**

The above animation techniques can be used in isolation or it can be merged with each other to achieve a certain animation behavior

The decision to use one approach vs the other purely depends on the animation complexity and the degree of control that is desired. Whether user interaction is desired on the screen that has animated layers etc.

If a simple animation such as animating the size of the NSWindow is required, it would be easier to simply use an animator proxy to achieve the functionality, however if we have complex requirement of creating a firework effect, we would need to look beyond View Animation or using an animator proxy.

In this post we will see how Core Animation works in Mac OSX.
CAAnimation is the abstract superclass for all Core Animations.
CAAnimation has the following subclasses

- CABasicAnimation
- CAKeyframeAnimation
- CAAnimationGroup
- CATransition

We can animate the contents of our applications by attaching animations (Stated above) with Core Animation Layers

CABasicAnimation

- Provides basic single-keyframe animations to the CALayers properties.
- While initializing a CABasicAnimation we state the keypath of the property that we want to animate.
- The property can be the backgroundColor, it can be the layer opacity or border color etc.
- The animation has a from and a to value that need to be stated.
- for example, we can animate the color change of a CALayer from red to green by creating a CABasicAnimation with ***backgroundColor*** keypath and then state the fromValue as red and toValue as green

CAKeyframeAnimation

- Is similar to CABasicAnimation with a difference that it can accept multiple intermediate values and multiple intermediate keyTimes that controls how the transition happens
- The timing and pacing of keyframe animations are complex than the basic animations.
- There is a property of CAkeyframeAnimation called as **calculationMode** which defines the algorithm of the animation timing.
- Below are the calculation modes
    - **CAAnimationCalculationMode.linear** – provides a linear calculation between keyframe value
    - **CAAnimationCalculationMode.discrete** – each keyframe value is used in turn and no interpolated values are calculated.
    - **CAAnimationCalculationMode.paced** – Linear keyframe values are interpolated to produce an even pace throughout the animation.
    - **CAAnimationCalculationMode.cubic** – Smooth spline calculation between keyframe values
    - **CAAnimationCalculationMode.cubicPaced** – Cubic keyframe values are interpolated to produce an even pace throughout the animation.

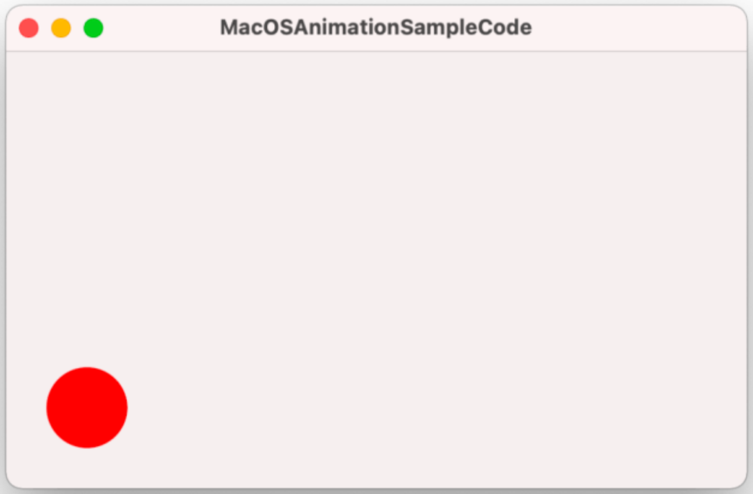
- The decision of the calculationMode plays a key role based on what type of animation we are trying to achieve. ***A bouncing ball effect would require the ball to fall at a slow speed initially and gradually the speed should increase and when it hits the ground it should bounce back with initial higher speed and the speed should taper at the top before it reverses direction.***

CAAnimationGroup

- Allows multiple animations to be grouped and run concurrently.
- We can create multiple animations using CABasicAnimation or CAKeyframeAnimation each having a different animation duration and then we can create a CAAnimationGroup using an array of individual animations.
- The CAAnimationGroup also has a duration property which if smaller than individual animation durations will clip the individual animation durations.

Using a combination of CABasicAnimation, CAKeyframeAnimation and CAAnimationGroup we can achieve amazing animation effects.

Lets check some animation behaviors using code



Initial Circle CALayer

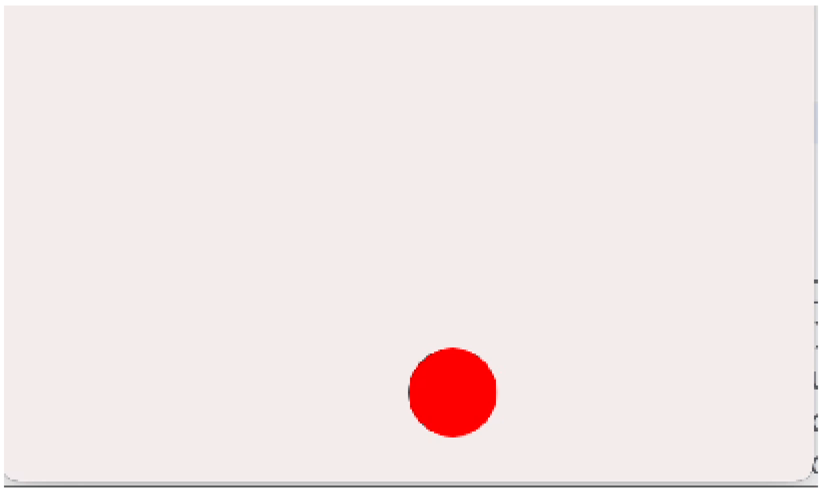
Below is the function to initialize the circle layer that we will be animating using different function calls

```
let circleLayer = CALayer()
// In the viewDidLoad method we will set the wantsLayer to be true
self.window.contentView?.wantsLayer = true

// Lets initialize the circle Layer
func initializeCircleLayer(){
        self.circleLayer.bounds = CGRect(x: 0, y: 0, width: 50, height: 50)
        self.circleLayer.position = CGPoint(x: 50, y: 50)
        self.circleLayer.backgroundColor = NSColor.red.cgColor
        self.circleLayer.cornerRadius = 25.0
        self.window.contentView?.layer?.addSublayer(circleLayer)
    }
```

Now lets add some basic animation to the above red Circle

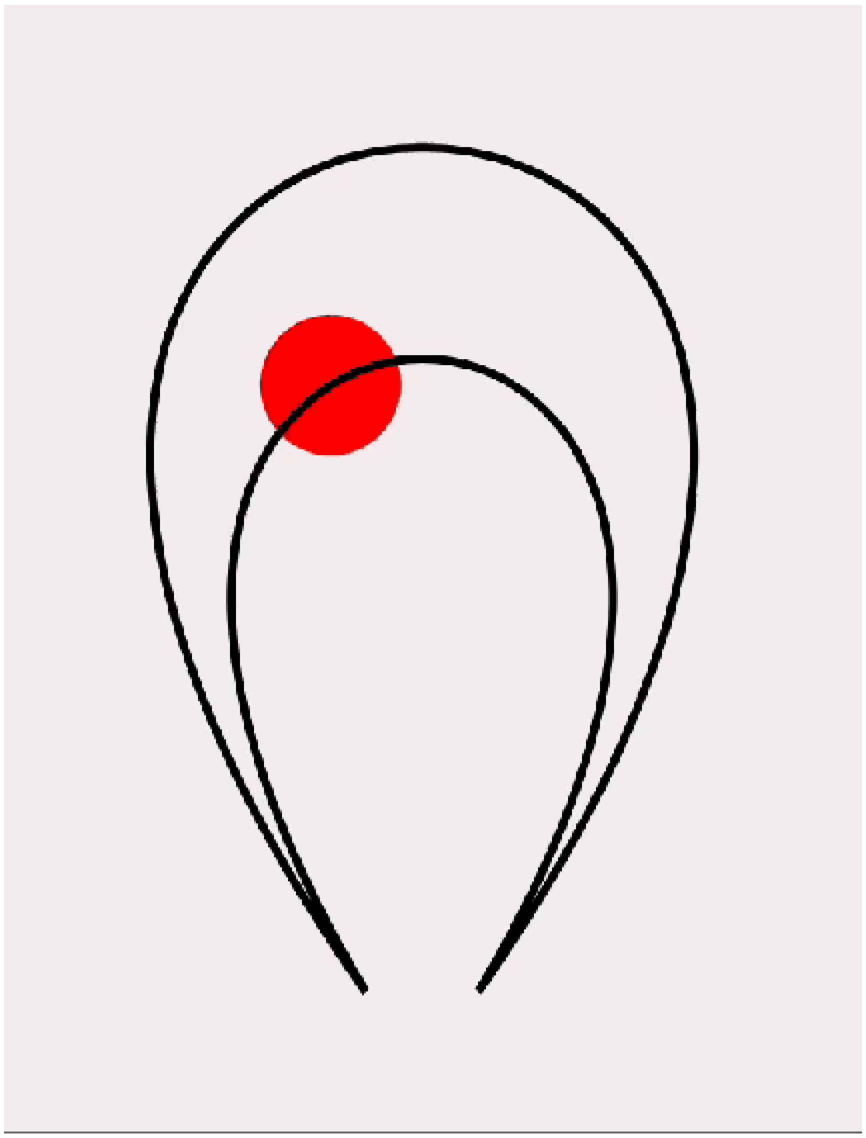
CABasicAnimation Sample Code



CABasicAnimation Sample

```
func simpleCAAnimationDemo(){
        self.circleLayer.removeAllAnimations()
        let animation = CABasicAnimation(keyPath: "position")
        let startingPoint = NSValue(point: NSPoint(x: 50, y: 50))
        let endingPoint = NSValue(point: NSPoint(x: 400, y: 50))
        animation.fromValue = startingPoint
        animation.toValue = endingPoint
        animation.repeatCount = Float.greatestFiniteMagnitude
        animation.duration = 3.0
        self.circleLayer.add(animation, forKey: "linearMovement")
    }
```

CAKeyframeAnimation Sample Code



CAKeyframeAnimation Sample

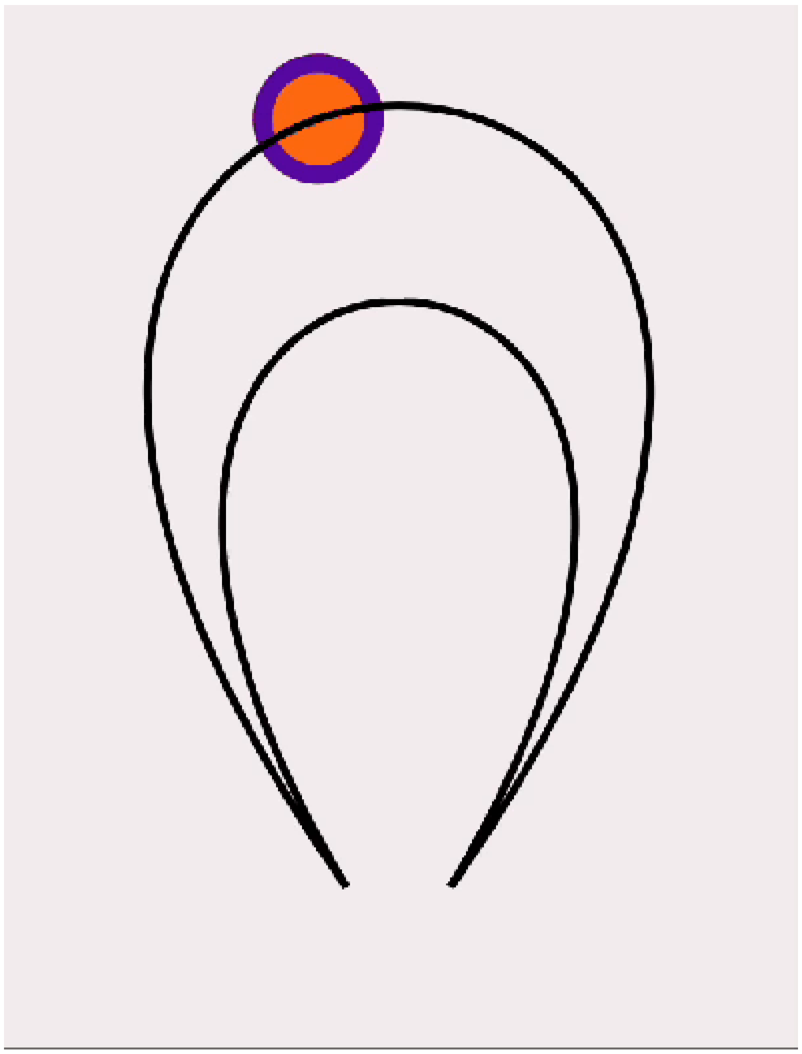
```
func keyFrameAnimationDemo(){
        self.circleLayer.removeAllAnimations()
        let path = CGMutablePath()
        path.move(to: CGPoint(x: 280, y: 100))
        path.addCurve(to: CGPoint(x: 320, y: 100), control1: CGPoint(x: 100, y: 400),
control2: CGPoint(x: 500, y: 400))
        path.addCurve(to: CGPoint(x: 280, y: 100), control1: CGPoint(x: 600, y: 500),
control2: CGPoint(x:0, y: 500))


        let shapeLayer = CAShapeLayer()
        shapeLayer.path = path
        shapeLayer.lineWidth = 3.0
        shapeLayer.fillColor = NSColor.clear.cgColor
        shapeLayer.strokeColor = NSColor.black.cgColor
        self.window.contentView?.layer?.addSublayer(shapeLayer)

        let animation = CAKeyframeAnimation(keyPath: "position")
        animation.calculationMode = CAAnimationCalculationMode.linear
        animation.path = path
        animation.repeatCount = Float.greatestFiniteMagnitude
        animation.autoreverses = true
        animation.duration = 3.0
        self.circleLayer.add(animation, forKey: "KeyFrameMovement")

    }
```

CAAnimationGroup Sample Code



CAAnimationGroup Sample Code

```
func groupedAnimationDemo(){
        self.circleLayer.removeAllAnimations()
        let path = CGMutablePath()
        path.move(to: CGPoint(x: 280, y: 100))
        path.addCurve(to: CGPoint(x: 320, y: 100), control1: CGPoint(x: 100, y: 400),
control2: CGPoint(x: 500, y: 400))
        path.addCurve(to: CGPoint(x: 280, y: 100), control1: CGPoint(x: 600, y: 500),
control2: CGPoint(x:0, y: 500))


        let shapeLayer = CAShapeLayer()
        shapeLayer.path = path
        shapeLayer.lineWidth = 3.0
        shapeLayer.fillColor = NSColor.clear.cgColor
        shapeLayer.strokeColor = NSColor.black.cgColor
        self.window.contentView?.layer?.addSublayer(shapeLayer)

        let widthAnimation = CAKeyframeAnimation(keyPath: "borderWidth")
        let widthValues = [2.0,4.0,6.0,8.0,6.0,4.0,2.0,4.0,6.0,8.0,6.0,4.0,2.0]
        widthAnimation.values = widthValues
        widthAnimation.calculationMode = CAAnimationCalculationMode.paced

        let positionAnimation = CAKeyframeAnimation(keyPath: "position")
        positionAnimation.calculationMode = CAAnimationCalculationMode.linear
        positionAnimation.path = path

        let colorAnimation = CAKeyframeAnimation(keyPath: "backgroundColor")
        let colors = [NSColor.green.cgColor, NSColor.yellow.cgColor,
NSColor.red.cgColor]
        colorAnimation.values = colors
        colorAnimation.calculationMode = CAAnimationCalculationMode.paced

        let borderColorAnimation = CAKeyframeAnimation(keyPath: "borderColor")
        let borderColors = [NSColor.black.cgColor, NSColor.red.cgColor,
NSColor.blue.cgColor]
        borderColorAnimation.values = borderColors
        borderColorAnimation.calculationMode = CAAnimationCalculationMode.paced


        let groupAnimation = CAAnimationGroup()
        groupAnimation.animations =
[widthAnimation,positionAnimation,colorAnimation,borderColorAnimation]
        groupAnimation.duration = 5.0
        groupAnimation.repeatCount = Float.greatestFiniteMagnitude
        groupAnimation.autoreverses = true
        self.circleLayer.add(groupAnimation, forKey: "multiAnimation")
    }
```

Animator Proxy

- Some simple Mac OSX Animation can be achieved through the usage of Animator Proxy.

- Animator proxy is the quickest and easiest way to implement animation effects in views and windows.
- The animations caused by using animator proxy is called as Cocoa Animation or Cocoa Animatable Proxy Animation.
- Animator proxy provides a benefit to achieve animations without using core animation layers.
- The way animator proxy works is rather than an UI object changing its own size or origin asks the animator proxy to change its properties values using an animation effect

Lets start with Creating Custom NSView Class that is blue in color

```
class ColoredView:NSView{
    override func draw(_ dirtyRect: NSRect) {
        super.draw(dirtyRect)
        NSColor.blue.setFill()
        NSBezierPath(rect: self.bounds).fill()
    }
}
```

Changing a NSView’s Frame Origin without Animation

```
func moveViewWithoutAnimation(){
        var origin = self.coloredView.frame.origin
        origin.x += 300
        self.coloredView.setFrameOrigin(origin)
    }
```

Simple View Animation using NSView Animator with Default Duration

```
func moveViewWithAnimationDefaultDuration(){
        var origin = self.coloredView.frame.origin
        origin.x += 300
        self.coloredView.animator().setFrameOrigin(origin)
    }
```

Simple View Animation with Custom Duration

```
func moveViewWithAnimationCustomDuration(){
        NSAnimationContext.beginGrouping()
        NSAnimationContext.current.duration = 3.0
        var origin = self.coloredView.frame.origin
        origin.x += 300
        self.coloredView.animator().setFrameOrigin(origin)
        NSAnimationContext.endGrouping()
    }
```

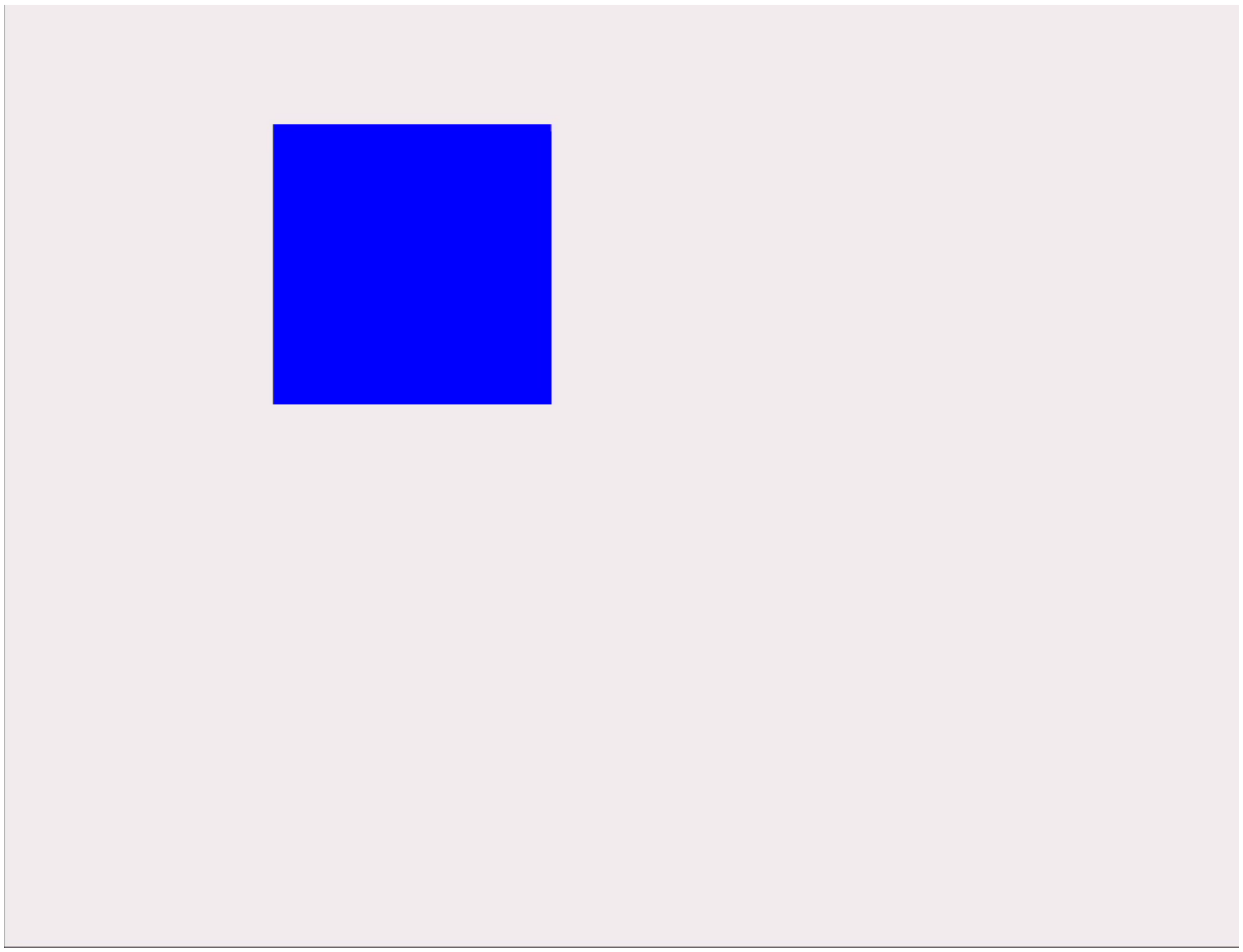
Animation Grouping

```
func moveWithWithNestedAnimations(){
        NSAnimationContext.beginGrouping()
        NSAnimationContext.current.duration = 3.0
        var origin = self.coloredView.frame.origin
        origin.x += 300
        self.coloredView.animator().setFrameOrigin(origin)

            NSAnimationContext.beginGrouping()
            NSAnimationContext.current.duration = 4.0
            var size = self.coloredView.frame.size
            size.height *= 2
            size.width *= 2
            self.coloredView.animator().setFrameSize(size)
            NSAnimationContext.endGrouping()

        NSAnimationContext.endGrouping()
    }
```

Animator Proxy and CAKeyFrameAnimations



Animator Proxy and CAKeyFrameAnimations

```
func animatorProxyAndKeyframeAnimations(){
        let posAnimation = CAKeyframeAnimation()
        posAnimation.duration = 3.0
        let x = self.coloredView.frame.origin.x
        let y = self.coloredView.frame.origin.y
        posAnimation.values = [CGPoint(x: x, y: y),CGPoint(x: x+100, y: y),CGPoint(x: 
x+200, y: y),CGPoint(x: x+200, y: y+100)]
        posAnimation.keyTimes = [0.0,0.5,0.8,1.0]
        posAnimation.timingFunctions = [CAMediaTimingFunction(name: 
.linear),CAMediaTimingFunction(name: .linear),CAMediaTimingFunction(name: .easeIn)]
        posAnimation.autoreverses = true
        posAnimation.repeatCount = Float.greatestFiniteMagnitude

        let sizeAnimation = CABasicAnimation(keyPath: "frameSize")
        sizeAnimation.fromValue = self.coloredView.frame.size
        sizeAnimation.toValue = CGSize(width: 200, height: 200)
        sizeAnimation.duration = 3.0
        sizeAnimation.autoreverses = true
        sizeAnimation.repeatCount = Float.greatestFiniteMagnitude

        var existingAnimations = self.coloredView.animations
        existingAnimations["frameOrigin"] = posAnimation
        existingAnimations["frameSize"] = sizeAnimation
        self.coloredView.animations = existingAnimations
        self.coloredView.animator().frame = CGRect(x: x+300, y: y+100, width: 200, 
height: 200)
    }
```