

Căn bản về Graphics

📅 Date	@September 2, 2021 → September 3, 2021
🕒 Date Created	@August 13, 2021 7:30 AM
📌 Status	Pending

Nghiên cứu các khái niệm trong Graphics

PHẦN OVERVIEW

▼ Các khái niệm căn bản trong hệ thống giao diện

- ✓ Các khái niệm trong **Core graphics**

▼ iOS

- ✓ Các thuộc tính trong **Scroll View**
- ✓ Các thuộc tính trong **View**
- ✓ Các thuộc tính trong **Layer**

▼ Các phương pháp thực hiện biến đổi trên View

- ✓ Animation

`animate(withDuration:delay:usingSpringWithDamping:initialSpringVelocity:options:animations:completion:)`

- ✓ Transition

`UIView.transition(from:to:duration:options:completion:)`

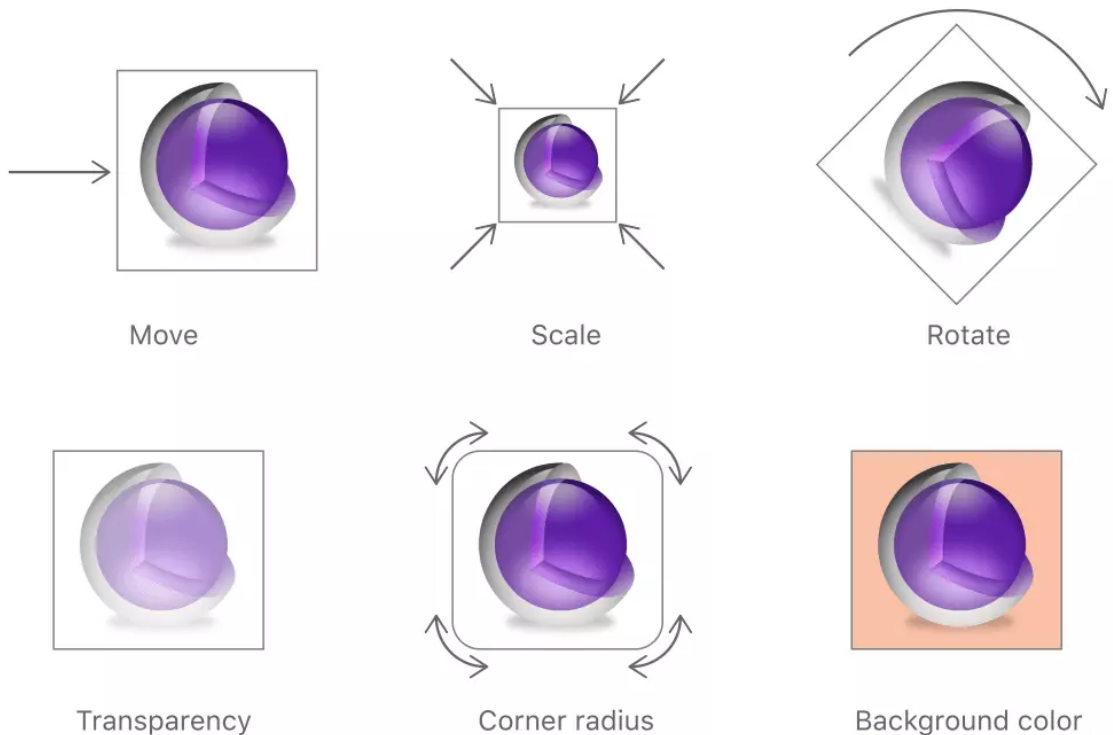
- ✓ Animation sử dụng Keyframes

`animate(withDuration:delay:usingSpringWithDamping:initialSpringVelocity:options:animations:completion:)`

PHẦN CHI TIẾT

▼ Các khái niệm căn bản trong hệ thống giao diện

- ▼ Các khái niệm trong **Core graphics**



▼ Định nghĩa **Point**

Khái niệm được Apple dùng để phản ánh độ phân giải của màn hình. Nhưng đơn vị điểm ảnh để nhắc đến vẫn là pixel. Point chỉ là một cách thay thế cho việc sử dụng Pixel. Đối với các sản phẩm của Apple, các loại màn hình có cùng kích thước sẽ có cùng số Point ảnh. Mỗi Point này sẽ tương ứng với số điểm ảnh được quy định cho dòng máy đó. (Ví dụ: trên các iphone đời đầu thì tỉ lệ là 1:1 còn trên iphone retina(5 5s 6 6s) thì tỉ lệ là 1:2, trên iphone 7+ tỉ lệ là 1:3)

▼ Định nghĩa **CGRect**

Core Graphic Rectangle đề cập đến vị trí của một view và chiều dài, chiều rộng của nó. Trong định nghĩa này ta quan tâm đến 2 thứ: position và size được định nghĩa tương ứng trong iOS lần lượt là **origin** và **size**.

▼ iOS

▼ Các thuộc tính trong **Scroll View**

```
// Hãy tắt chế độ auto layout trong lúc thử nghiệm
// Hãy thử đặt lại.contentSize lớn hơn so với màn hình (Ta sẽ có thể scroll ngang và scroll dọc)
// Nguyên nhân: Do kích thước của scrollView(contentSize) lúc này lớn hơn kích thước hiển thị
scrollViewSample.contentSize = CGSize(width: 800, height: 940)

// contentOffset là một điểm dữ liệu để mô tả vị trí của khung nhìn so với scroll view (theo tọa độ góc bên trái trên top)
// Khi ta set tọa độ này -100 theo x => Khung nhìn bị dịch về bên phải 100 so với (góc bên trái trên top) scrollView
// Khi ta set tọa độ này -100 theo y => Khung nhìn bị dịch xuống dưới 100 so với (góc bên trái trên top) scrollView
scrollViewSample.contentOffset = CGPoint(x: -100, y: -100)

// contentInset là một tập hợp các padding cho scrollView
// Khi ta set tập hợp này cách đều các phía là 30 thì có nghĩa là vùng sử dụng của chúng ta hiện tại không phải là 800 x 800
// Mà là: 800 height - (10top + 30bottom) = 760 height và 800 width - (20left + 40right) = 740 width
scrollViewSample.contentInset = UIEdgeInsets(top: 10, left: 20, bottom: 30, right: 40)

print("\nIN \t\t \ \(scrollViewSample.contentInset)")
print("OFF \t \ \(scrollViewSample.contentOffset)")
print("SIZE \t \ \(scrollViewSample.contentSize)");
```

```
loginBtn.addTarget(self, action: #selector(tap), for: .touchUpInside)
```

▼ Các thuộc tính trong **View**

▼ frame: CGRect

Vị trí của một view so với view **CHA** đang chứa nó.

▼ bounds: CGRect

Vị trí của một view so với **CHÍNH NÓ** ở vị trí gốc ban đầu của nó

▼ center: CGPoint

Toạ độ trung tâm của một view.

▼ transform: CGAffineTransform (thuộc tính khá thú vị) (**thí nghiệm vui**)

Thuộc tính cho phép xoay đối tượng, phóng to, thu nhỏ và dịch chuyển.

CGAffineTransform là một ma trận 2 cột, 3 dòng có thể nhân với một vector 2D (CGPoint) để thay đổi giá trị của nó.

▼ Core Graphics cung cấp các hàm cho phép biến đổi đối tượng

```
//  
CGAffineTransform(a: CGFloat, b: CGFloat, c: CGFloat, d: CGFloat, tx: CGFloat, ty: CGFloat)  
CGAffineTransform(from: Decoder) throws  
// Phép xoay  
CGAffineTransform(rotationAngle: CGFloat)  
// Phép phóng to, thu nhỏ  
CGAffineTransform(scaleX: CGFloat, y: CGFloat)  
// Phép dịch chuyển  
CGAffineTransform(translationX: CGFloat, y: CGFloat)
```

▼ clipsToBounds: Bool

Thuộc tính chỉ định cho việc ta có được nhìn thấy phần bị cắt đi sau khi View nó dịch chuyển khỏi vị trí gốc (dịch chuyển khỏi boundary). Phần dư ra khỏi bound sẽ không nhìn thấy nếu thuộc tính này được set là **true**.

▼ backgroundColor: UIColor

Thuộc tính dùng để set **màu nền** cho View

▼ alpha: CGFloat

Nhận giá trị từ **0 - 1** là kênh mô tả **độ trong suốt** của View. Nhận **0 trong suốt hoàn toàn**.

▼ isOpaque: Bool

Để xác định view có bị **đục, mờ** hay không? **true là có**.

▼ isHidden: Bool

Đánh dấu hoặc thông tin xem **view có đang bị ẩn** hay không? **true là có bị ẩn**.

▼ contentMode: UIView.ContentMode

Đây là các thuộc tính dùng để chỉnh sửa **vị trí của nội dung** hiện tại như: **scaleToFill, scaleAspectFit, center, left...**

▼ mask: UIView (thuộc tính khá thú vị)

Thuộc tính cho phép ta cắt một phần tùy ý trên view hiện tại. Lưu ý rằng, view bạn khởi tại ở đây sẽ làm nhiệm vụ là **cắt 1 khoảng bằng với mặt nạ mình khởi tạo** chứ không phải là add thêm vào view một mặt nạ.

▼ tint color: UIColor

Thuộc tính màu mặc định đối với các thuộc tính view được cài đặt mà không thiết lập màu cho nó (**màu này không phải màu background**) ví dụ như màu của text trong button chẳng hạn.

▼ Các thuộc tính trong **Layer** (**tham khảo**)

▼ position: CGPoint

Là vị trí của **anchorPoint** so với **view cha**

▼ anchorPoint: CGPoint

```
// AnchorPoint là điểm để xác định tâm vật thể để thực hiện các thao tác transform như xoay chẳng hạn.
// Mặc định AnchorPoint sẽ nằm ở tâm của vật thể.
// Plugin để thay đổi vị trí anchor point.
// box.setAnchorPoint(CGPoint(x: 0, y: 0))
func setAnchorPoint(_ point: CGPoint) {
    var newPoint = CGPoint(x: bounds.size.width * point.x, y: bounds.size.height * point.y)
    var oldPoint = CGPoint(x: bounds.size.width * layer.anchorPoint.x, y: bounds.size.height * layer.anchorPoint.y);

    newPoint = newPoint.applying(transform)
    oldPoint = oldPoint.applying(transform)

    var position = layer.position

    position.x -= oldPoint.x
    position.x += newPoint.x

    position.y -= oldPoint.y
    position.y += newPoint.y

    layer.position = position
    layer.anchorPoint = point
}
```

▼ transform: CATransform3D

Thực hiện các phép biến đổi 3D

▼ setAffineTransform(_ m: CGAffineTransform)

Thực hiện các phép biến đổi 2D như xoay, phóng to, thu nhỏ, di chuyển view

▼ borderColor: CGColor?

Màu viền

▼ borderWidth

Độ dày đường viền

▼ contents

```
// Dùng để đưa ảnh vào layer
sampleView.layer.contents = UIImage(named: "summericons_100px_03")?.cgImage
```

▼ opacity: Float

Độ mờ, giá trị càng thấp view càng mờ. Giá trị từ 0 đến 1

▼ animationKeys

```
// Phương thức get → Lấy ra danh sách các animation key đang sử dụng
// Ngoài cách gọi trực tiếp keypath ta còn có thể thực hiện lấy thông qua cú pháp
// Không cần nhớ transform.scale
// CABasicAnimation(keyPath: "transform.scale")
// được viết lại như bên dưới
// Ưu điểm: Không cần nhớ key "transform.scale"
// Nhược điểm: dài dòng, code bị chia ra làm nhiều dòng hơn

sampleView.layer.contents = UIImage(named: "summericons_100px_03")?.cgImage
let fadeOut = CABasicAnimation(keyPath: #keyPath(CALayer.opacity))
fadeOut.fromValue = 1
fadeOut.toValue = 0
fadeOut.duration = 5

let expandScale = CABasicAnimation(keyPath: #keyPath(CALayer.transform))
expandScale.valueFunction = CAValueFunction(name: .scale)
expandScale.fromValue = [1, 1, 1]
expandScale.toValue = [3, 3, 3]

// Ngoài ra còn có sử dụng CAAnimationGroup để nhóm nhiều layer lại
// Các layer trong groupd được thực hiện 1 cách đồng thời
let fadeAndScale = CAAnimationGroup()
fadeAndScale.animations = [fadeOut, expandScale]
```

```
fadeAndScale.duration = 1

sampleView.layer.add(fadeOut, forKey: "1key")
sampleView.layer.add(expandScale, forKey: "2key")
sampleView.layer.add(fadeAndScale, forKey: nil)
// keys animation: ["1key", "2key"]
print("keys animation: \(sampleView.layer.animationKeys() ?? [])")
```

▼ Các kỹ thuật vẽ

▼ vẽ line chạy (CAShapeLayer, path, strokeColor, keyPath: "lineDashPhase")

```
// Code để tạo ra một hình các nét đứt (line dash)
let layer = CAShapeLayer()
let bounds = CGRect(x: 50, y: 50, width: 250, height: 250)
// Vẽ đường viền cho layer (Nếu ko có đường viền sẽ ko thể fill màu được)
// Độ cong của cạnh phụ thuộc vào giá trị cornerRadii -> width (thử đổi sẽ thấy)
layer.path = UIBezierPath(roundedRect: bounds, byRoundingCorners: .allCorners, cornerRadii: CGSize(width: 20, height: 20))
// Màu line
layer.strokeColor = UIColor.black.cgColor
// Màu nền bằng rỗng
layer.fillColor = nil
layer.lineDashPattern = [8, 6]
view.layer.addSublayer(layer)

// Sử dụng CABasicAnimation
let animation = CABasicAnimation(keyPath: "lineDashPhase")
animation.fromValue = 0
// With the line dash pattern used above - 8, 6 - that will result in toValue being set to 14.
animation.toValue = layer.lineDashPattern?.reduce(0) { $0 - $1.intValue } ?? 0
// Thời gian thực hiện animation
animation.duration = 1
// Tốc độ animation
animation.speed = 10
// Số lần lặp animation (giống như repeat)
animation.repeatCount = .infinity
layer.add(animation, forKey: "line")
```

▼ CAKeyframeAnimation (keyPath, keyTimes, values, duration, repeatCount)

```
// Tạo view di chuyển lên xuống (sử dụng position.y)
let animation = CAKeyframeAnimation()
animation.keyPath = "position.y"
// Các thời điểm animation
animation.keyTimes = [0, 0.5, 1]
// Các giá trị tương ứng cho các thời điểm
animation.values = [0, 300, 0]
// Thời gian thực hiện animation
animation.duration = 2
animation.isAdditive = true
// Số lần lặp
// .infinity: vô tận, vô cùng
animation.repeatCount = .infinity
view.layer.add(animation, forKey: "move")

// Tạo hiệu ứng rung lắc (shake) (sử dụng position.x)
let animation1 = CAKeyframeAnimation()
animation1.keyPath = "position.x"
animation1.keyTimes = [0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1]
animation1.values = [0, 10, -10, 10, -5, 5, -5, 0]
animation1.duration = 0.4
animation1.isAdditive = true
animation1.repeatCount = .infinity
view.layer.add(animation1, forKey: "shake")
```

▼ CAGradientLayer (frame, colors)


```
let layer = CAGradientLayer()
layer.frame = CGRect(x: 64, y: 64, width: 160, height: 160)
// Dây màu cần dần trải
layer.colors = [UIColor.red.cgColor, UIColor.black.cgColor]
view.layer.addSublayer(layer)
```

▼ CAEmitterLayer

Tạo các số lượng lớn các đối tượng

How to emit particles using CAEmitterLayer

Swift version: 5.4 Paul Hudson @twostraws Believe it or not, iOS has a built-in particle system that works great in all UIKit apps and is immensely customizable. To get started you need to create a CAEmitterLayer object and tell it how to create particles: where it should

 <https://www.hackingwithswift.com/example-code/calayer/how-to-emit-particles-using-caemitterlayer>



▼ vẽ progress với strokeEnd (autoreverses, keyPath: "strokeEnd")

```
let animation = CABasicAnimation(keyPath: "strokeEnd")
// Bắt đầu vẽ từ giá trị 0
animation.fromValue = 0
// Kết thúc là một nửa khung hình. Giá trị từ 0 đến 1 (1 là toàn bộ khung hình)
animation.toValue = 0.5
// Thời gian vẽ là 2s
animation.duration = 2
// Đi đến và đi ngược lại
animation.autoreverses = true
animation.repeatCount = .infinity
layer.add(animation, forKey: "line")
```

▼ Các kỹ thuật đổ bóng (shadowOffset, shadowColor, shadowRadius, shadowOpacity, shadowPath)

```
// Xem thêm các kỹ thuật tại: https://ichi.pro/vi/cach-tao-bong-nang-cao-trong-swift-huong-dan-danh-cho-ios-swift-172
let vw = UIView(frame: CGRect(x: 100, y: 100, width: 128, height: 128))
vw.backgroundColor = .white
// Độ lệch của bóng
// Nếu ta để .zero thì bóng sẽ toả đều
// Nếu ta để như bên dưới bóng bị lệch (+30, +30) tức là (lệch xuống phía dưới bên phải)
vw.layer.shadowOffset = CGSize(width: 30, height: 30)
vw.layer.shadowColor = UIColor.red.cgColor
// Bán kính mà bóng có thể phủ (Giống như đèn pin vậy càng hội tụ càng sáng)
// Nếu càng xa thì bóng càng mờ do lan toả xa
// Nếu càng gần thì bóng sẽ rõ
vw.layer.shadowRadius = 50
// Tùy chỉnh độ mờ của bóng, giữa 0 và 1.
// Giá trị của 0 nghĩa là bóng sẽ không được nhìn thấy.
vw.layer.shadowOpacity = 1

vw.layer.shadowPath = UIBezierPath(rect: vw.bounds).cgPath
view.addSubview(vw)
```

▼ cornerRadius và maskedCorners

```
let redBox = UIView(frame: CGRect(x: 100, y: 100, width: 128, height: 128))
redBox.backgroundColor = .red
// Bo tròn một góc 25 độ
redBox.layer.cornerRadius = 25
// Chỉ bo tròn top-left(layerMinXMinYCorner) và bottom-right(layerMaxXMaxYCorner)
redBox.layer.maskedCorners = [.layerMinXMinYCorner, .layerMaxXMaxYCorner]
view.addSubview(redBox)
```

▼ Các phương pháp thực hiện biến đổi trên View

▼ animate(withDuration:delay:usingSpringWithDamping:initialSpringVelocity:options:animations:completion:)



Được dùng khi bạn muốn thực hiện các thao tác animation lên view.

▼ duration

Thời gian để một view chuyển từ trạng thái cũ sang trạng thái mới (Tính bằng giây)

▼ delay

Thời gian chờ, sau thời gian này thì mới bắt đầu thực hiện việc animation (Tính bằng giây)

▼ spring

Mô tả chuyển động co giãn như chuyển động của con lắc lò xo. Vật được chuyển động tới vị trí mới và lắc nhẹ trước khi đặt vào vị trí đích.

▼ damping

Hệ số dao động có giá trị từ **0 đến 1**. (Nếu giá trị càng thấp thì vật sẽ dao động càng nhanh và ngược lại).

▼ velocity

Vận tốc của vật chuyển động.

▼ options (Các chế độ animation trong *UIView.AnimationOptions*)

▼ nhóm **Curve** ([Tham khảo](#))

▼ *curveEaseInOut* (thuộc tính khả thú vị)

Tăng tốc lúc đầu và giảm vào lúc cuối

▼ curveEaseIn

Di chuyển **nhẹ dần đều**

▼ curveEaseOut

Di chuyển **chậm dần đều**

▼ curveLinear

Di chuyển đều đặn không nhanh, không chậm

▼ nhóm **Setting khác**

▼ `allowUserInteraction`

Cho phép tương tác lúc đang thực hiện animation.

▼ `beginFromCurrentState`

Có vẻ như là thực hiện animation ngay cả với trạng thái view đang animation còn dang dở (*cần kiểm nghiệm lại*).

▼ `repeat`

Dùng để lặp lại animation **vô hạn**.

▼ `autoreverse`

Hiệu ứng view tới điểm kết thúc sau đó **di chuyển dần trở lại** điểm xuất phát.

▼ `overrideInheritedDuration`

▼ `overrideInheritedCurve`

▼ `allowAnimatedContent`

▼ `showHideTransitionViews`

▼ `overrideInheritedOptions`

▼ animations

Khối định nghĩa dữ liệu ta muốn thay đổi so với trạng thái cũ.

▼ completion

Hành động sẽ thực hiện khi kết thúc animation.

▼ *UIView.transition(from:to:duration:options:completion:)*



Được dùng khi bạn muốn thêm một view vào view hierarchy (hệ thống phân cấp chế độ xem) hoặc xóa một view ra khỏi view hierarchy.

▼ from

View nguồn trong để bắt đầu chuyển đổi.

▼ to

View cuối cùng trong quá trình chuyển đổi.

▼ duration

Thời gian để một view chuyển từ trạng thái cũ sang trạng thái mới (Tính bằng giây)

▼ options (Các chế độ animation trong *UIView.AnimationOptions*)

▼ nhóm **Transition** (Tham khảo)

▼ transitionFlipFromLeft

Hiệu ứng **lật trang từ bên trái**

▼ transitionFlipFromRight

Hiệu ứng **lật trang từ bên phải**

▼ transitionCurlUp

Hiệu ứng xé lịch từ dưới lên

▼ transitionCurlDown

Hiệu ứng dán lại lịch từ trên xuống

▼ **transitionCrossDissolve** (*Hiệu ứng hay*)

Hiệu ứng chuyển hoạt cảnh từ phân đoạn này sang phân đoạn mới.

▼ transitionFlipFromTop

Lật view **theo chiều kim đồng hồ** (đè đầu vào trong, lật đít lên)

▼ transitionFlipFromBottom

Lật view **ngược chiều kim đồng hồ** (đè đít vào trong, lộn đầu xuống)

▼ completion

Hành động sẽ thực hiện khi kết thúc animation.

▼ **UIView.animateKeyframes(withDuration:delay:options:animations:completion:)**



Phương thức định nghĩa một nhóm các giai đoạn animation khác nhau vào thành 1 nhóm. Ngoài ra, ta có thể dùng chung một số thuộc tính mà vẫn kiểm soát các giai đoạn này 1 cách riêng biệt.

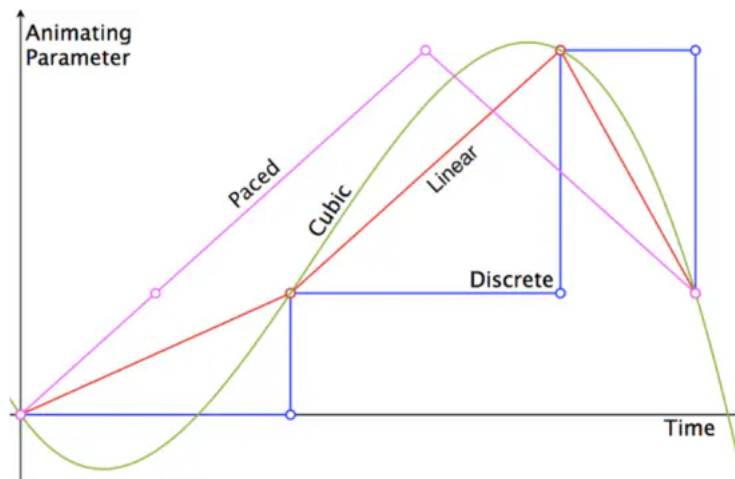
▼ duration

Thời gian để một view chuyển từ trạng thái cũ sang trạng thái mới (Tính bằng giây)

▼ delay

Thời gian chờ, sau thời gian này thì mới bắt đầu thực hiện việc animation (Tính bằng giây)

▼ options (Các chế độ animation của keyframe *UIView.KeyframeAnimationOptions*)



<https://codepen.io/AmeliaBR/pen/EzAju>

▼ calculationModeLinear

Thực hiện các hành động **tuyến tính** đến các mốc thời gian.

▼ calculationModeDiscrete

Thực hiện cách hành động một cách **rời rạc**, chỉ đúng giờ mới thực hiện biến đổi

▼ calculationModePaced

Thực hiện hành động **không tuyến tính theo thời gian**, không phụ thuộc thời gian thiết đặt, thời gian giữa 2 lần chạy **phụ thuộc vào giá trị, tốc độ đã set trước đó**.

▼ calculationModeCubic

Thực hiện **uốn chuyển**, có sự **làm mượt mạnh** giữa những **mối ghép nối**.

▼ animations

Khối định nghĩa dữ liệu ta muốn thay đổi so với trạng thái cũ.

▼ completion

Hành động sẽ thực hiện khi kết thúc quá trình các animation trong toàn bộ frame.

▼ UIView.addKeyframe(withRelativeStartTime:relativeDuration:animations:)

Phương thức giúp ta thêm vào danh sách các keyframe các giai đoạn animation khác nhau.

▼ withRelativeStartTime

Khoảng thời gian để start animation, parameter này có giá trị từ **0 đến 1**. Trong đó 0 đại diện cho sự bắt đầu của list các giai đoạn animation, còn 1 đại diện cho sự kết thúc của các list giai đoạn animation.

(Ví dụ nếu vật có thời gian hiệu ứng động là 3s mà value của withRelativeStartTime là 0.4 thì có nghĩa tại phút thứ 1,2s thì animation của method UIView.addKeyframe sẽ bắt đầu.)

▼ relativeDuration

Khoảng thời gian show animation của 1 giai đoạn trong 1 list các giai đoạn animation. parameter nhận giá trị từ **0 đến 1**.

(Ví dụ Nếu duration của list giai đoạn animation là 2 và relativeDuration là 0.5 thì khoảng thời gian show animation của giai đoạn này là 1s.)

▼ animations

Khối định nghĩa dữ liệu ta muốn thay đổi so với trạng thái cũ.

