GeekBrains

Пользовательский интерфейс iOS-приложений

Анимации. Часть II

Keyframe-анимации. Группы анимаций. Кривые Безье и их анимирование. 3D-анимации. UIViewPropertyAnimator.

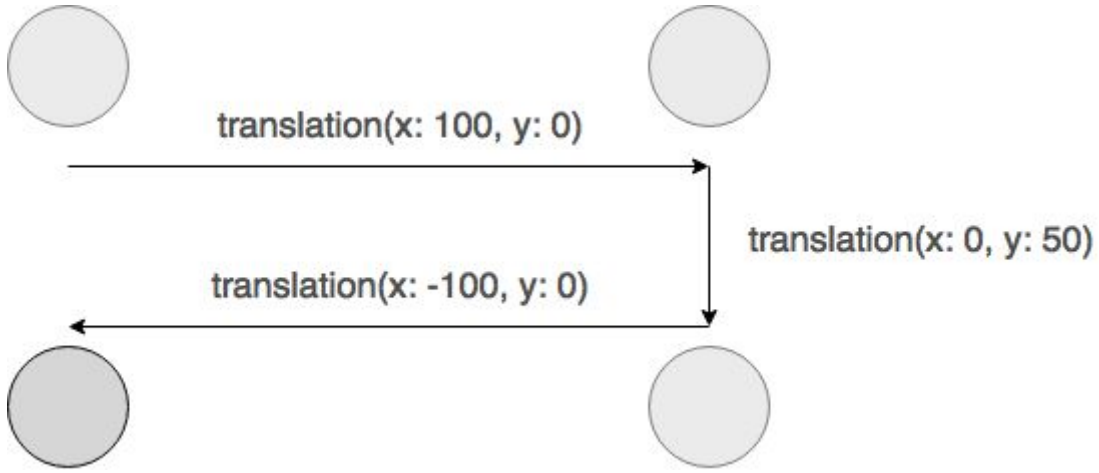
Оглавление

Keyframe-анимации

Чаще всего приходится делать анимации, которые состоят из одного действия: передвинуть **view**, повернуть. Но бывают и комплексные анимации. На предыдущем уроке мы рассматривали пример двух анимаций, следующих друг за другом. Такой подход не всегда удобен — представьте, что надо сделать пять последовательных анимаций.

В таких случаях применяются keyframe-анимации. Они состоят из контрольных точек (**keyframe**) с информацией об анимациях, которые будут в них.

Схематически keyframe-анимации можно изобразить так:



Рассмотрим код такой анимации, чтобы продемонстрировать преимущества **keyframe**:

```
UIView.animate(withDuration: 0.5,
  animations: {
    view.center.x += 200.0
  },
  completion: { _ in
    UIView.animate(withDuration: 0.5,
      animations: {
        view.center.y += 100.0
      },
      completion: { _ in
        UIView.animate(withDuration: 0.5,
          animations: {
            view.center.x -= 200.0
          },
          completion: { _ in
            UIView.animate(withDuration: 0.5,
              animations: {
                view.center.y -= 100.0
              }
            )
          }
        )
      }
    )
  }
)
```

Такой код тяжело читать и изменять.

Создание keyframe-анимации состоит из двух этапов: вызова метода **UIView.animateKeyframes** и добавления самих **keyframes**. Метод выглядит так:

```
UIView.animateKeyframes(withDuration: 2,
                        delay: 0,
                        options: [],
                        animations: {
                            // Здесь будут keyframes
                        },
                        completion: nil)
```

Он похож на обычный метод анимации, но в блоке **animations** будут находиться **keyframes**.

Чтобы добавить **keyframe**, нужно вызвать метод **UIView.addKeyframe**:

```
UIView.addKeyframe(withRelativeStartTime: 0,
                   relativeDuration: 0.25,
                   animations: {
                       view.center.x += 200
                   })
```

В качестве первого параметра передается относительное время начала анимации. Если хотим начать анимацию с середины, нужно задать в этом параметре значение 0.5. **RelativeDuration** — это относительное время анимации. В нашем случае оно равно 0.5 секунды. В блоке **animations** изменяются свойства **view** — по аналогии с обычными анимациями.

В итоге код будет выглядеть так:

```
UIView.animateKeyframes(withDuration: 2,
                        delay: 0,
                        options: [],
                        animations: {
                            UIView.addKeyframe(withRelativeStartTime: 0,
                                               relativeDuration: 0.25,
                                               animations: {
                                                   view.center.x += 200
                                               })
                            UIView.addKeyframe(withRelativeStartTime: 0.25,
                                               relativeDuration: 0.25,
                                               animations: {
                                                   view.center.y += 100
                                               })
                            UIView.addKeyframe(withRelativeStartTime: 0.5,
                                               relativeDuration: 0.25,
                                               animations: {
                                                   view.center.x -= 200
                                               })
                            UIView.addKeyframe(withRelativeStartTime: 0.75,
                                               relativeDuration: 0.25,
                                               animations: {
                                                   view.center.x -= 100
                                               })
                        },
                        completion: nil)
```

Теперь можем легко менять последовательность анимаций и добавлять новые **keyframes**.

Группы анимаций

Несколько анимаций можно объединить в группу. В этом случае они будут выполняться синхронно. Это полезно, если необходимо одновременно переместить и повернуть **view**, например.

Для **UIView** группировка анимаций работает во всех методах **UIView.animate**, но в случае со слоем добавление анимаций будет ставить их в очередь.

Чтобы сгруппировать несколько анимаций слоя, используется класс **CAAnimationGroup**. Он содержит информацию о времени анимаций, задержке и остальных параметрах.

Пример создания группы анимаций:

```
let animationsGroup = CAAnimationGroup()
animationsGroup.duration = 0.5
animationsGroup.fillMode = CAMediaTimingFillMode.backwards
```

Теперь нужно создать анимации и добавить их в группу:

```
let translation = CABasicAnimation(keyPath: "position.x")
translation.toValue = 100
let alpha = CABasicAnimation(keyPath: "opacity")
translation.toValue = 0

animationsGroup.animations = [translation, alpha]
```

В случае с группой не нужно задавать время анимации и задержку для каждой ее части — только для всей группы.

Когда группа анимаций создана и настроена, ее можно добавить на слой:

```
layer.add(animationsGroup, forKey: nil)
```

Дополнительные опции анимации слоя

Временные функции

Рассмотрим применение уже знакомых нам временных функций — **linear**, **curveEaseIn** и других — к анимациям слоя.

За временную функцию отвечает свойство **CABasicAnimation.timingFunction**. Оно имеет тип **CAMediaTimingFunction**, у которого есть два инициализатора:

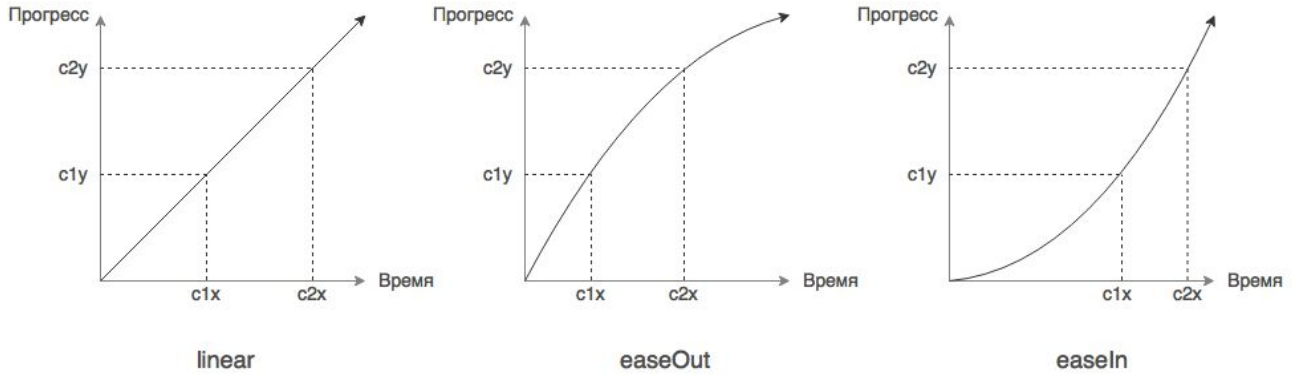
- **init(name: String)** — инициализирует временную функцию с заданным именем;
- **init(controlPoints c1x: Float, c1y: Float, c2x: Float, c2y: Float)** — инициализирует временную функцию с заданными контрольными точками. Такая функция называется кубической кривой Безье.

В качестве имени временной функции в первый инициализатор можно передать следующие значения:

- **CAMediaTimingFunctionName.linear**;
- **CAMediaTimingFunctionName.easeIn**;
- **CAMediaTimingFunctionName.easeOut**;
- **CAMediaTimingFunctionName.easeInEaseOut**;
- **CAMediaTimingFunctionName.default**.

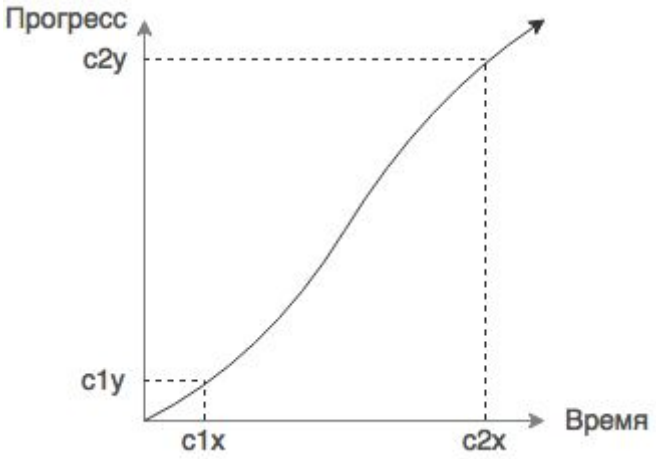
Последнее значение — это временная функция, которая используется системой для большинства анимаций.

Второй инициализатор позволяет создать собственную временную функцию на основе переданных точек. Чтобы понять принцип, рассмотрим графики стандартных функций:



На графиках видно, что контрольные точки (параметры **c1x**, **c1y**, ...) — это те места, где анимация изменяет скорость. Разберем пример временной функции и ее график:

```
let timingFunction = CAMediaTimingFunction(controlPoints: 0.15, 0.1, 0.85, 0.9)
```



Наглядно потренироваться в создании своих временных кривых можно на сайте: https://cubic-bezier.com/.

Повторение

Как и в случае с методом **UIView.animate**, анимацию слоя можно зациклить. Для этого нужно установить количество повторений с помощью свойства **repeatCount** — любое положительное число типа **Float**. Чтобы создать бесконечную анимацию, следует задать в этом свойстве значение **Float.infinity**.

Чтобы проигрывать анимацию в обратном направлении, надо установить свойство **autoreverses**, равное **true**.

Анимирование путей Безье

Анимация обводки

В четвертом уроке мы рассматривали класс **UIBezierPath** и рисовали с его помощью звезду. Чтобы анимировать эту звезду, как будто ее рисуют, понадобится создать **CAShapeLayer**, установить **path** звезды в качестве **path** слоя и создать анимации для свойств **strokeStart** и **strokeEnd** (начало и конец рисования линии). Они относительны и могут находиться в промежутке от 0 до 1.

Если для линии длиной в 100 точек установить **strokeStart** равным 0.25, а **strokeEnd** — 0.75, то линия нарисуется с 25 по 75 точку. Чтобы анимировать **path**, будто он отрисовывается от начала до конца, надо установить **strokeEnd** в «0» перед началом анимации, а в ней самой задать его значение равным единице.

Код анимации будет выглядеть так:

```
let pathAnimation = CABasicAnimation(keyPath: "strokeEnd")
pathAnimation.fromValue = 0
pathAnimation.toValue = 1
```

После этого можно добавить анимацию на слой и увидеть результат.

Можно комбинировать анимации для свойств **strokeStart** и **strokeEnd**. Например, сделать так, чтобы рисовался только отрезок из контура. Для этого нужно создать две анимации и добавить их в группу:

```
let strokeStartAnimation = CABasicAnimation(keyPath: "strokeStart")
strokeStartAnimation.fromValue = 0
strokeStartAnimation.toValue = 1

let strokeEndAnimation = CABasicAnimation(keyPath: "strokeEnd")
strokeEndAnimation.fromValue = 0
strokeEndAnimation.toValue = 1.2

let animationGroup = CAAnimationGroup()
animationGroup.duration = 2
animationGroup.animations = [strokeStartAnimation, strokeEndAnimation]
```

В результате увидим, как линия движется от начала до конца **path**. Можно установить повтор, чтобы анимация выполнялась несколько раз:

Анимация движения вдоль пути

Чтобы анимировать движение слоя вдоль **UIBezierPath**, можно использовать keyframe-анимации. У класса **CAKeyframeAnimation** есть свойство **path**, после установки которого контрольные точки добавятся автоматически. Также в комбинации с **path** можно установить свойство **calculationMode**. Оно определяет, как будет сгенерирована анимация между контрольными точками, и может принимать следующие значения:

- **CAAnimationCalculationMode.linear** — значение по умолчанию. Движение вдоль пути будет выполняться в соответствии с заданным временем;
- **CAAnimationCalculationMode.discrete** — при установке этого значения анимируемый объект не будет двигаться вдоль пути, а будет «прыгать» по контрольным точкам;
- **CAAnimationCalculationMode.paced** — это значение генерирует оптимальное время анимации каждой контрольной точки, что делает всю анимацию плавной;
- **CAAnimationCalculationMode.cubic** — задает плавный путь, все «повороты» будут сглажены;
- **CAAnimationCalculationMode.cubicPaced** — комбинация двух предыдущих.

В нашем случае будем использовать значение **CAAnimationCalculationMode.paced**, чтобы время между контрольными точками рассчиталось автоматически, а путь не изменился.

Создадим слой, который будем анимировать:

```
let circleLayer = CAShapeLayer()
circleLayer.backgroundColor = UIColor.red.cgColor
circleLayer.bounds = CGRect(x: 0, y: 0, width: 20, height: 20)
circleLayer.position = CGPoint(x: 40, y: 20)
```

Теперь можно создать анимацию и добавить ее на этот слой:

```
let followPathAnimation = CAKeyframeAnimation(keyPath: "position")
followPathAnimation.path = path.cgPath
followPathAnimation.calculationMode = CAAnimationCalculationMode.paced
circleLayer.add(followPathAnimation, forKey: nil)
```

Его, в свою очередь, можно добавить на слой со звездой, чтобы совместить движение круга по обводке звезды. В результате получим такую анимацию:

3D-анимации

3D-анимации создаются с помощью трехмерной трансформации, с методами которой — **CATransform3D** — мы уже знакомы.

В качестве примера возьмем анимацию разворота **view** вокруг оси **y** и его увеличение.

Создадим трансформацию поворота. Изначально **view** будет трансформированным, а анимироваться будет к начальной трансформации — **CATransform.identity**.

Начальная трансформация будет выглядеть так:

```
func setInitialViewTransform() {
    let rotation = CATransform3DMakeRotation(.pi, 0, 1, 0)
    let scale = CATransform3DScale(CATransform3DIdentity, 0.8, 0.8, 0)
    let transform = CATransform3DConcat(rotation, scale)
    self.testView.transform = CATransform3DGetAffineTransform(transform)
}
```

Первая трансформация — это разворот, вторая — уменьшение. Далее эти анимации сливаются в одну с помощью метода **CATransform3DConcat** и устанавливаются для **view**.

Напишем функцию анимации. Сделаем так, чтобы **view** можно было показать, а затем скрыть. Для этого заведем переменную, обозначающую текущее состояние **view**. Код анимации:

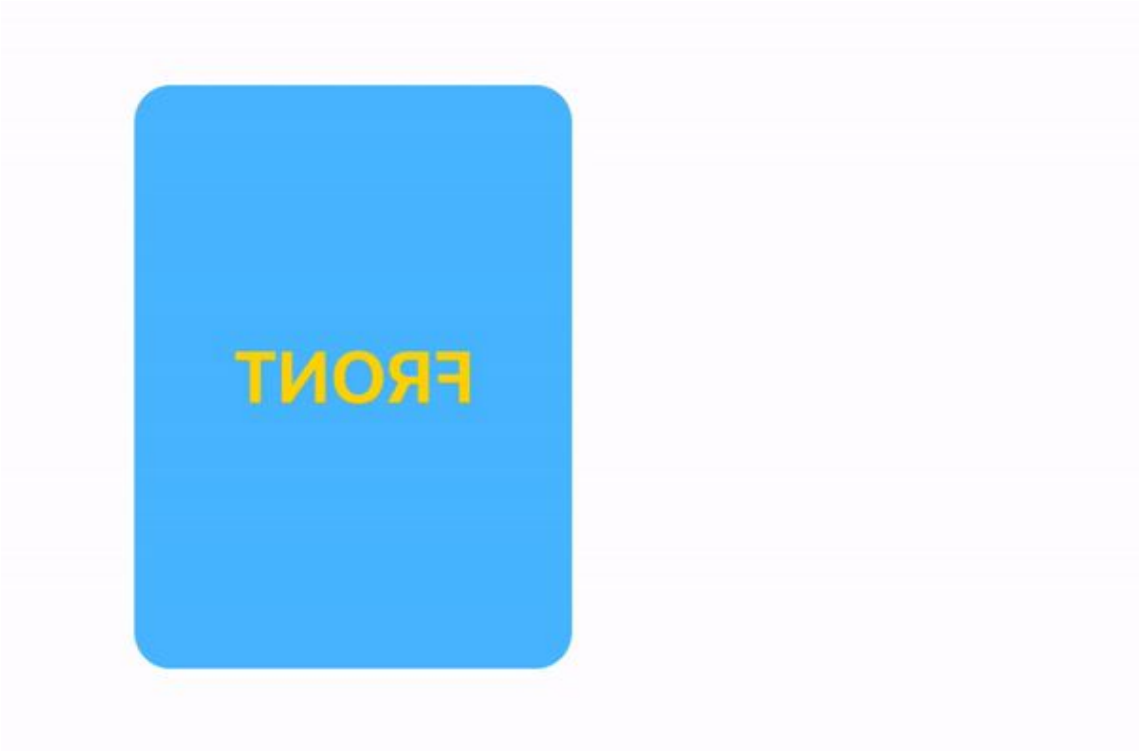
```
@IBAction func toggleViewVisibility() {
    UIView.animate(withDuration: 1, animations: {
        if self.viewIsShown {
            self.setInitialViewTransform()
        } else {
            self.testView.transform = .identity
        }
    }, completion: { _ in
        self.viewIsShown = !self.viewIsShown
    })
}
```

Создали простую анимацию, которая в зависимости от свойства **viewIsShown** показывает или скрывает **view**. Результат:

Изменим анимацию так, чтобы разворот выполнялся относительно левой границы.

Для этого рассмотрим **anchorPoint** — свойство класса **CALayer**. Это точка, относительно которой происходит трансформация слоя. По умолчанию она находится в середине слоя и имеет значение (0.5, 0.5). Но для нашей анимации нужно установить это свойство равным (0, 0.5) — чтобы анимация поворота происходила относительно левого края слоя.

Установим это свойство во **viewDidLoad** и посмотрим на результат:



UIViewPropertyAnimator

Обзор класса UIViewPropertyAnimator

В iOS 10 появился класс **UIViewPropertyAnimator**. Он позволяет управлять воспроизведением анимации — запускать и ставить на паузу, получать ее текущее состояние, добавлять анимации динамически и легко создавать их группы. Важная особенность — можно управлять прогрессом анимаций и делать их интерактивными.

Чтобы создать анимацию, нужно воспользоваться одним из конструкторов **UIViewPropertyAnimator**:

- **UIViewPropertyAnimator(duration: CGFloat, curve: UIViewAnimationCurve, animations: (() -> Void)?)**;
- **UIViewPropertyAnimator(duration: CGFloat, dampingRatio: CGFloat, animations: (() -> Void)?)**.

Первый конструктор создает обычную анимацию с заданной временной функцией. Второй — пружинную с заданным коэффициентом затухания.

Когда аниматор создан, его можно запустить. Для этого есть две функции:

- **startAnimation()**;
- **startAnimation(afterDelay: TimeInterval)**.

Первый метод просто начинает анимацию, а второй запускает ее с задержкой.

Для остановки анимации используются следующие методы:

- **pauseAnimation()**;

- **stopAnimation(withoutFinishing: Bool)**.

Первый метод ставит анимацию на паузу — ее можно продолжить с места остановки. Второй останавливает без такой возможности. Параметр **withoutFinishing** влияет на то, как закончится анимация — если значение параметра — **true**, блок окончания анимации не будет выполнен, и придется вручную вызывать метод завершения анимации **finishAnimation(position: UIViewAnimatingPosition)**. Он полностью завершает анимацию в переданной позиции (**start**, **current** или **end**).

Продолжить анимацию можно с помощью метода **continueAnimation(withTimingParameters: UITimingCurveProvider?, durationFactor: CGFloat)**. Первый параметр предоставляет информацию о том, каким должен быть переход от предыдущих анимаций к следующим. Если анимации не менялись после паузы, нужно передать **nil**. Второй параметр нужен для установки нового времени анимации. Это множитель, который применяется к изначальному времени. Если длительность анимации менять не нужно, можно передать 0.

Разберем использование **UIViewPropertyAnimator** на примере с движением **view** вверх на 100 точек:

```
let animator = UIViewPropertyAnimator(duration: 0.5, curve: .easeInOut) {
    self.view.frame = self.view.frame.offsetBy(dx: 0, dy: 100)
}
animator.startAnimation()
```

Сделаем пружинную анимацию:

```
let animator = UIViewPropertyAnimator(duration: 0.5, dampingRatio: 0.5) {
    self.view.frame = self.view.frame.offsetBy(dx: 0, dy: 100)
}
animator.startAnimation()
```

Создание интерактивной анимации

У класса **UIViewPropertyAnimator** есть свойство **fractionComplete**, которое отражает прогресс анимации. У него может быть значение от 0 до 1, где 0 — это начальное состояние анимации, а 1 — конечное. Можно использовать это свойство, чтобы сделать интерактивную анимацию.

Сначала добавим **UIPanGestureRecognizer** в тот **view**, в котором находится анимируемый **view**:

```
let recognizer = UIPanGestureRecognizer(target: self, action:
#selector(onPan(_:)))
self.view.addGestureRecognizer(recognizer)
```

После этого реализуем метод **onPan**, в котором и будет логика работы с анимацией:

```
var interactiveAnimator: UIViewPropertyAnimator!

@objc func onPan(_ recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
    case .began:
        interactiveAnimator = UIViewPropertyAnimator(duration: 0.5,
                                                     curve: .easeInOut,
animations: {
             self.animatingView.frame = self.animatingView.frame.offsetBy(dx: 0,
dy: 100)
        })
        interactiveAnimator.pauseAnimation()
    case .changed:
        let translation = recognizer.translation(in: self.view)
        interactiveAnimator.fractionComplete = translation.y / 100
    case .ended:
                interactiveAnimator.continueAnimation(withTimingParameters: nil,
durationFactor: 0)
    default: return
    }
}
```

В этом методе обрабатываются состояния распознавателя и соответствующие действия с аниматором. Когда распознаватель находится в положении **began**, создаем новый аниматор и ставим анимацию на паузу. Когда получаем состояние **changed** — считаем перемещение в том **view**, где находится распознаватель. Устанавливаем это значение, предварительно разделив его на 100, в свойство **fractionComplete**. При получении состояния **ended** продолжаем анимацию с текущего места.

Теперь при движении пальца по экрану будет происходить интерактивная анимация.

Практика

Создание анимаций на экране авторизации

Сделаем анимации для элементов на экране авторизации, используя изученные технологии.

Начнем с анимации для надписей над полями ввода логина и пароля. Для них применим keyframe-анимацию, при которой эти надписи будут меняться местами. Изначально они будут расположены на местах друг друга.

Зададим начальную трансформацию. Чтобы поставить надписи на соседнее место, достаточно поменять их центры. Для этого — вычислить расстояние по оси **y** и сделать трансформацию:

```
let offset = abs(self.loginTitleView.frame.midY -
self.passwordTitleView.frame.midY)

self.loginTitleView.transform = CGAffineTransform(translationX: 0, y: offset)
self.passwordTitleView.transform = CGAffineTransform(translationX: 0, y:
-offset)
```

Теперь создадим keyframe-анимацию и добавим в нее ключевые кадры. Первым будет перемещение в сторону и вниз, а вторым — на исходное положение. В итоге получится такой код:

```
UIView.animateKeyframes(withDuration: 1,
                        delay: 1,
                        options: .calculationModeCubicPaced,
                        animations: {
                            UIView.addKeyframe(withRelativeStartTime: 0,
                                               relativeDuration: 0.5,
                                               animations: {
    self.loginTitleView.transform = CGAffineTransform(translationX: 150, y: 50)
    self.passwordTitleView.transform = CGAffineTransform(translationX: -150, y:
-50)
                            })
                            UIView.addKeyframe(withRelativeStartTime: 0.5,
                                               relativeDuration: 0.5,
                                               animations: {
    self.loginTitleView.transform = .identity
    self.passwordTitleView.transform = .identity
                            })
}, completion: nil)
```

В качестве **options** выбрали опцию **calculationModeCubicPaced**, так как нужно получить равномерную анимацию с плавными поворотами.

Для полей ввода будем использовать предыдущую анимацию плавного появления с одновременным увеличением. Мы создавали их на прошлом уроке, а сейчас объединим в группу:

```
let fadeInAnimation = CABasicAnimation(keyPath: "opacity")
fadeInAnimation.fromValue = 0
fadeInAnimation.toValue = 1

let scaleAnimation = CASpringAnimation(keyPath: "transform.scale")
scaleAnimation.fromValue = 0
scaleAnimation.toValue = 1
scaleAnimation.stiffness = 150
scaleAnimation.mass = 2

let animationsGroup = CAAnimationGroup()
animationsGroup.duration = 1
animationsGroup.beginTime = CACurrentMediaTime() + 1
animationsGroup.timingFunction = CAMediaTimingFunction(name:
CAMediaTimingFunctionName.easeOut)
animationsGroup.fillMode = CAMediaTimingFillMode.backwards
animationsGroup.animations = [fadeInAnimation, scaleAnimation]

self.loginView.layer.add(animationsGroup, forKey: nil)
self.passwordView.layer.add(animationsGroup, forKey: nil)
```

Анимацию для заголовка менять не будем, а сделаем с помощью **UIViewpropertyAnimator**:

```
self.titleView.transform = CGAffineTransform(translationX: 0, y:
-self.view.bounds.height / 2)

let animator = UIViewPropertyAnimator(duration: 1,
                                      dampingRatio: 0.5,
                                      animations: {
                                          self.titleView.transform = .identity
})

animator.startAnimation(afterDelay: 1)
```

Создадим интерактивную анимацию для кнопки авторизации. Сделаем так, чтобы ее можно было оттягивать вниз, а при отпускании она бы с эффектом пружины возвращалась на исходную точку.

Сначала потребуется создать **UIPanGestureRecognizer** и добавить его на основной **view** экрана:

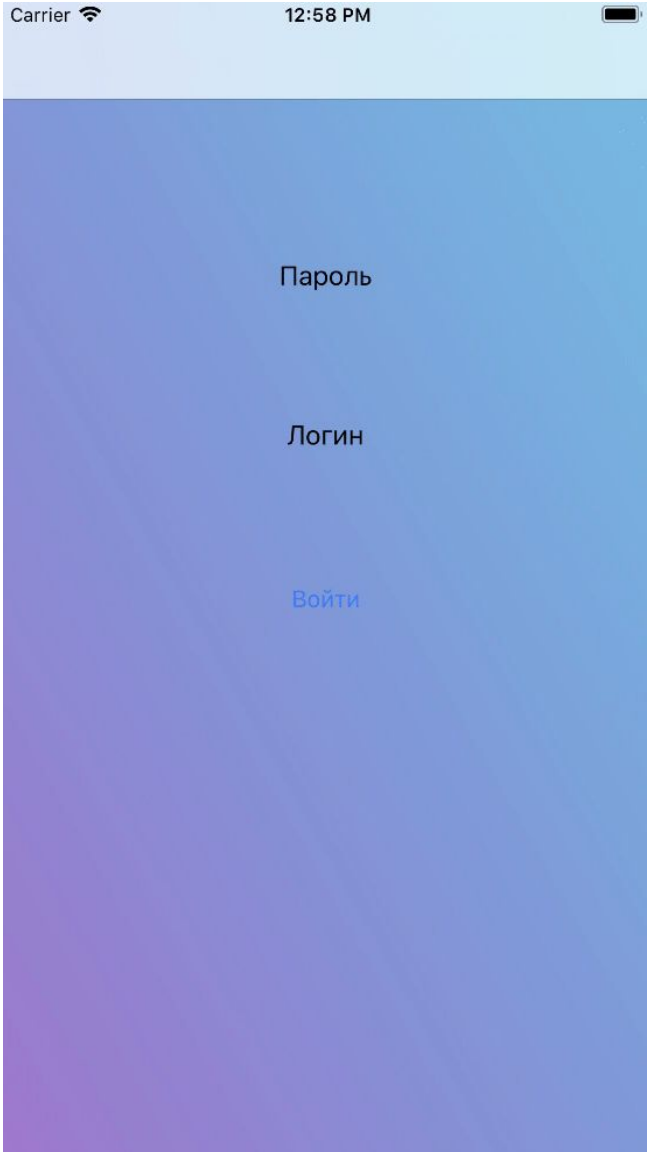
```
override func viewDidLoad() {
    // Предыдущий код

    let recognizer = UIPanGestureRecognizer(target: self, action:
#selector(onPan(_:)))
    self.view.addGestureRecognizer(recognizer)
}
```

Далее — реализовать метод **onPan** (наподобие того, что было показано в теоретической части). Исключение — обработка отпускания пальца, так как в этом месте нужно сделать так, чтобы кнопка

возвращалась в исходное положение с эффектом пружины. Для этого остановим анимацию и добавим новую, которая вернет кнопку на прежнее место. Код обработки перетягивания:

```
var interactiveAnimator: UIViewPropertyAnimator!

@objc func onPan(_ recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
    case .began:
        interactiveAnimator?.startAnimation()

        interactiveAnimator = UIViewPropertyAnimator(duration: 0.5,
                                                     dampingRatio: 0.5,
                                                     animations: {
            self.authButton.transform = CGAffineTransform(translationX: 0,
                                                          y: 150)
        })

        interactiveAnimator.pauseAnimation()
    case .changed:
        let translation = recognizer.translation(in: self.view)
        interactiveAnimator.fractionComplete = translation.y / 100
    case .ended:
        interactiveAnimator.stopAnimation(true)

        interactiveAnimator.addAnimations {
            self.authButton.transform = .identity
        }

        interactiveAnimator.startAnimation()
    default: return
    }
}
```

Когда все анимации готовы, можно запустить проект и посмотреть, что получилось:



Превращение точки в крестик

Создадим сложную анимацию — из изначальной точки, которая трансформируется в линию, а та делится на две и превращается в крестик.

Будем работать в отдельном проекте. Создадим два слоя, которые впоследствии будем трансформировать. Сразу зададим начальные свойства слоев и добавим их на слой главного **view** экрана:

```
let firstLayer = CAShapeLayer()
let secondLayer = CAShapeLayer()

firstLayer.path = UIBezierPath(rect: CGRect(x: 0, y: 0, width: 4, height: 4)).cgPath
secondLayer.path = UIBezierPath(rect: CGRect(x: 0, y: 0, width: 4, height: 4)).cgPath

firstLayer.backgroundColor = UIColor.black.cgColor
secondLayer.backgroundColor = UIColor.black.cgColor

firstLayer.frame = CGRect(x: 100, y: 100, width: 4, height: 4)
secondLayer.frame = CGRect(x: 100, y: 100, width: 4, height: 4)

firstLayer.masksToBounds = true
secondLayer.masksToBounds = true
firstLayer.cornerRadius = 2
secondLayer.cornerRadius = 2

self.view.layer.addSublayer(firstLayer)
self.view.layer.addSublayer(secondLayer)
```

Анимаций будет **3**: превращение точки в линию, поворот первой линии по часовой стрелке и второй — против часовой стрелки.

Создадим анимацию превращения точки в линию:

```
let scale = CABasicAnimation(keyPath: "bounds.size.width")
scale.byValue = 16
scale.duration = 1
scale.fillMode = CAMediaTimingFillMode.forwards
scale.isRemovedOnCompletion = false
```

Заметьте, анимируем **bounds**, а не **transform.scale** — иначе получили бы неверный вид линии. **Transform.scale** растягивает слой, а не меняет его геометрию.

Также использовали свойство **byValue**, чтобы просто поменять ширину слоя на какое-то значение, а не указывать **from-to**. Еще задали **fillMode** и **isRemovedOnCompletion**, чтобы слой остался в том состоянии, в котором завершилась анимация.

Теперь создадим анимации поворотов:

```
let rotationLeft = CABasicAnimation(keyPath: "transform.rotation")
rotationLeft.byValue = CGFloat.pi / 4
rotationLeft.duration = 1
rotationLeft.beginTime = CACurrentMediaTime() + 1
rotationLeft.fillMode = CAMediaTimingFillMode.both
rotationLeft.isRemovedOnCompletion = false

let rotationRight = CABasicAnimation(keyPath: "transform.rotation")
rotationRight.byValue = -CGFloat.pi / 4
rotationRight.duration = 1
rotationRight.beginTime = CACurrentMediaTime() + 1
rotationRight.fillMode = CAMediaTimingFillMode.both
rotationRight.isRemovedOnCompletion = false
```

Здесь тоже использовали свойства **byValue** и **fillMode**, а также установили свойство **beginTime**, чтобы вторая анимация вызывалась с небольшой задержкой.

После этого нужно добавить анимации на слои:

```
firstLayer.add(scale, forKey: nil)
firstLayer.add(rotationLeft, forKey: nil)
secondLayer.add(scale, forKey: nil)
secondLayer.add(rotationRight, forKey: nil)
```

В результате получим такую анимацию:

Практическое задание

На основе предыдущего ПЗ.

1. На экране просмотра фото добавить возможность просматривать все снимки по очереди. На всем экране, как и раньше, будет фотография, но перелистывать можно будет свайпами. Не используйте **UICollectionView/UICollectionViewController** — создайте **UIImageView** и анимируйте его/их.

2. Сделать анимацию перелистывания, которая состоит из двух частей. Сначала фотография немного отдаляется, а затем новый снимок выдвигается справа. При пролистывании назад анимация должна показываться в обратную сторону.

3. * Модифицировать анимацию перелистывания фотографий так, чтобы она была интерактивной — с возможностью начать перелистывать и отменить это действие, а также управлять прогрессом анимации. (Необязательное задание — для тех, у кого есть время.)

4. * Модифицировать индикатор загрузки. Удалить предыдущую анимацию и сделать новую — фигуру в виде облака, по контуру которого передвигается линия. Линия должна быть

фиксированной длины, с закругленными концами (Необязательное задание — для тех, у кого есть время.)

Примеры выполненных работ

1. Анимация скролла коллекции фотографий друга.
2. Анимация индикатора загрузки «облака».

Дополнительные материалы

1. Animations Explained.
2. iOS 10: новое в создании анимаций.
3. Create a Cool 3D Sidebar Menu Animation.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Animations Explained.
2. CAKeyframeAnimation.
3. UIViewPropertyAnimator.