The SwiftUI Lab

When the documentation is missing, we experiment.

Menu

Advanced SwiftUI Animations – Part 3: AnimatableModifier

September 9, 2019 by javier

We have seen how the Animatable protocol has helped us in animating paths and transform matrices. In this last part of the series, we will take it even further. The AnimatableModifier is the most powerful of all three. With it, we have no limits on what we can accomplish.

The name says it all: AnimatableModifier. It is a ViewModifier, that conforms to Animatable (your old friend from part 1). If you don't know how Animatable and animatableData work, please go to the first part of this series and check that out first.

Ok, so let's pause here and think what it means to have an animatable modifier… You probably think it is too good to be true. Can I really modify my view multiple times through an animation? The answer is simple: yes, you can.

The complete sample code for this article can be found at:
https://gist.github.com/swiftui-lab/e5901123101ffad6d39020cc7a810798

Example8 requires images from an Asset catalog. Download it from here:
https://swiftui-lab.com/?smd_process_download=1&download_id=916

AnimatableModifier Does Not Animate! Why?

If you are planning on using AnimatableModifier in production code, make sure you read the final section: Dancing with Versions.

If you tried the protocol yourself, chances are, you probably hit a wall at first. I certainly did. For my first try, I wrote a very simple animatable modifier, and yet, the view did not animate. I tried a couple more, and nothing happened. Since we were in early beta stages I thought the feature just wasn't there and abandoned it altogether. Luckily, I persevered later. Let me stress the word: "luckily". It turns out, my first modifier was perfectly fine, but animatable modifiers DO NOT work inside containers. It just so happened that the second time I tried, my view was not inside a container. If I weren't so lucky, you wouldn't be reading this third article.

For example, the following modifier will animate fine:

```
MyView().modifier(MyAnimatableModifier(value: flag ? 1 : 0))
```

But the same modifier, inside a VStack will not:

```
VStack {
    MyView().modifier(MyAnimatableModifier(value: flag ? 1 : 0))
}
```

Ok, so I hear you saying: but you promised me gold! And gold you should get. Until this big issue gets resolved, there's a way of making our animatable modifiers work inside a VStack. We just need one more dirty trick in our bag:

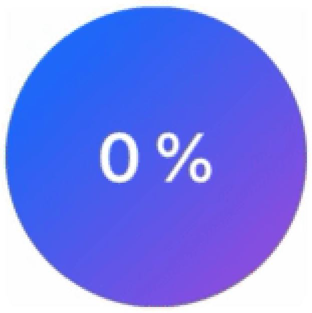
```
VStack {
    Color.clear.overlay(MyView().modifier(MyAnimatableModifier(value: flag ? 1 : 0))).frame(width: 
}
```

We are basically using a transparent view to occupy the space of our actual view, which will be placed above it, using `.overlay()`. The only inconvenience, is we need to know how big the actual view is, so we can set the frame of the transparent view behind it. It can be tricky sometimes, but tricks we have to spare. We'll see in the examples below.

I reported this to Apple, check FB code here. I encourage you to do the same.

Animating Text

Our first goal is to make some text to follow the animation. For this example, we are going to create a loading indicator. It will be a ring with a label:



Our first instinct may be to use an animatable path. However, that will not let us animate the label. Instead we are going to use an AnimatableModifier:

*The full code is available as **Example10**, in the gist file linked at the top of this page.*

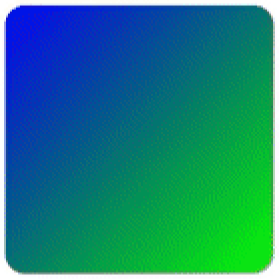
```
struct PercentageIndicator: AnimatableModifier {
    var pct: CGFloat = 0

    var animatableData: CGFloat {
        get { pct }
        set { pct = newValue }
    }

    func body(content: Content) -> some View {
        content
            .overlay(ArcShape(pct: pct).foregroundColor(.red))
            .overlay(LabelView(pct: pct))
    }

    struct ArcShape: Shape {
        let pct: CGFloat

        func path(in rect: CGRect) -> Path {

            var p = Path()

            p.addArc(center: CGPoint(x: rect.width / 2.0, y:rect.height / 2.0),
                     radius: rect.height / 2.0 + 5.0,
                     startAngle: .degrees(0),
                     endAngle: .degrees(360.0 * Double(pct)), clockwise: false)

            return p.strokedPath(.init(lineWidth: 10, dash: [6, 3], dashPhase: 10))
        }
    }

    struct LabelView: View {
        let pct: CGFloat

        var body: some View {
            Text("\(Int(pct * 100)) %")
```

```
        }
        }
    }
```

As you can see in the example, we did not make the ArcShape animatable. It is not necessary, because the modifier is already creating the shape multiple times, with different pct values.

Animating Gradients

If you ever tried to animate a gradient, you probably found out that there are limitations. For example, you can animate starting and ending points, but you cannot animate the gradient colors. Here's where we can also benefit from AnimatableModifier:



This implementation is rather basic, but it is a good starting point if you later need something more elaborate. To interpolate the intermediate colors, we simply calculate an average of its RGB values. Also note that the the modifier assumes the input color arrays (from and to) both contain the same number of colors.

The full code is available as ***Example11****, in the gist file linked at the top of this page.*

```
struct AnimatableGradient: AnimatableModifier {
    let from: [UIColor]
    let to: [UIColor]
    var pct: CGFloat = 0

    var animatableData: CGFloat {
        get { pct }
        set { pct = newValue }
    }

    func body(content: Content) -> some View {
        var gColors = [Color]()

        for i in 0..<from.count {
            gColors.append(colorMixer(c1: from[i], c2: to[i], pct: pct))
        }

        return RoundedRectangle(cornerRadius: 15)
            .fill(LinearGradient(gradient: Gradient(colors: gColors),
                                 startPoint: UnitPoint(x: 0, y: 0),
                                 endPoint: UnitPoint(x: 1, y: 1)))
            .
    }

    // This is a very basic implementation of a color interpolation
```

```
        // between two values.
        func colorMixer(c1: UIColor, c2: UIColor, pct: CGFloat) -> Color {
            guard let cc1 = c1.cgColor.components else { return Color(c1) }
            guard let cc2 = c2.cgColor.components else { return Color(c1) }

            let r = (cc1[0] + (cc2[0] - cc1[0]) * pct)
            let g = (cc1[1] + (cc2[1] - cc1[1]) * pct)
            let b = (cc1[2] + (cc2[2] - cc1[2]) * pct)

            return Color(red: Double(r), green: Double(g), blue: Double(b))
        }
    }
```

More Text Animation

On our next example, we will animate text again. In this case, however, we'll do it progressively: one character at a time:

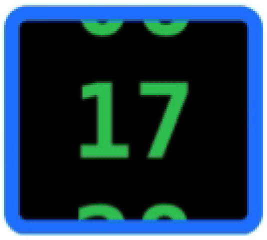
The smooth progressive scaling requires a little math, but the result is worth the effort. *The full code is available as **Example12**, in the gist file linked at the top of this page.*

```
struct WaveTextModifier: AnimatableModifier {
    let text: String
    let waveWidth: Int
    var pct: Double
    var size: CGFloat

    var animatableData: Double {
        get { pct }
        set { pct = newValue }
    }

    func body(content: Content) -> some View {

        HStack(spacing: 0) {
            ForEach(Array(text.enumerated()), id: \.0) { (n, ch) in
                Text(String(ch))
                    .font(Font.custom("Menlo", size: self.size).bold())
                    .scaleEffect(self.effect(self.pct, n, self.text.count, Double(self.waveWidth)))
            }
        }
    }

    func effect(_ pct: Double, _ n: Int, _ total: Int, _ waveWidth: Double) -> CGFloat {
```

```
    func effect(_ pct: Double, _ n: Int, _ total: Int, _ waveWidth: Double) -> CGFloat {
        let n = Double(n)
        let total = Double(total)

        return CGFloat(1 + valueInCurve(pct: pct, total: total, x: n/total, waveWidth: waveWidth))
    }

    func valueInCurve(pct: Double, total: Double, x: Double, waveWidth: Double) -> Double {
        let chunk = waveWidth / total
        let m = 1 / chunk
        let offset = (chunk - (1 / total)) * pct
        let lowerLimit = (pct - chunk) + offset
        let upperLimit = (pct) + offset
        guard x >= lowerLimit && x < upperLimit else { return 0 }

        let angle = ((x - pct - offset) * m)*360-90

        return (sin(angle.rad) + 1) / 2
    }
}

extension Double {
    var rad: Double { return self * .pi / 180 }
    var deg: Double { return self * 180 / .pi }
}
```

Getting Creative

Before we knew anything about the AnimatableModifier, the following example might have seem impossible to achieve. Our next challenge is to create a counter:



The trick of this exercise, is using 5 Text views for each digit and move them up and down, with a .spring() animation. We also need to use a .clipShape() modifier, to hide the part that draws outside the border. To understand better how it works, you may comment the .clipShape() and slow down the animation considerably. *The full code is available as **Example13**, in the gist file linked at the top of this page.*

```
struct MovingCounterModifier: AnimatableModifier {
        @State private var height: CGFloat = 0

        var number: Double

        var a
            g
            set { number = newValue }
```

```
    }

    func body(content: Content) -> some View {
        let n = self.number + 1

        let tOffset: CGFloat = getOffsetForTensDigit(n)
        let uOffset: CGFloat = getOffsetForUnitDigit(n)

        let u = [n - 2, n - 1, n + 0, n + 1, n + 2].map { getUnitDigit($0) }
        let x = getTensDigit(n)
        var t = [abs(x - 2), abs(x - 1), abs(x + 0), abs(x + 1), abs(x + 2)]
        t = t.map { getUnitDigit(Double($0)) }

        let font = Font.custom("Menlo", size: 34).bold()

        return HStack(alignment: .top, spacing: 0) {
            VStack {
                Text("\(t[0])").font(font)
                Text("\(t[1])").font(font)
                Text("\(t[2])").font(font)
                Text("\(t[3])").font(font)
                Text("\(t[4])").font(font)
            }.foregroundColor(.green).modifier(ShiftEffect(pct: tOffset))

            VStack {
                Text("\(u[0])").font(font)
                Text("\(u[1])").font(font)
                Text("\(u[2])").font(font)
                Text("\(u[3])").font(font)
                Text("\(u[4])").font(font)
            }.foregroundColor(.green).modifier(ShiftEffect(pct: uOffset))
        }
        .clipShape(ClipShape())
        .overlay(CounterBorder(height: $height))
        .background(CounterBackground(height: $height))
    }

    func getUnitDigit(_ number: Double) -> Int {
        return abs(Int(number) - ((Int(number) / 10) * 10))
    }

    func getTensDigit(_ number: Double) -> Int {
        return abs(Int(number) / 10)
    }

    func getOffsetForUnitDigit(_ number: Double) -> CGFloat {
        return 1 - CGFloat(number - Double(Int(number)))
    }

    func getOffsetForTensDigit(_ number: Double) -> CGFloat {
        if getUnitDigit(number) == 0 {
            return 1 - CGFloat(number - Double(Int(number)))
        } else {
            return 0
        }
    }

}
```

Animating Text Color

If you ever tried to animate .foregroundColor(), you may have noticed it works nicely, except when the view is of type Text. I don't know if it is a bug, or missing functionality. Nevertheless, should you need to animate the color of text, you can do so with an AnimatableModifier like the one below. *The full code is available as **Example14**, in the gist file linked at the top of this page.*

```
struct AnimatableColorText: View {
    let from: UIColor
    let to: UIColor
    let pct: CGFloat
    let text: () -> Text

    var body: some View {
        let textView = text()

        return textView.foregroundColor(Color.clear)
            .overlay(Color.clear.modifier(AnimatableColorTextModifier(from: from, to: to, pct: pct,
    }

    struct AnimatableColorTextModifier: AnimatableModifier {
        let from: UIColor
        let to: UIColor
        var pct: CGFloat
        let text: Text

        var animatableData: CGFloat {
            get { pct }
            set { pct = newValue }
        }

        func body(content: Content) -> some View {
            return text.foregroundColor(colorMixer(c1: from, c2: to, pct: pct))
        }

        // This is a very basic implementation of a color interpolation
        // between two values.
        func colorMixer(c1: UIColor, c2: UIColor, pct: CGFloat) -> Color {
            guard let cc1 = c1.cgColor.components else { return Color(c1) }
            guard let cc2 = c2.cgColor.components else { return Color(c1) }

            let r = (cc1[0] + (cc2[0] - cc1[0]) * pct)
            let g = (cc1[1] + (cc2[1] - cc1[1]) * pct)
            l

            return Color(red: Double(r), green: Double(g), blue: Double(b))
        }
    }
}
```

Dancing With Versions

We've seen that AnimatableModifier is very powerful… but also, a little buggy. The biggest issue is that under certain combinations of Xcode and iOS/macOS versions, the application will simply crash at launch. What's even worst, that normally happens when deploying the app, but not when compiling and running with Xcode during normal development. You may spend a lot of time developing and debugging, thinking all is good with the world, but then deploy, and you get something like this:

```
dyld: Symbol not found:
_$s7SwiftUI18AnimatableModifierPAAE13_makeViewList8modifier6inputs4bodyAA01_fG7Outpu
tsVAA11_GraphValueVyxG_AA01_fG6InputsVAiA01_L0V_ANtctFZ
  Referenced from: /Applications/MyApp.app/Contents/MacOS/MyApp
  Expected in: /System/Library/Frameworks/SwiftUI.framework/Versions/A/SwiftUI
```

For example, if the app is deployed with Xcode 11.3, and executed on macOS 10.15.0 it will fail to launch with the "Symbol not found" error. However, running the same executable on 10.15.1 works fine.

On the contrary, if we deploy with Xcode 11.1, it works fine with all macOS versions (at least the ones I tried).

Something similar happens with iOS. An app that uses AnimatableModifier, deployed with Xcode 11.2 will fail to launch on iOS 13.2.2 but will work fine on iOS 13.2.3.

For the time being, I'll keep using Xcode 11.1 for my macOS projects that need AnimatableModifier. In the future, I will probably use a newer version of Xcode, but increase the app requirement to macOS 10.15.1 (unless the problem gets fixed, which I seriously doubt).

Summary / What's Next

We've seen how simple the Animatable protocol is and how much it has to offer. Put your creativity to work, and the results will be spectacular.

This concludes the "Advanced SwiftUI Animations" series. Soon I will be posting an article on custom Transitions, which will complement nicely with this series. Make sure you follow me on twitter, if you want to be notified when new articles are published. The link is below. Until next time.

Animations, SwiftUI

‹ Trigonometric Recipes for SwiftUI

› Dismiss Gesture for SwiftUI Modals

13 thoughts on “Advanced SwiftUI Animations – Part 3: AnimatableModifier”

filimo
September 19, 2019 at 4:31 pm

This article is just a bomb, MovingCounter is awesome.
ColorMixer is a very useful function for animating the transition between two colors.
I found a small typo in valueInCurve, you may have done on purpose.

```
let angle: Angle = .degrees(((x - pct - offset) * m)*360-90) return (sin(angle.radians)
+ 1) / 2
```

Thanks a lot.

Reply

javier
September 19, 2019 at 4:50 pm

Hi filimo, thanks! It's not a typo. It is using an extension from Example 12. Perhaps if you copied only part of the code, the extension was left out. Your fix however, will also work.

The extension was:
```
extension Double { var rad: Double { return self * .pi / 180 } var deg: Double {
return self * 180 / .pi } }
```

Reply

Lucas
October 31, 2019 at 1:54 pm

Very nice article. Very useful information, keep up the good work!
I am playing around with SwiftUI and the main difficulty (for me) is that when it does not work, I am lost how to debug the issue. Maybe a good article about debugging SwiftUI? Would be great!
Apparently the bug (https://swiftui-lab.com/animatablemodifier-inside-containers-bug/) has been solved, at least in Xcode 11.2 beta 2 (11B44) the ContentView2 (with the Rectangle in a VStack) works as expected, although ContentView1 does not work anymore.

Reply

Alex
November 9, 2019 at 1:28 pm

In Xcode 11.2, many animation-specific bugs have been resolved.
The workaround is no longer needed ;D

Reply

javier
November 10, 2019 at 9:31 am

Thanks! I'll keep the workaround around… just in case anyone needs to still target iOS13.0. Have you tried using XCode 11.2 to target iOS 13.0 and see if the workaround is still not needed when not running on iOS13.2 or higher? I may need to test that before scraping the workaround completely.

Reply

Kevin
November 17, 2019 at 8:57 pm

With Xcode 11.2, when building for App Store or Ad Hoc I get a SIGABRT crash on app launch but not when built directly to device that didn't happen with 11.1. Specifically, with animating the text label in Example 10. Not sure how to address a fix for this.

Termination Description: DYLD, symbol '_$s7SwiftUI18AnimatableModifierPAAE13_makeViewList8modifier6inputs4bodyAA01_fG7OutputsVAA11_GraphValueVyxG_AA01_fG6InputsVAiA01_L0V_ANtctFZ' not found, expected in '/System/Library/Frameworks/SwiftUI.framework/SwiftUI'

Reply

javier

November 18, 2019 at 6:59 pm

Thank you, Kevin, I haven't seen that problem, but I will have a look and let you know if I find something.

Reply

Jean-Charles Mourey

December 7, 2019 at 9:12 pm

I get a SIGABRT crash with AnimatableModifiers on an iPhone 8, whether built for App Store or directly to device, but works fine on iPhone 11 Pro Max, iPad Pro.

XCode 11.2.1 (11B500)

Works fine in iPhone 8 Simulator.

Crash goes away as soon as I remove the AnimatableModifier.

The AnimatableModifier I was using is a modified version of MovingCounter, but it crashes even when I put in an empty modifier to test. And if I change the modifier from AnimatableModifier to ViewModifier, crash goes away.

Can't fig

Any ideas?

Reply

javier

December 9, 2019 at 5:50 am

After exchanging several emails with you, I'm glad you found that the problem is fixed in iOS 13.2.3.

Reply

Maciek

October 1, 2020 at 1:28 pm

I love your examples, Javier. They are absolutely unconventional.
The example with a counter is also doable with just 3 Texts per digit. Leaving the gist here just in case: https://gist.github.com/Czajnikowski/1c75f29809b4afc5ef293b2bf89ea324

Reply

Leon

October 5, 2020 at 2:05 pm

Hi, I like the approach in the "Animating Gradients" section. However, I am wondering if there is a way not to hardcode the Rectangle() in the "AnimatableGradient" struct? The problem is that ".fill" is not available for "Content". Does somebody have an idea how that would be possible? Thanks for answering!

Reply

abhi137

December 28, 2020 at 10:44 am

Javier,

Your site Is a great resource! I found a very simple way to animate the Text properties after reading your id() post. I was looking for a simple way to fade in text changes. Something like this worked for me –

```
Text(text).font(.largeTitle)
                .id(pscale)
                .onTapGesture {
                    withAnimation {
                        pscale += 1
                        text = String(text.reversed())
                    }
                }
```

This works for foreground/background color changes as well.

Reply

javier

December 31, 2020 at 7:07 am

Nice example, thank you for sharing! I'm guessing you already know this, but for anyone else reading your example, it is worth clarifying that you are not really animating the text properties. In reality you are replacing one Text view with another Text view. Because both views transition in and out with their default transition effect (opacity), it produces the illusion that there is only one view transforming. This is explained in the second example of the article you mentioned: Identifying SwiftUI Views, section "Triggering Transitions".

Reply

Leave a Comment

Name *

Email *

Website

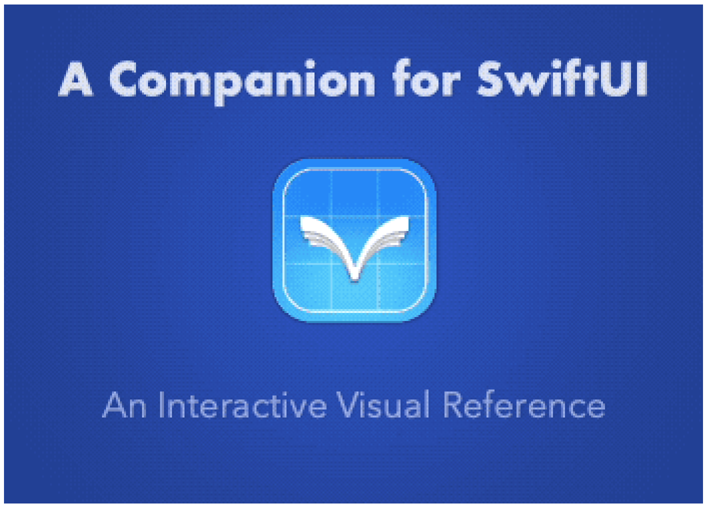
I'm not a robot

reCAPTCHA
Privacy - Terms

Replies to my comments Notify me of followup comments via e-mail. You can also subscribe without commenting.

Post Comment

Search ...



Categories

Animations
Drawing
Layout
SwiftUI
User Interface

Archives

September 2021
June 2021
November 2020
September 2020
July 2020
June 2020
March 2020
February 20
December 2

November 2019

October 2019

September 2019

August 2019

July 2019

June 2019

javier@swiftui-lab.com

@SwiftUILab