Prototyping Custom UI in SwiftUI

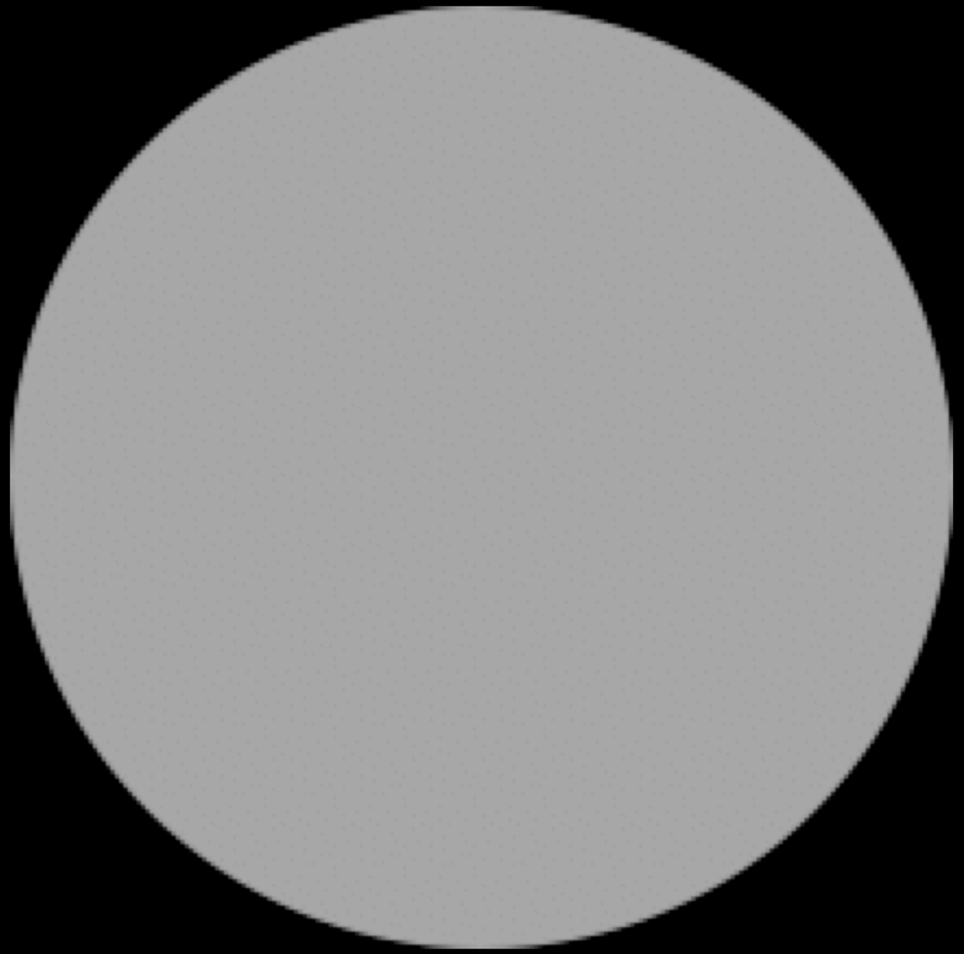
Tobias
Due
Munk

dotSwift

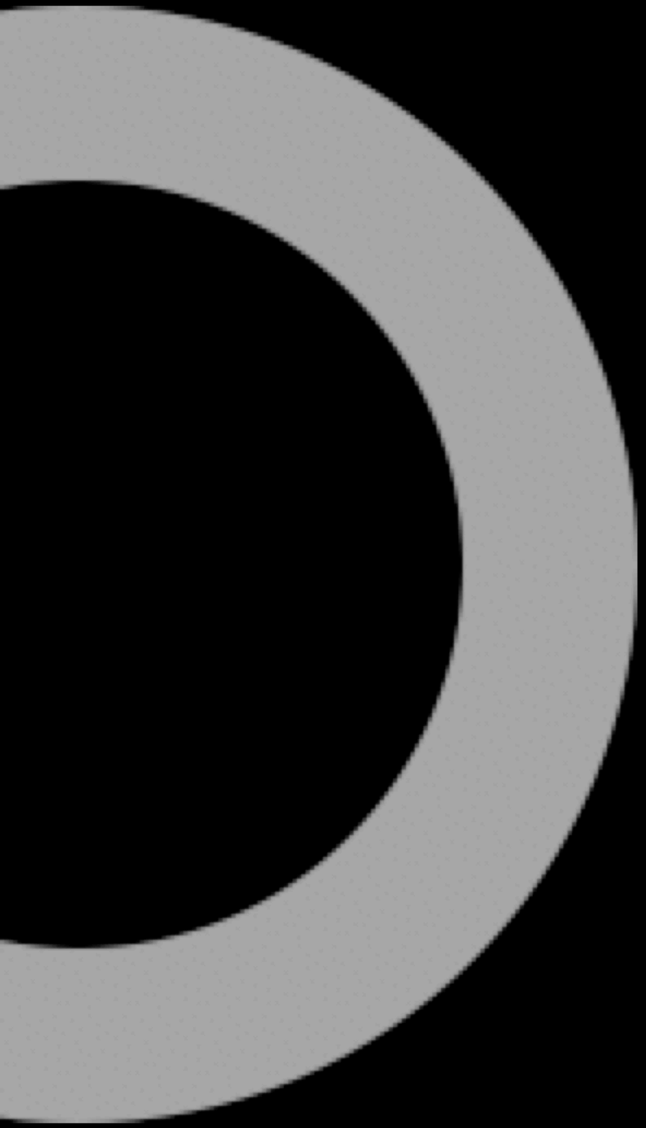
February 3
2020

Visualize Sound Level

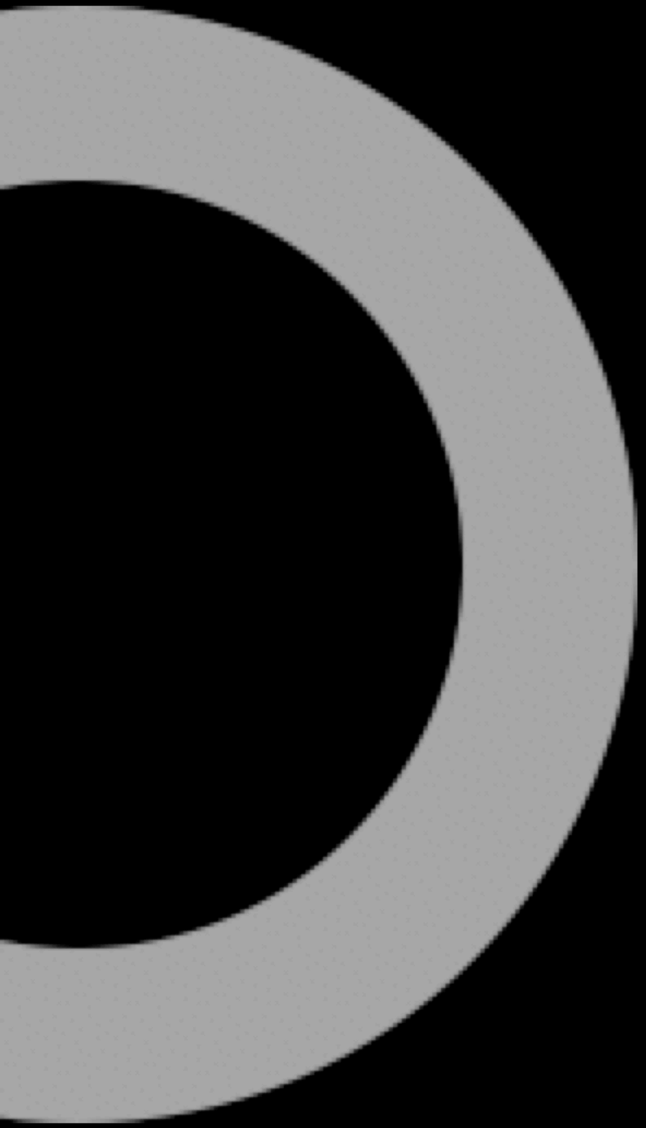
```
struct Gauge: View {
    var body: some View {

    }
}
```



```
struct Gauge: View {
    var body: some View {
        Circle()
    }
}
```

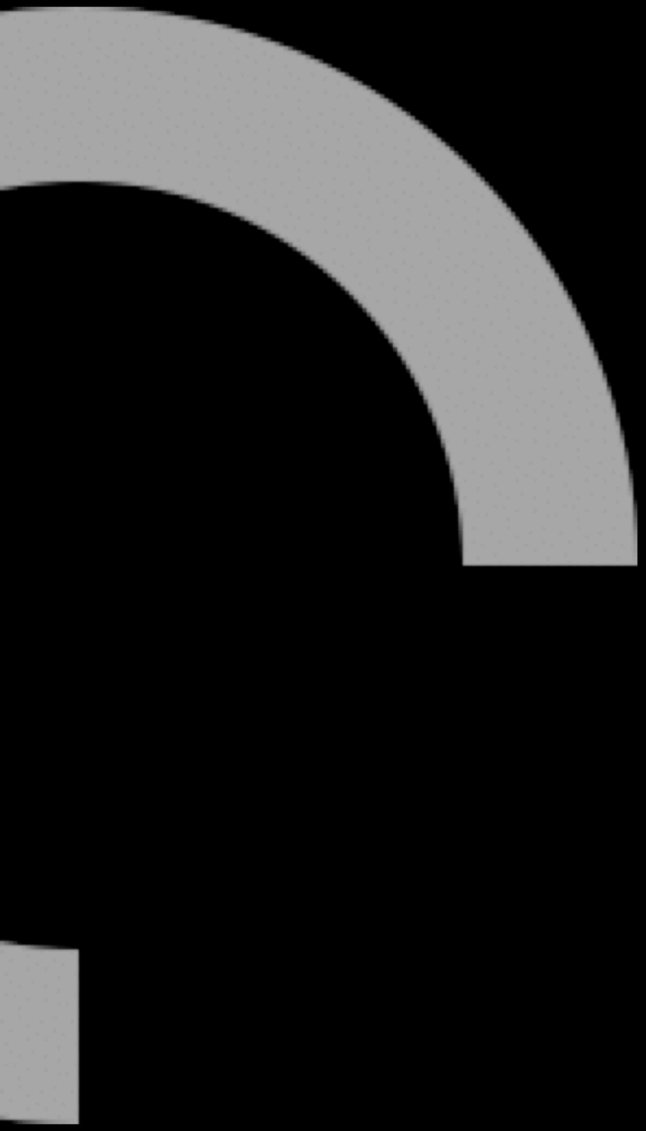


```
struct Gauge: View {
    var body: some View {
        Circle()
            .stroke(style:
                StrokeStyle(lineWidth: 40)
            )
    }
}
```



```
struct Gauge: View {
    let thickness: CGFloat = 40

    var body: some View {
        Circle()
            .stroke(style:
                StrokeStyle(lineWidth: thickness)
            )
    }
}
```

```
struct Gauge: View {
    let thickness: CGFloat = 40
    let trim: ClosedRange<CGFloat> = 0...0.65

    var body: some View {
        Circle()
            .trim(from: trim.lowerBound,
                  end: trim.upperBound)
            .stroke(style: ...)
    }
}
```



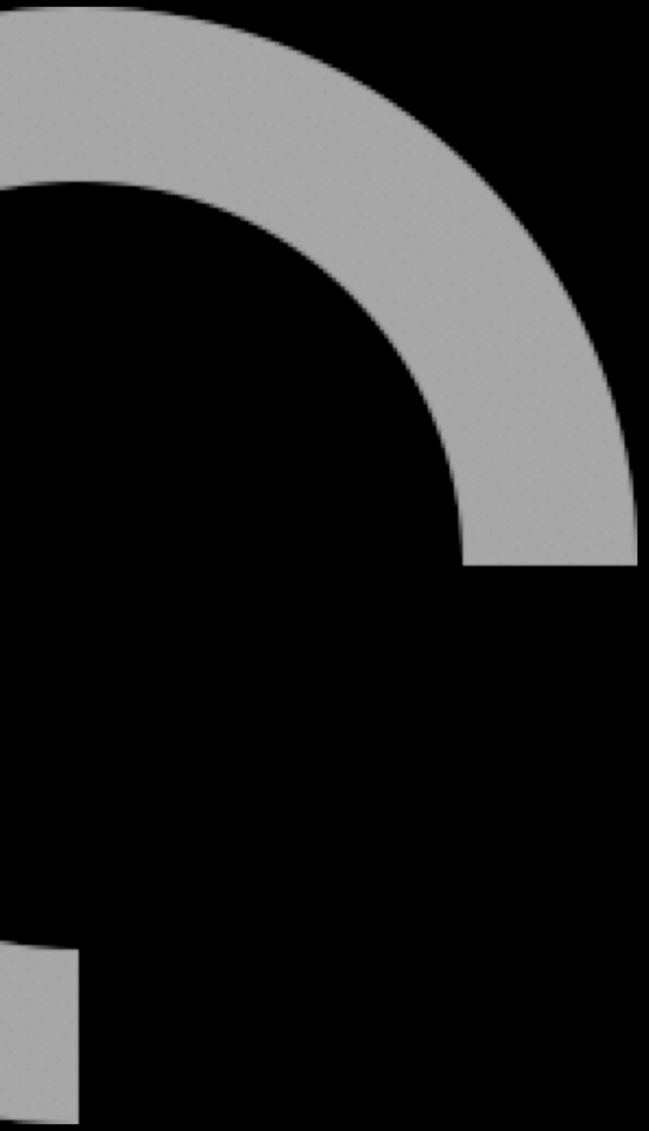
```
struct Gauge: View {
    var thickness: CGFloat = 40
    var trim: ClosedRange<CGFloat> = 0...1

    var body: some View {
        Circle()
            .trim(from: trim.lowerBound,
                  end: trim.upperBound)
            .stroke(style: ...)
    }
}

Gauge()
Gauge(thickness: 30, trim: 0...0.75)
Gauge(trim: 0...0.75)
```

Gauge()

```
AngularGradient(gradient: ...,
                center: .center)
    .mask(Gauge())
```

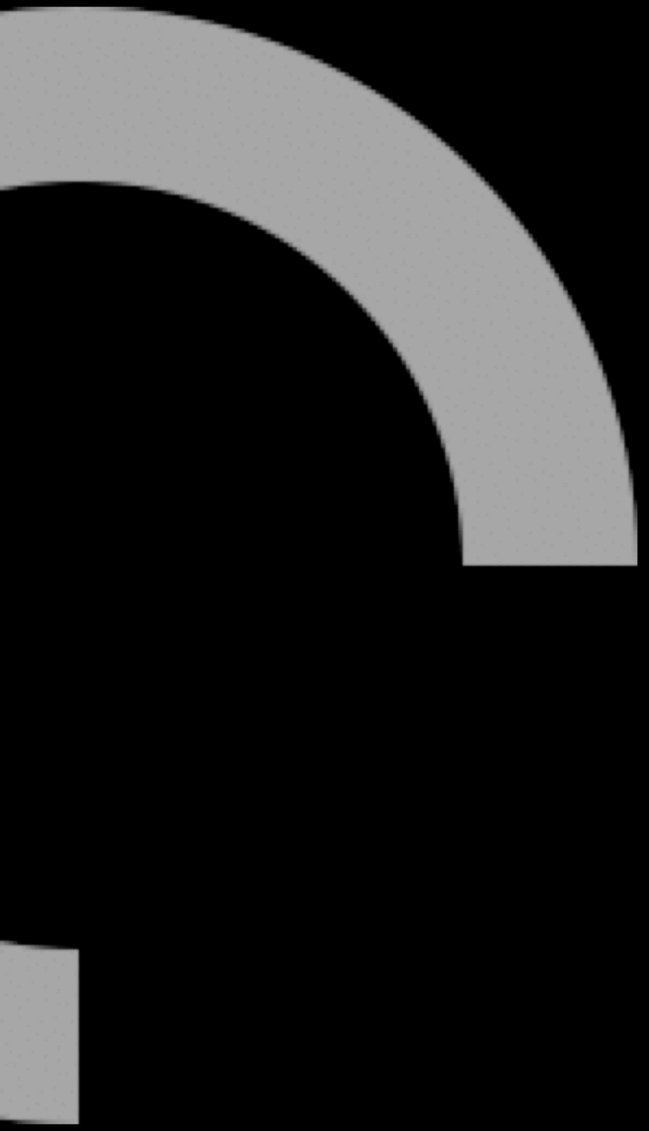
```
Gauge()
    .maskContent(using:
        AngularGradient(gradient: ...,
                        center: .center)
    )
```

```
Gauge()
    .maskContent(using:
        AngularGradient(gradient: ...,
                        center: .center)
    )

extension View {
    func maskContent<T: View>(using: T) -> some View {
        using.mask(self)
    }
}
```



```swift
Gauge()
//   .maskContent(using:
//       AngularGradient(gradient: ...,
//                       center: .center)
//   )

extension View {
    func maskContent<T: View>(using: T) -> some View {
        using.mask(self)
    }
}
```

```
var colorize: Bool = false

Gauge()
    .if(colorize) {
        $0.maskContent(...)
    }
```



```swift
extension View {

    func `if`<T: View>(_ conditional: Bool,
                       transform: (Self) -> T) -> some View {
        Group {
            if conditional {
                transform(self)
            } else {
                self
            }
        }
    }
}
```

```
var colorize: Bool = true

Gauge()
    .if(colorize) {
        $0.maskContent(...)
    }
```

```
var thickness: CGFloat

Gauge(thickness: thickness)
```

```
@State var thickness: CGFloat

Gauge(thickness: thickness)

Slider($thickness, in: 10...60) {
    Text("Thickness: \(thickness)")
}
```

Thickness: 40

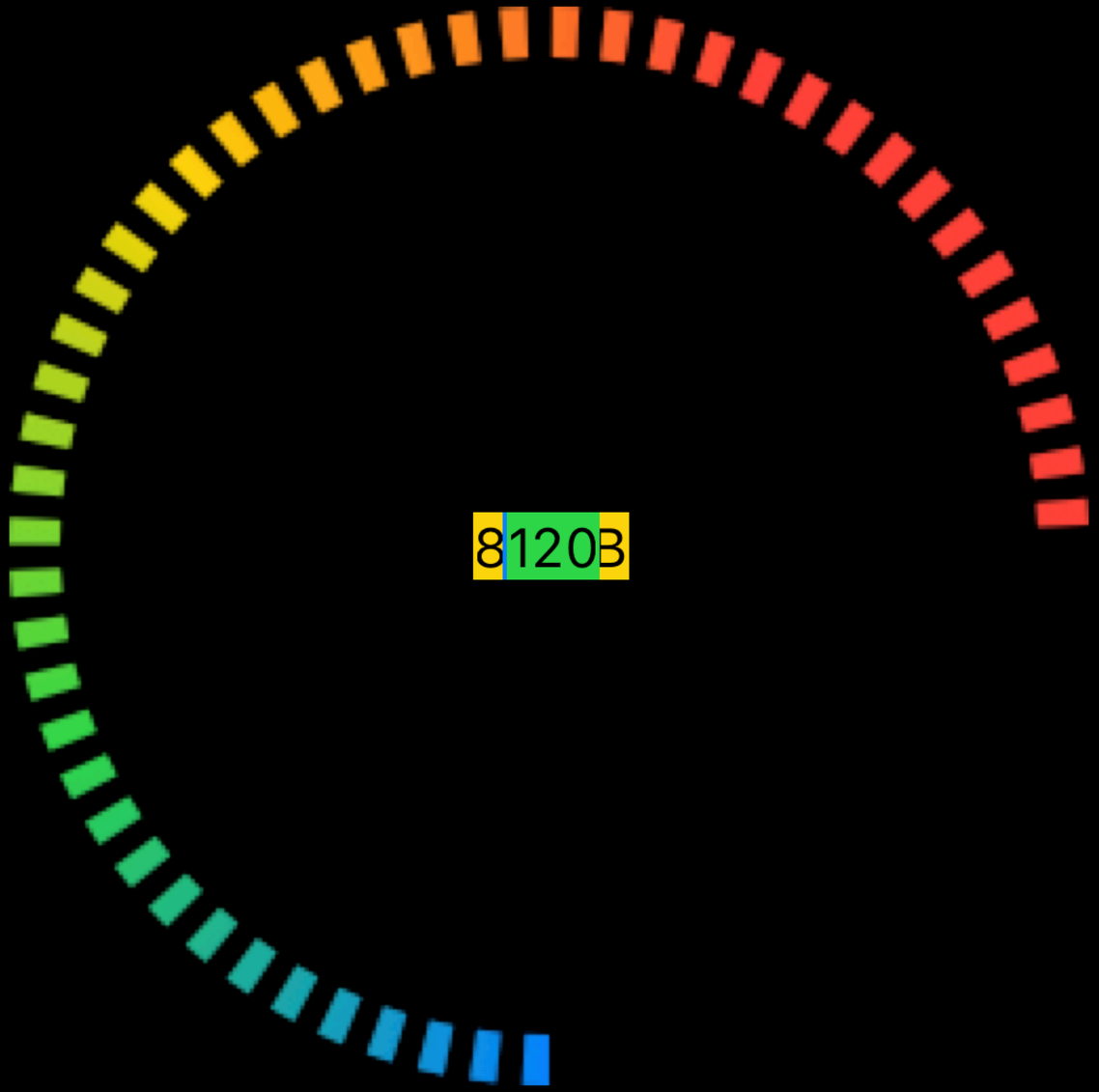
```
@State var thickness: CGFloat = 40
@State var gap: CGFloat = 0

Gauge(thickness: thickness, dashGap: gap)

Slider($thickness, in: 10...60) {
    Text("Thickness: \(thickness)")
}
Slider($gap, in: 0...20) {
    Text("Dash Gap: \(gap)")
}
```

Thickness: 40

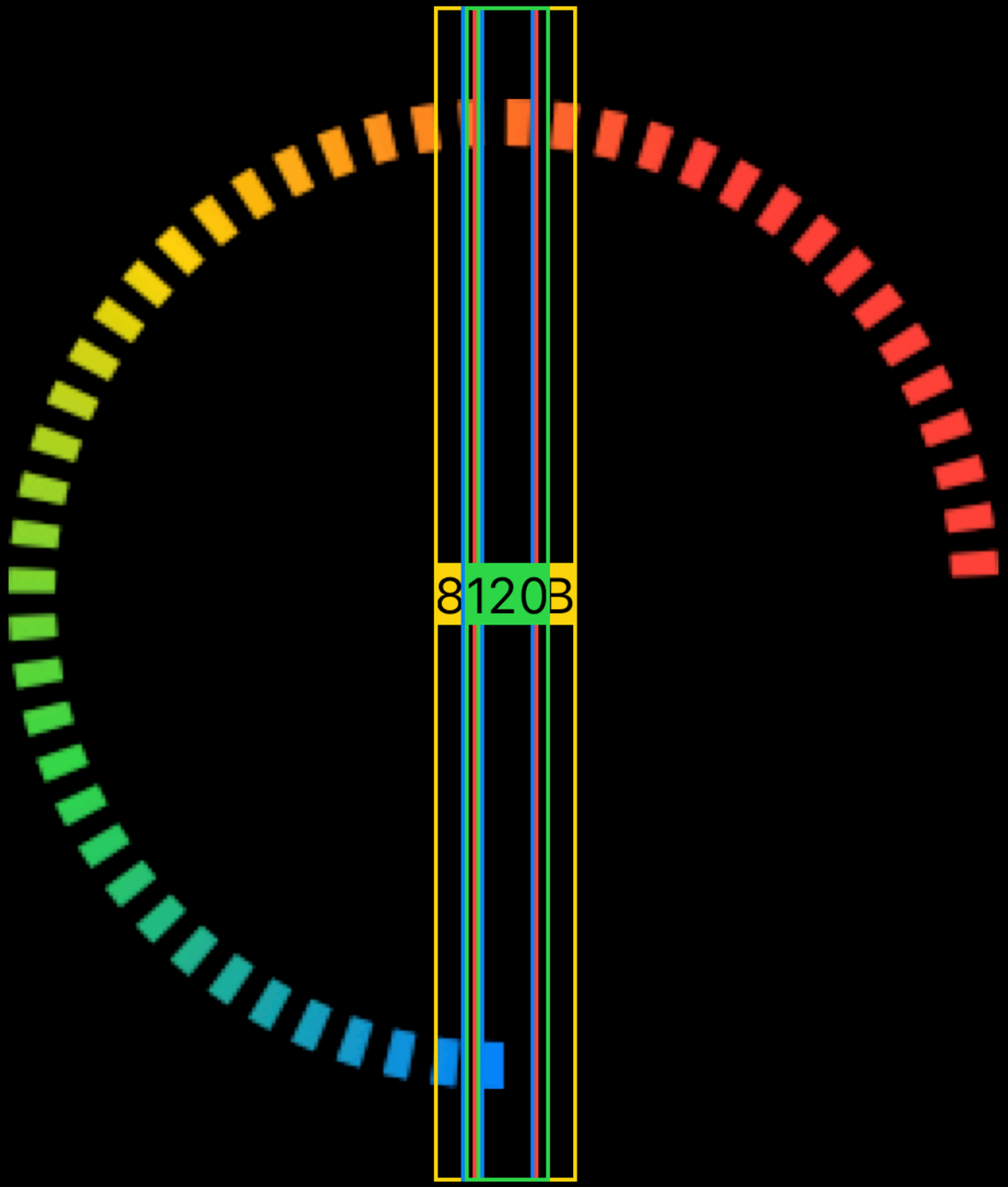
Dash Gap: 0

ZStack {}

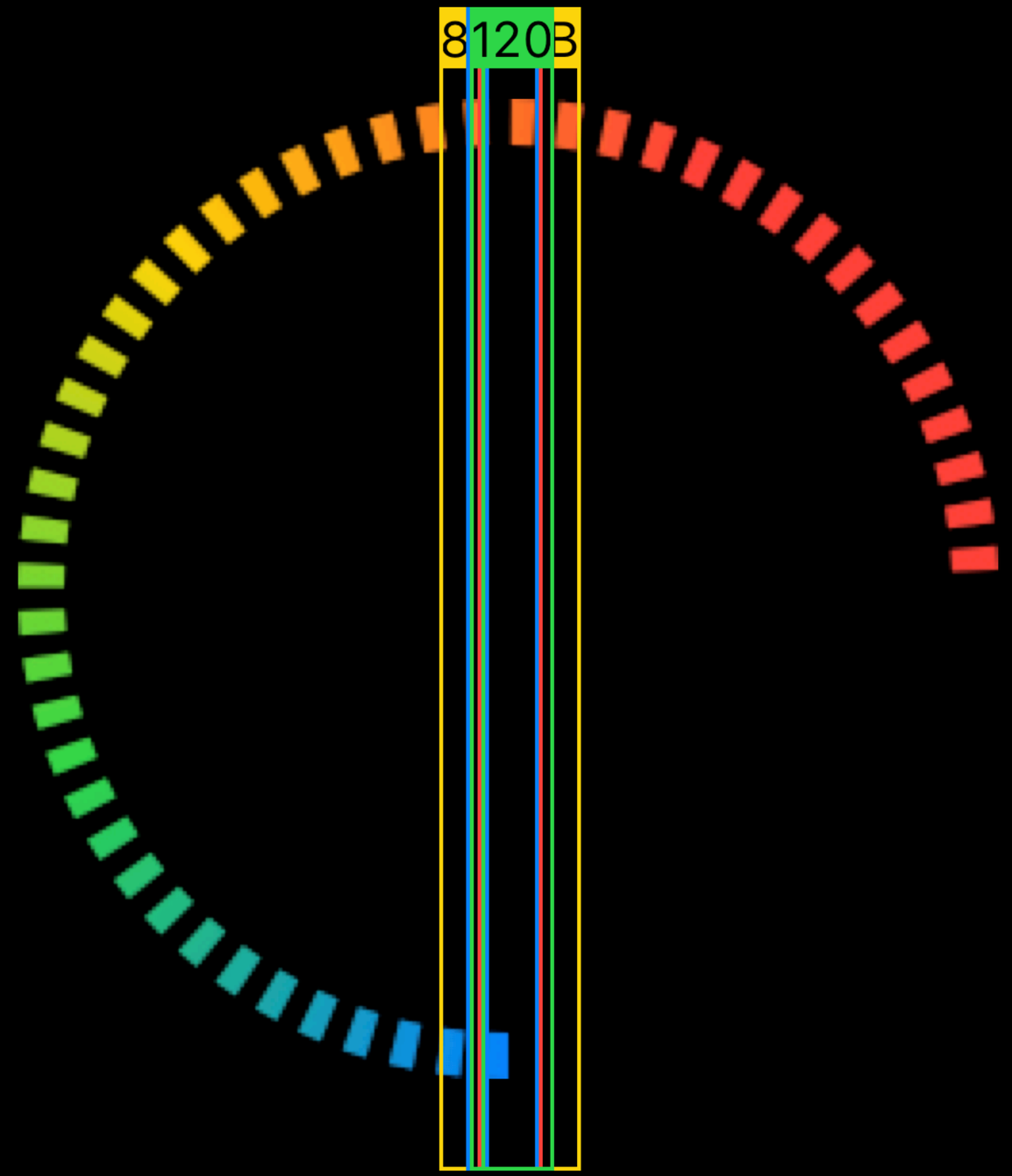

```
ZStack {
    ForEach(0...9, id: \.self) { i in
        Text(decibels(at: i))
    }
}
```



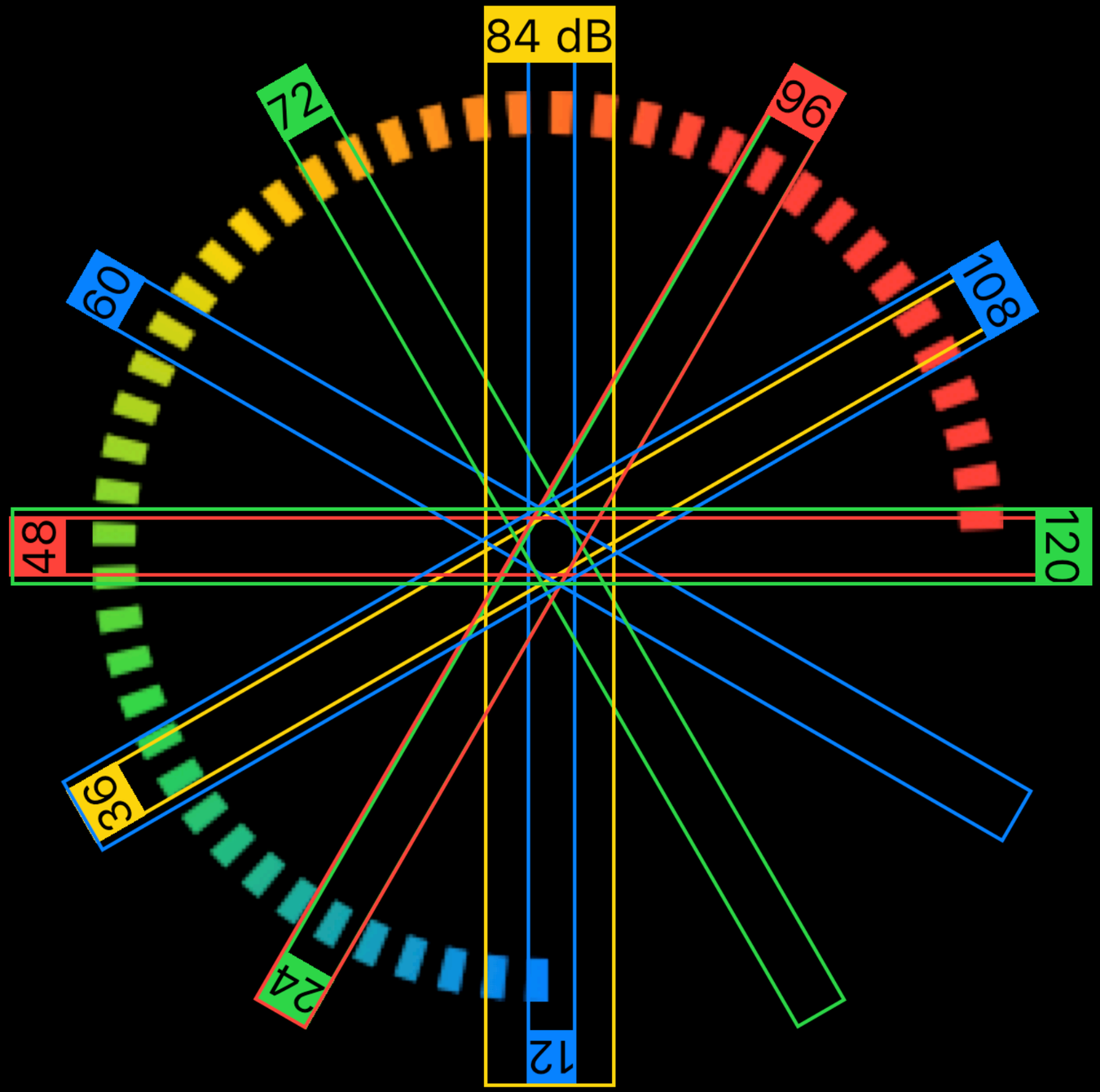

```
Text(decibels(at: i))
    .background(/* Color */)
```



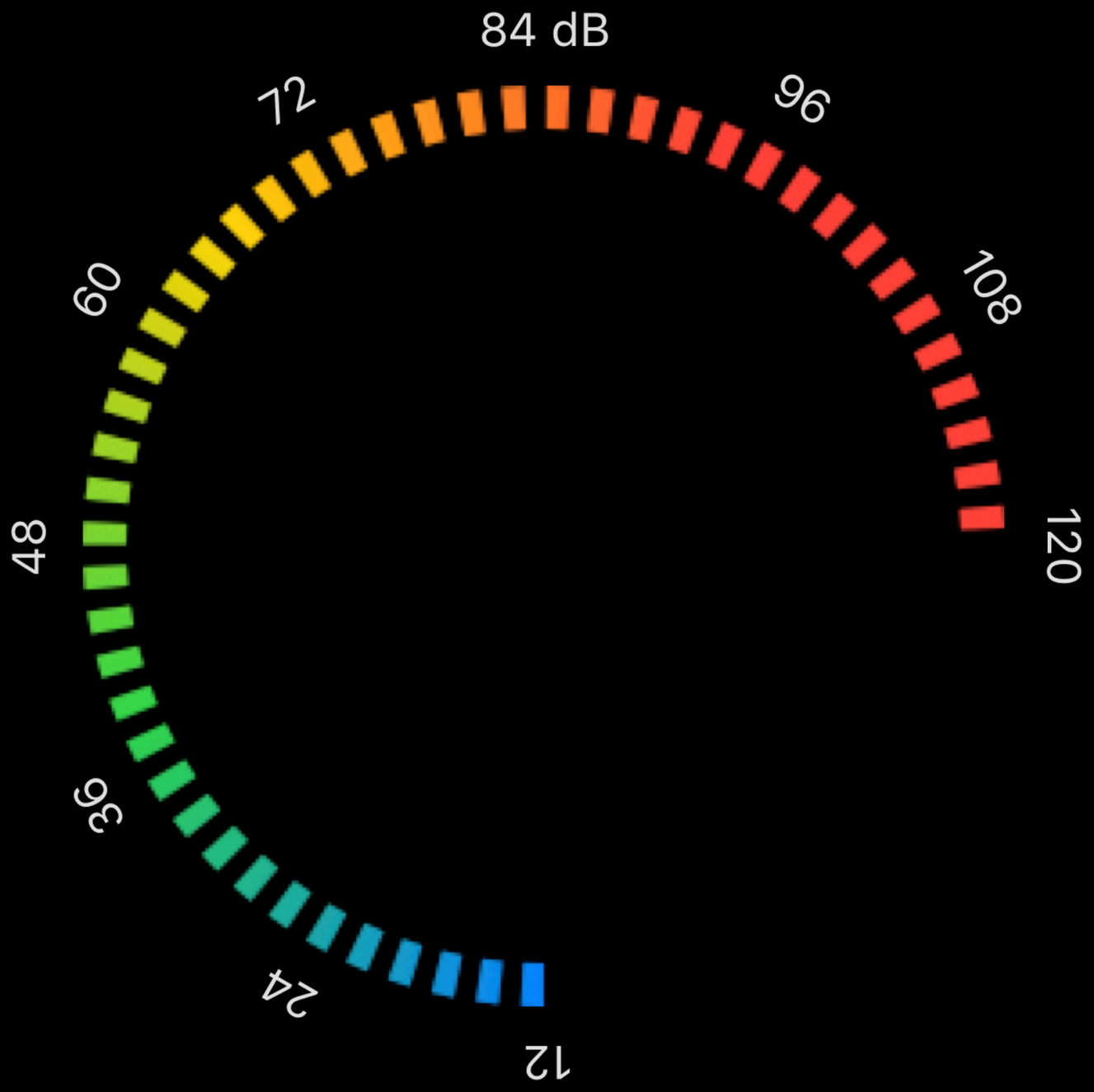

```
Text(decibels(at: i))
    .background(/* Color */)
    .frame(maxHeight: .infinity)
    .border(/* Color */)
```



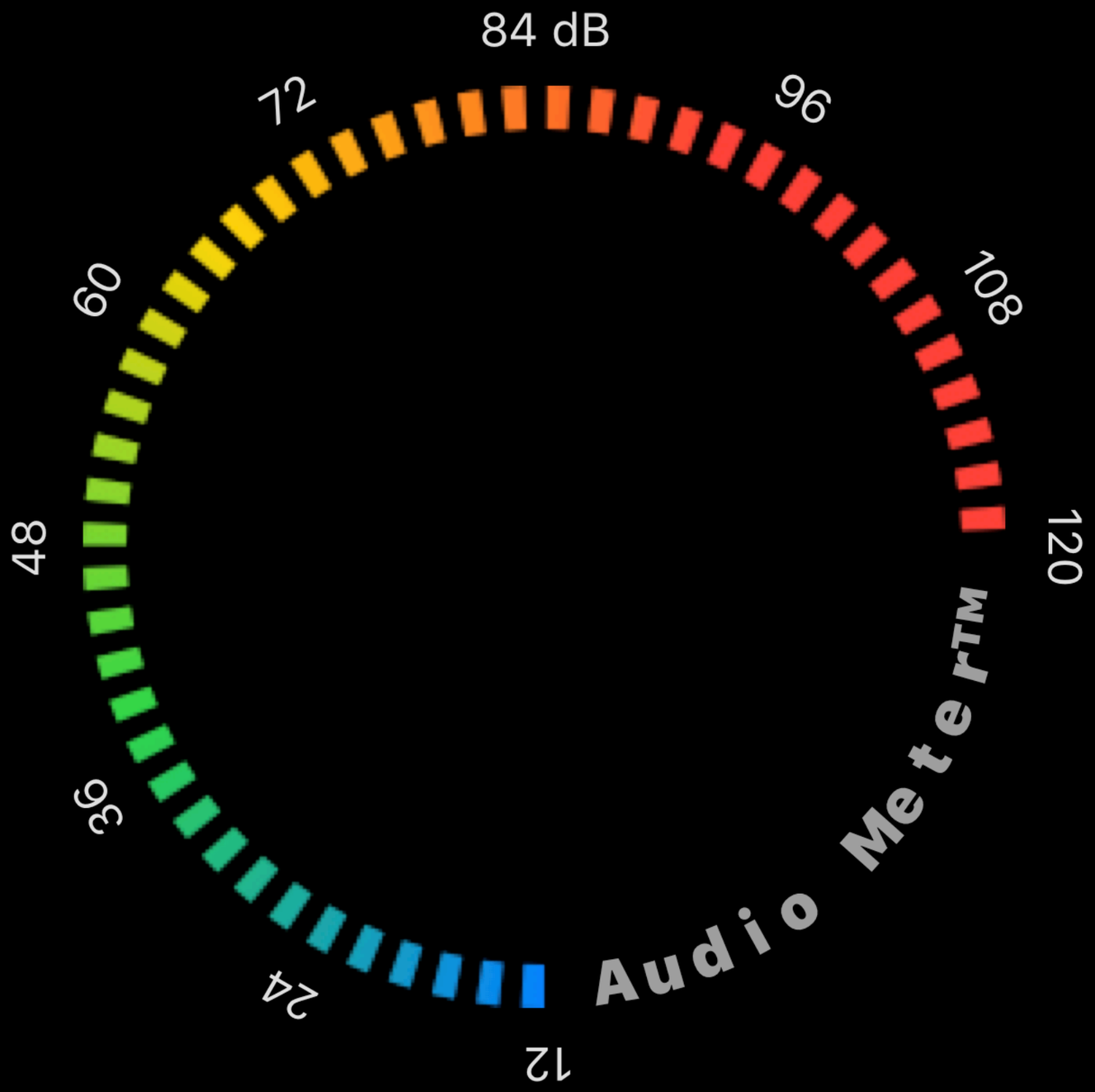

```
Text(decibels(at: i))
    .background(/* Color */)
    .frame(maxHeight: .infinity,
           alignment: .top)
    .border(/* Color */)
```




```
Text(decibels(at: i))
    .background(/* Color */)
    .frame(maxHeight: .infinity,
           alignment: .top)
    .border(/* Color */)
    .rotationEffect(.degrees(i * 30))
```

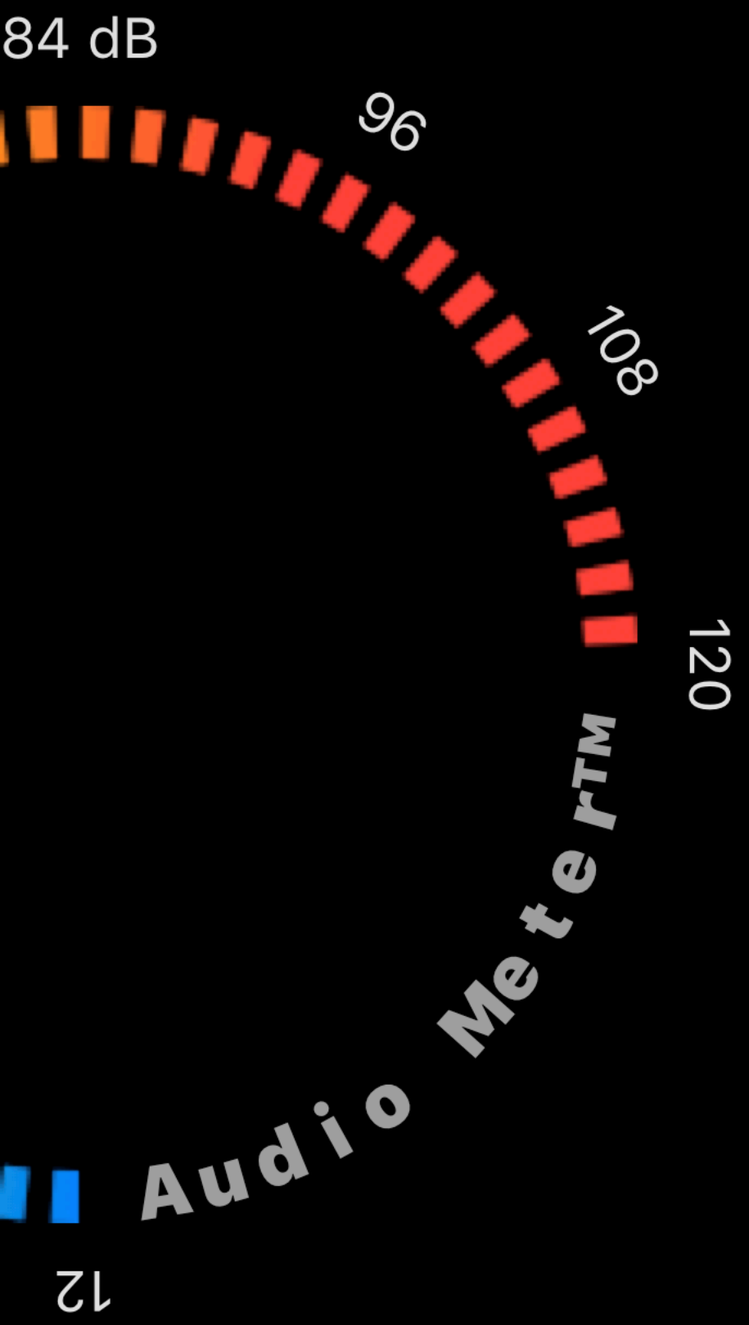



```
Text(decibels(at: i))
    .frame(maxHeight: .infinity,
           alignment: .top)
    .rotationEffect(.degrees(i * 30))
```

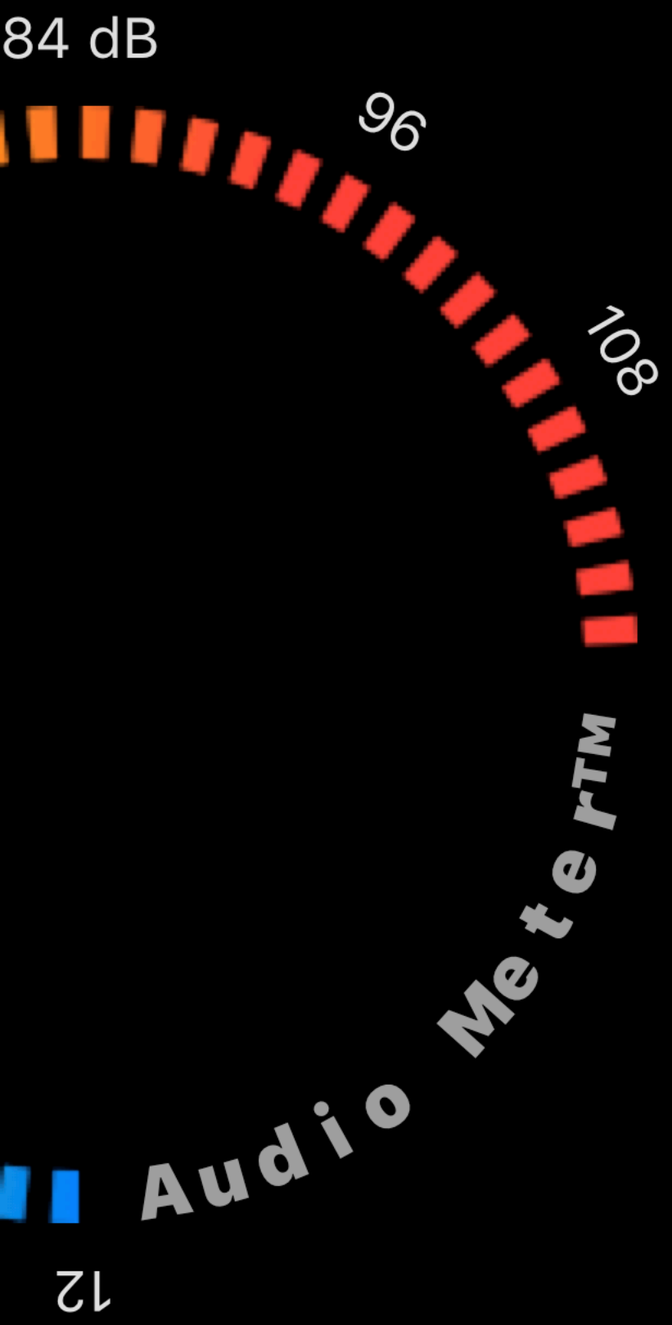



```
Text(character)
    .frame(maxHeight: ...,
           alignment: .bottom)
    .rotationEffect(/* Angle */)
```

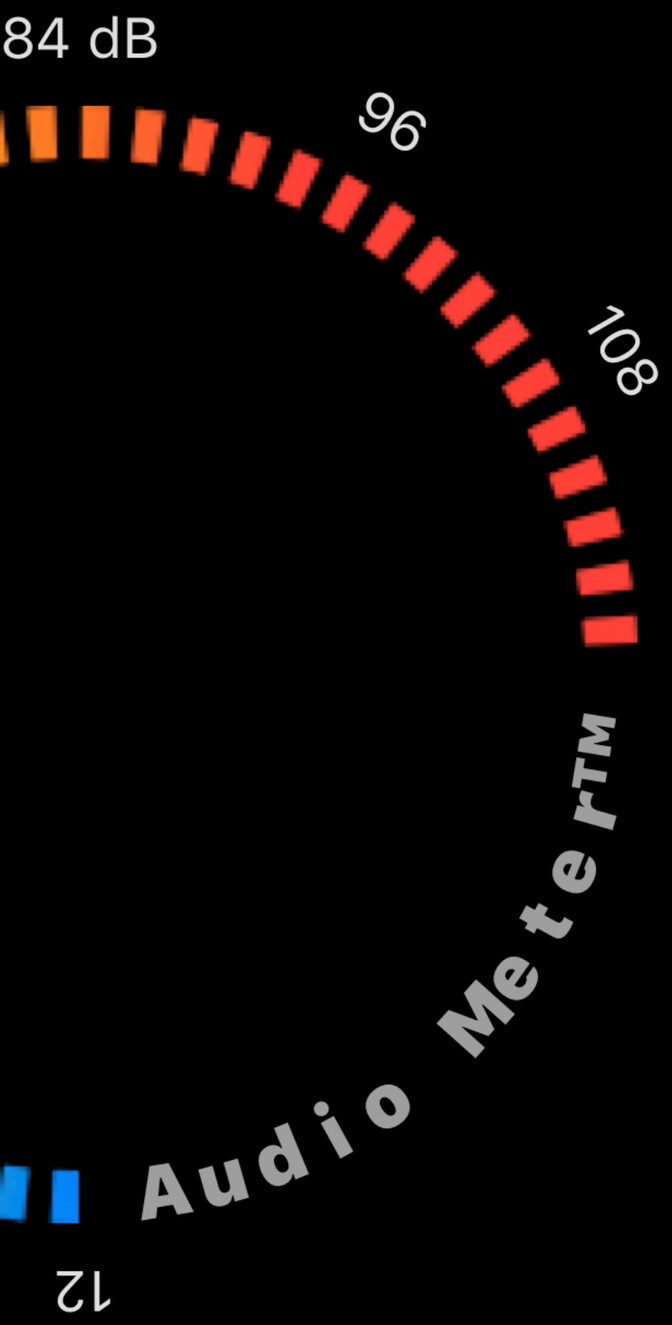
```
Text(character)
    .background(

    )
```



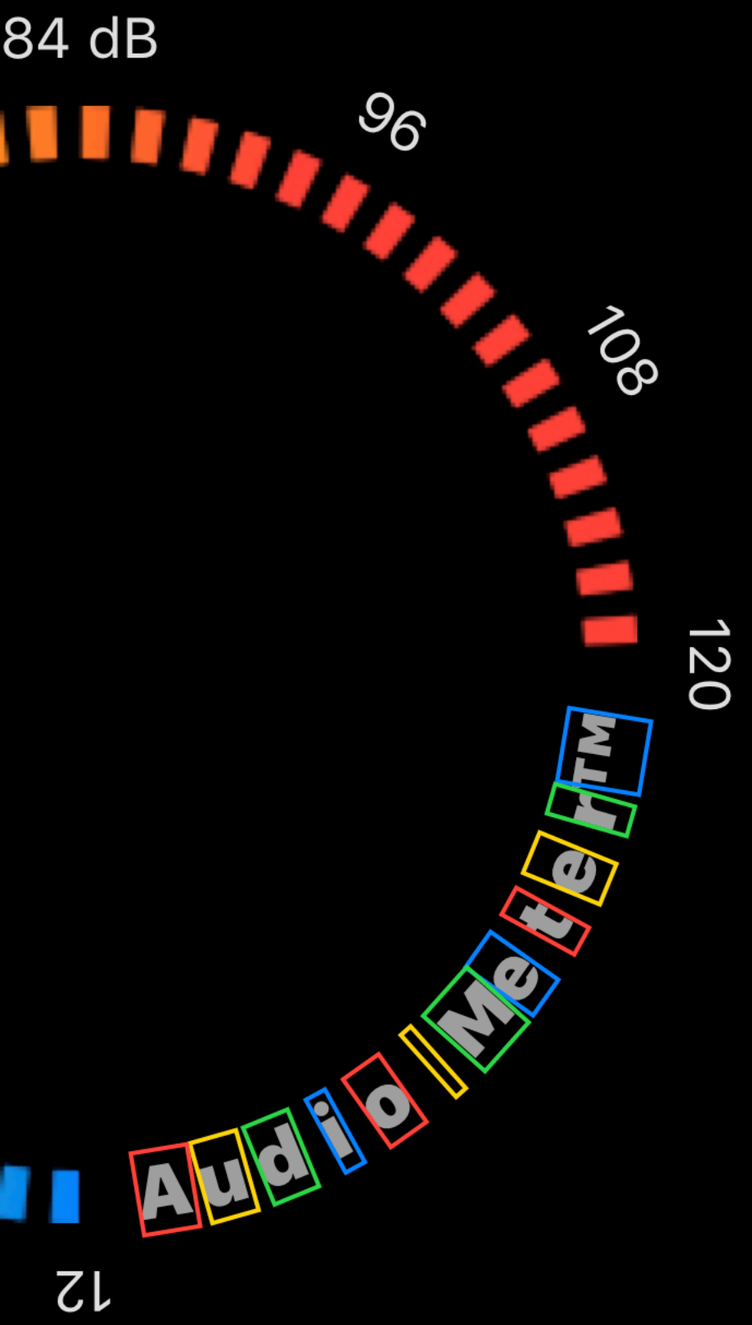

```
Text(character)
    .background(
        GeometryReader { proxy in
            /* proxy.size */
        }
    )
```



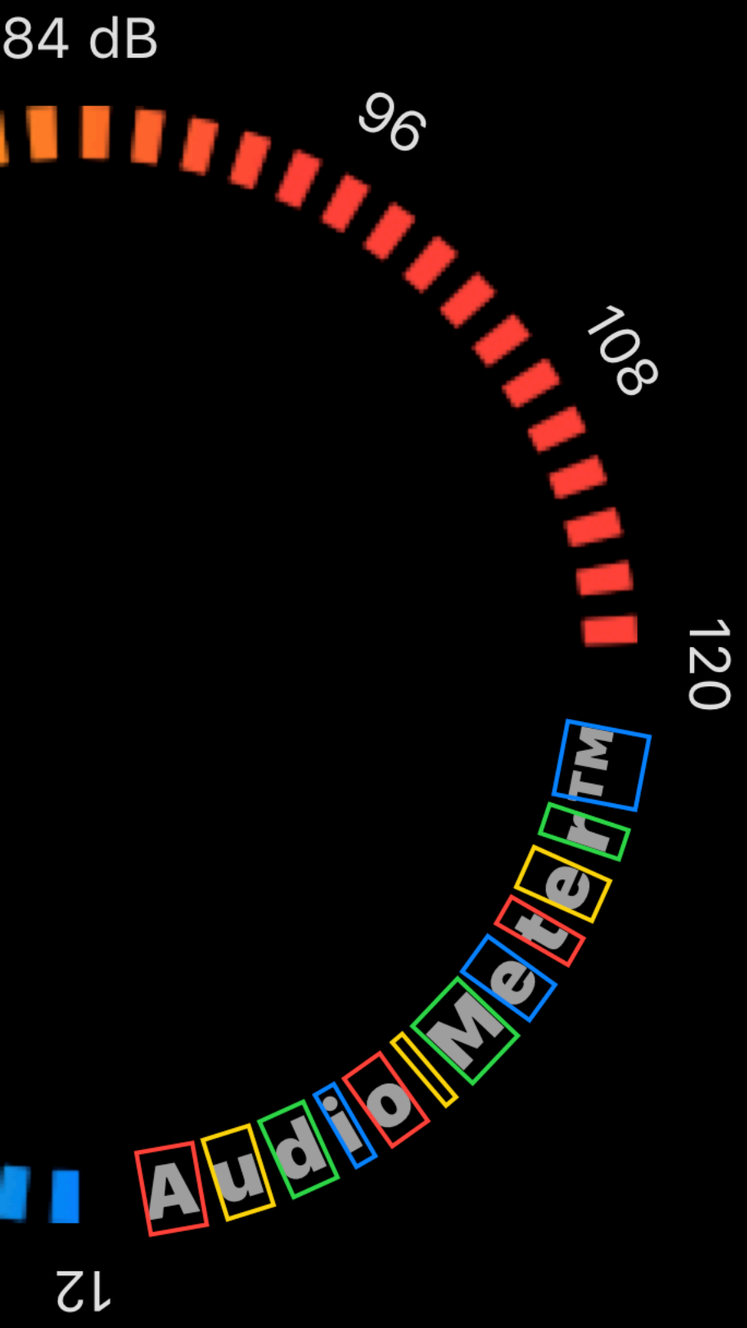

```
private struct SizeKey: PreferenceKey {
    static var defaultValue: [CGSize] { [] }

    static func reduce(value: inout [CGSize],
                       nextValue: () -> [CGSize]) {
        value.append(contentsOf: nextValue())
    }
}
```



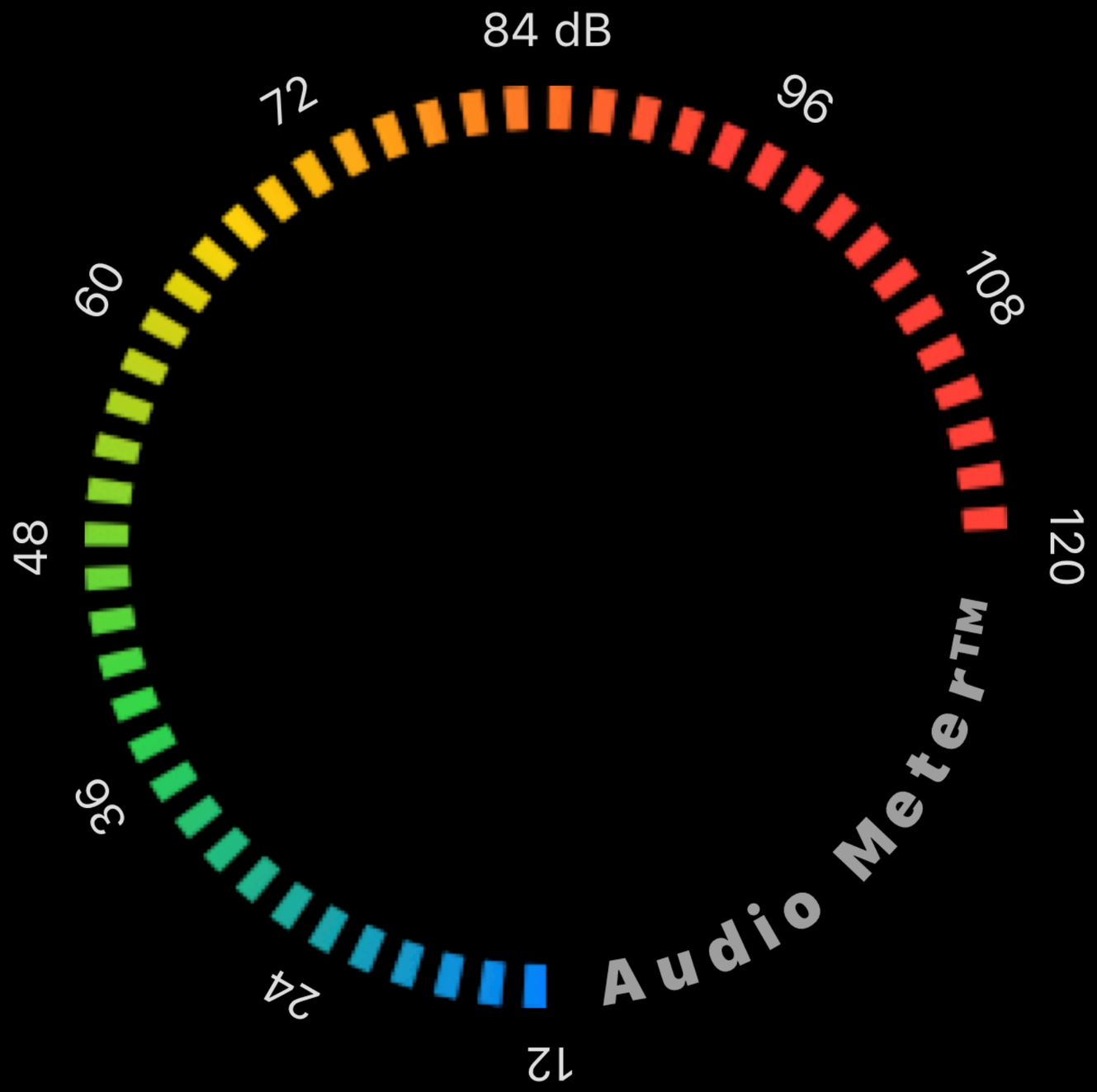

```
Text(character)
    .background(
        GeometryReader { proxy in
            Color.clear.preference(
                key: SizeKey.self,
                value: [proxy.size]
            )
        }
    )
```



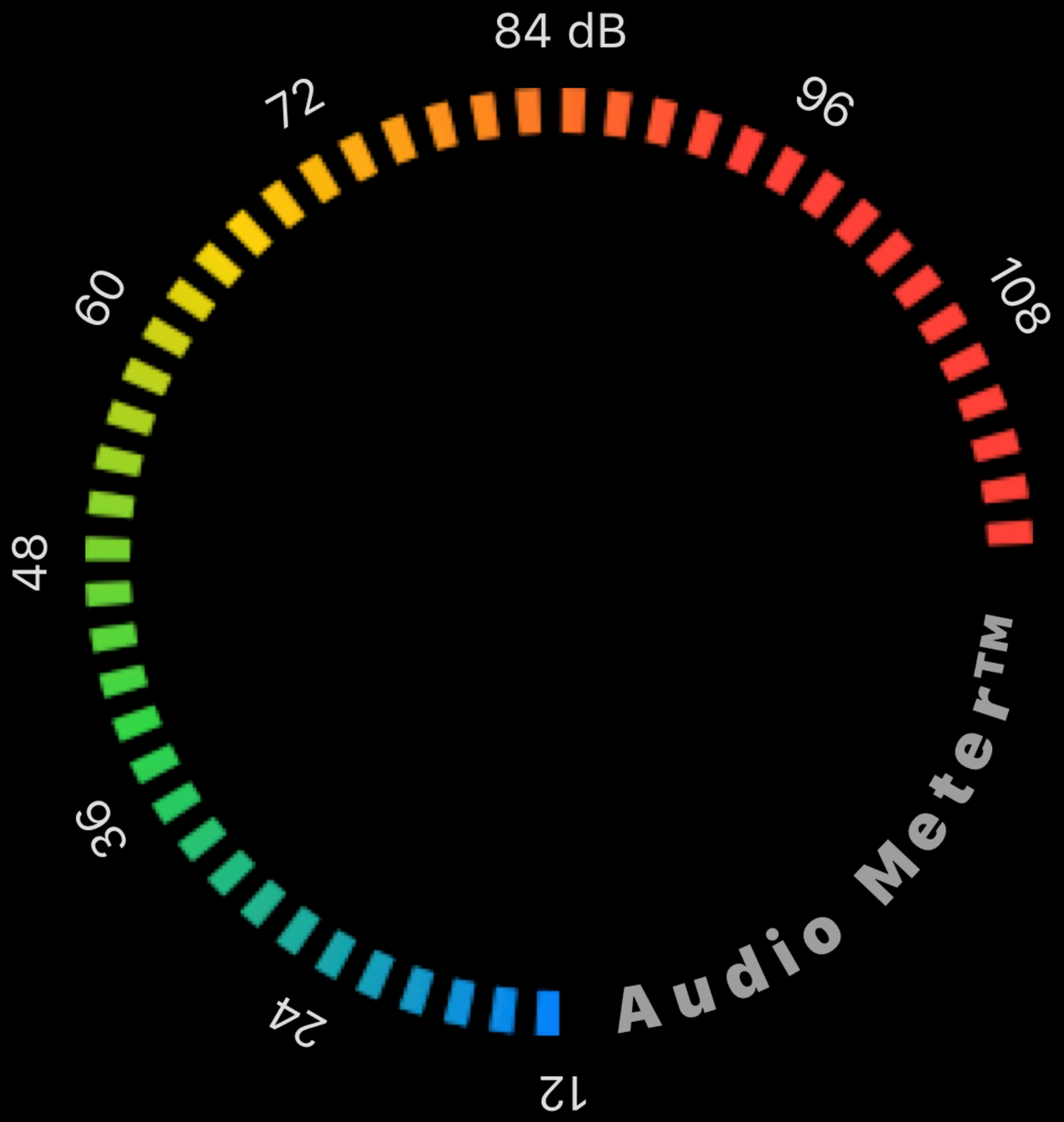

```
@State private var sizes: [CGSize] = []

ForEach(characters) {
    Text($0)
        .border(.randomColor)
}
.onPreferenceChange(SizeKey.self) {
    self.sizes = $0
}
```



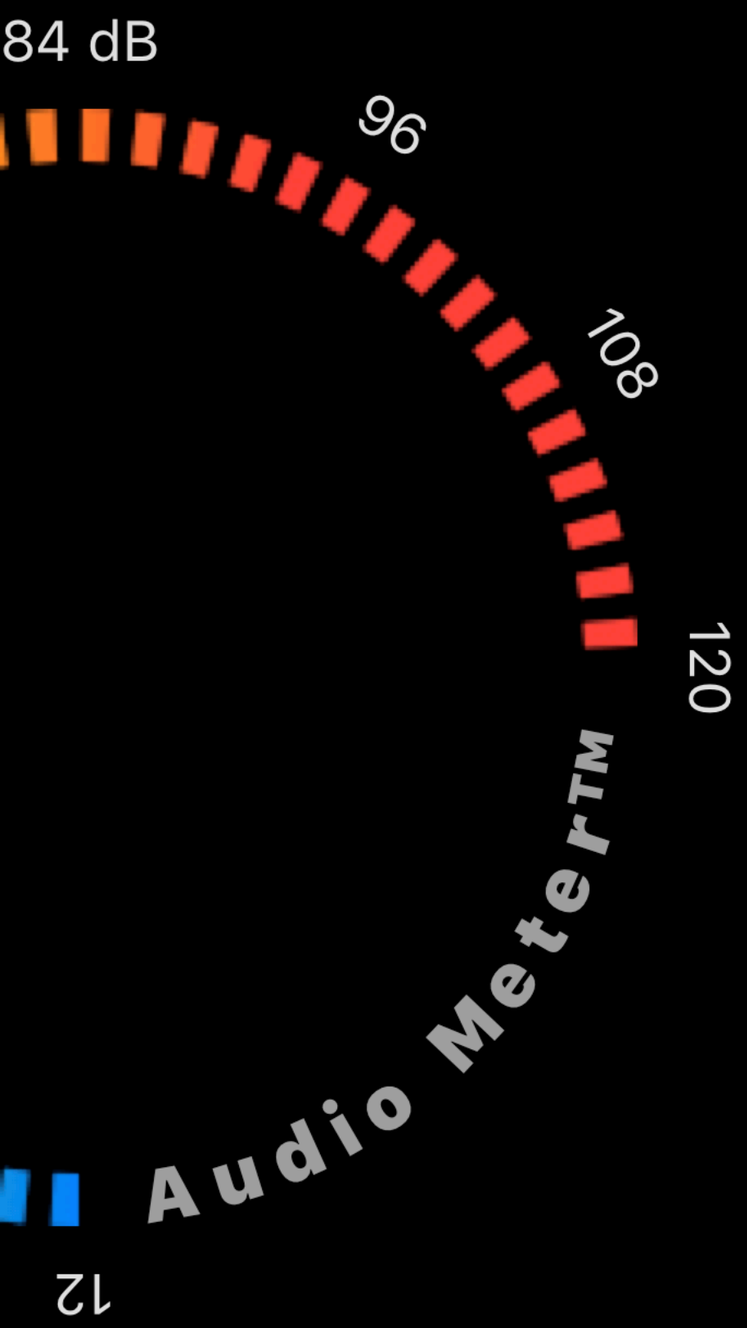

```
@State private var sizes: [CGSize] = []

ForEach(characters) {
    Text($0)
        .border(.randomColor)
        .frame(maxHeight: infinity,
               alignment: .bottom)
        .rotationEffect(...)
}
.onPreferenceChange(SizeKey.self) {
    self.sizes = $0
}
```



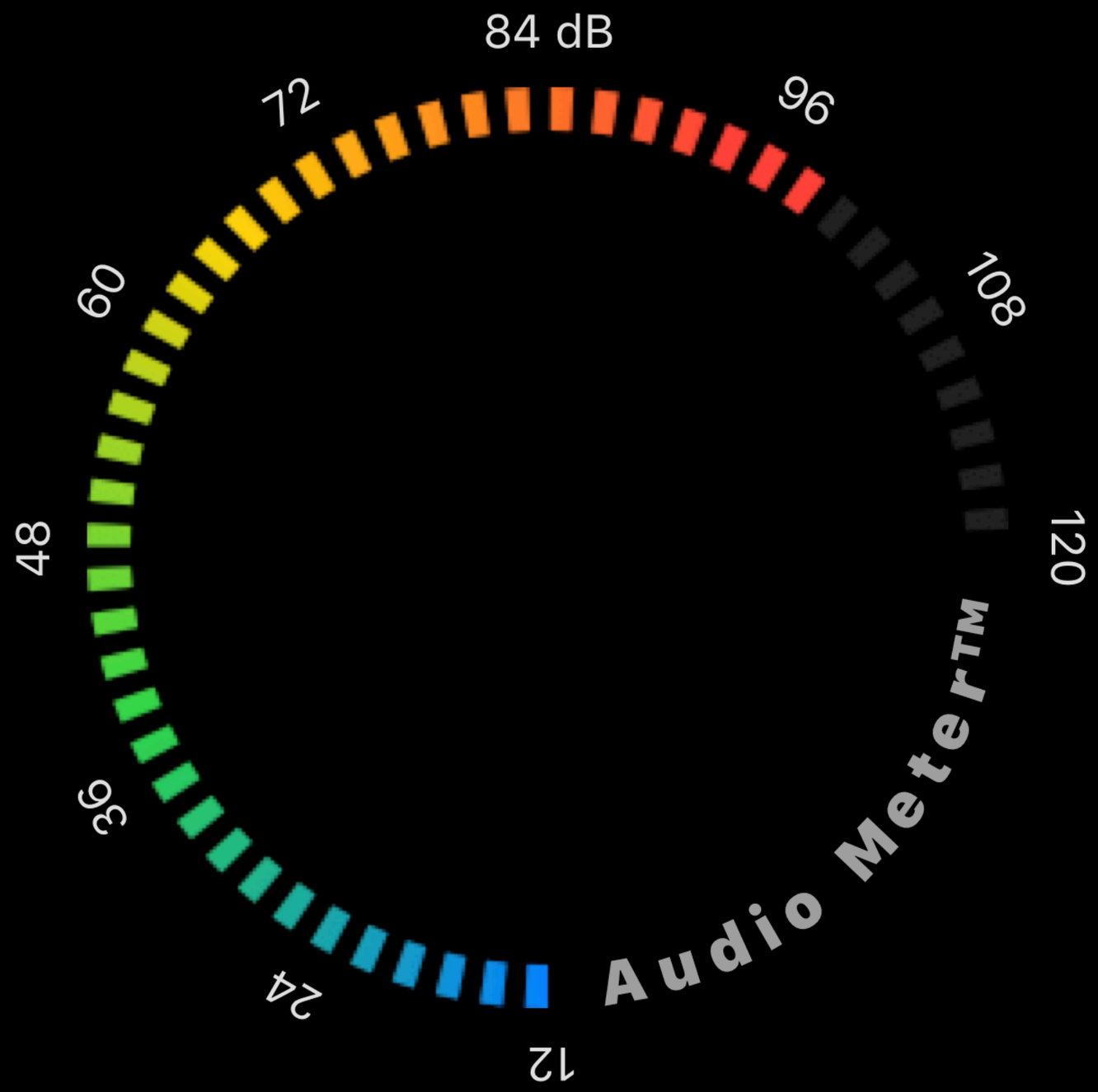

```
Text($0)
    .frame(maxHeight: infinity,
           alignment: .bottom)
    .rotationEffect(...)
}
```



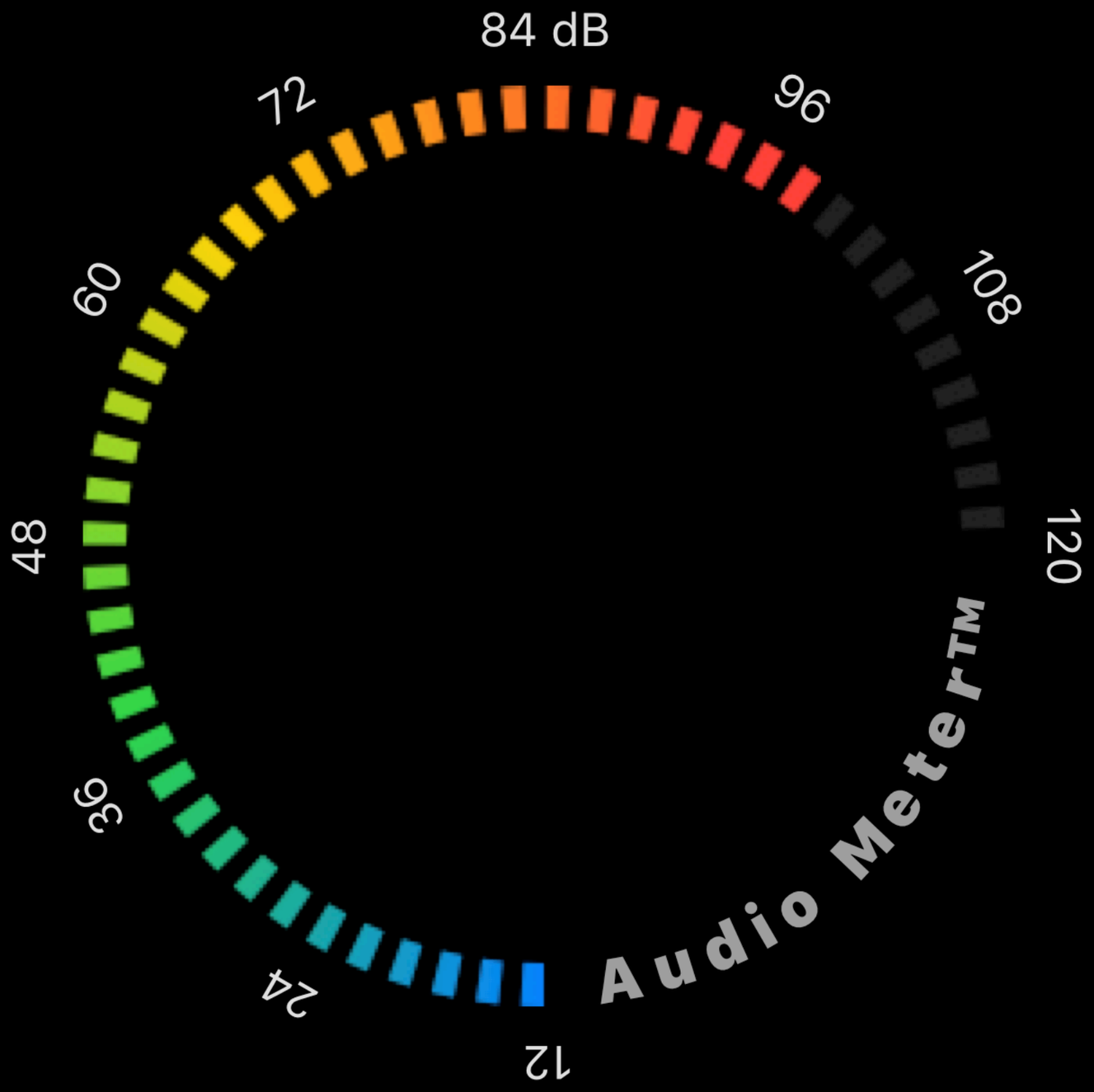

```
Float.random(in: 12...120)
```

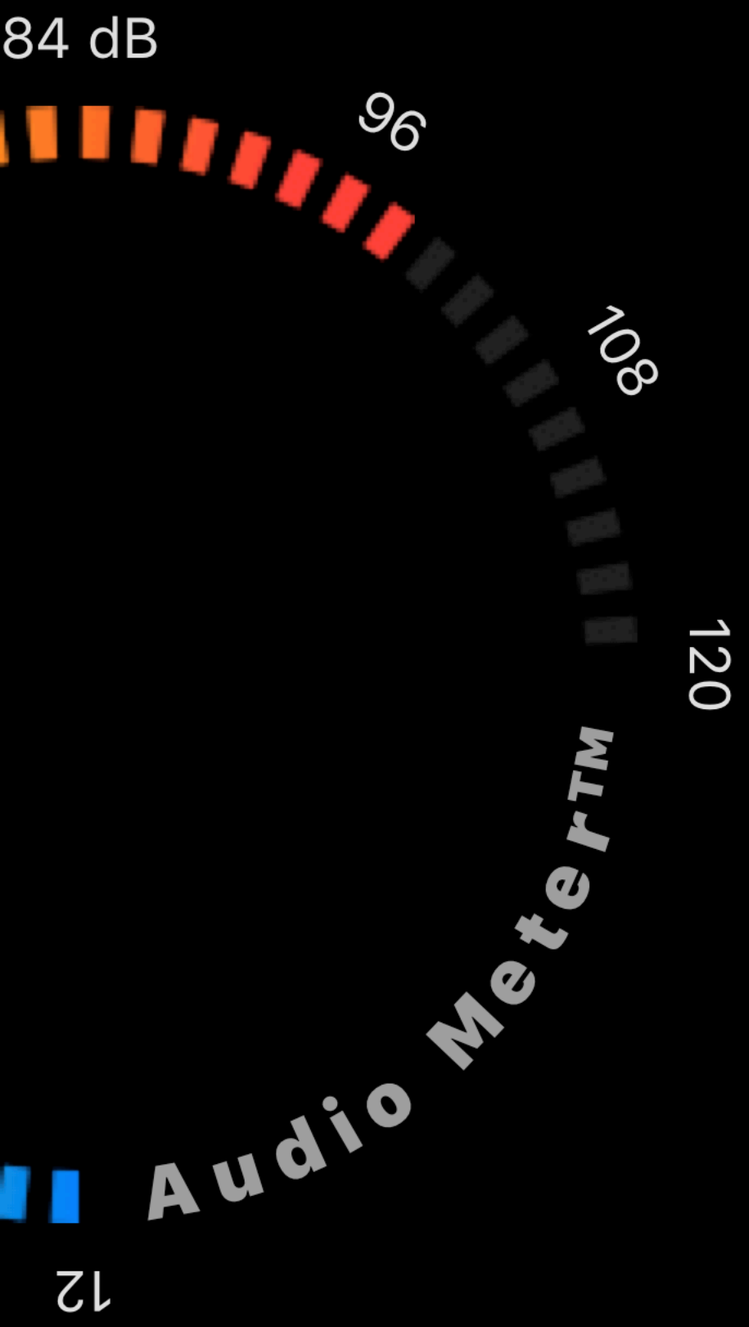



```
class Audio: ObservableObject {
    @Published var decibels: Float = 12

    init() {
        Timer.scheduledTimer(withTimeInterval: 1/30,
                             repeats: true) { _ in
            self.decibels = .random(in: 12...120)
        }
    }
}
```

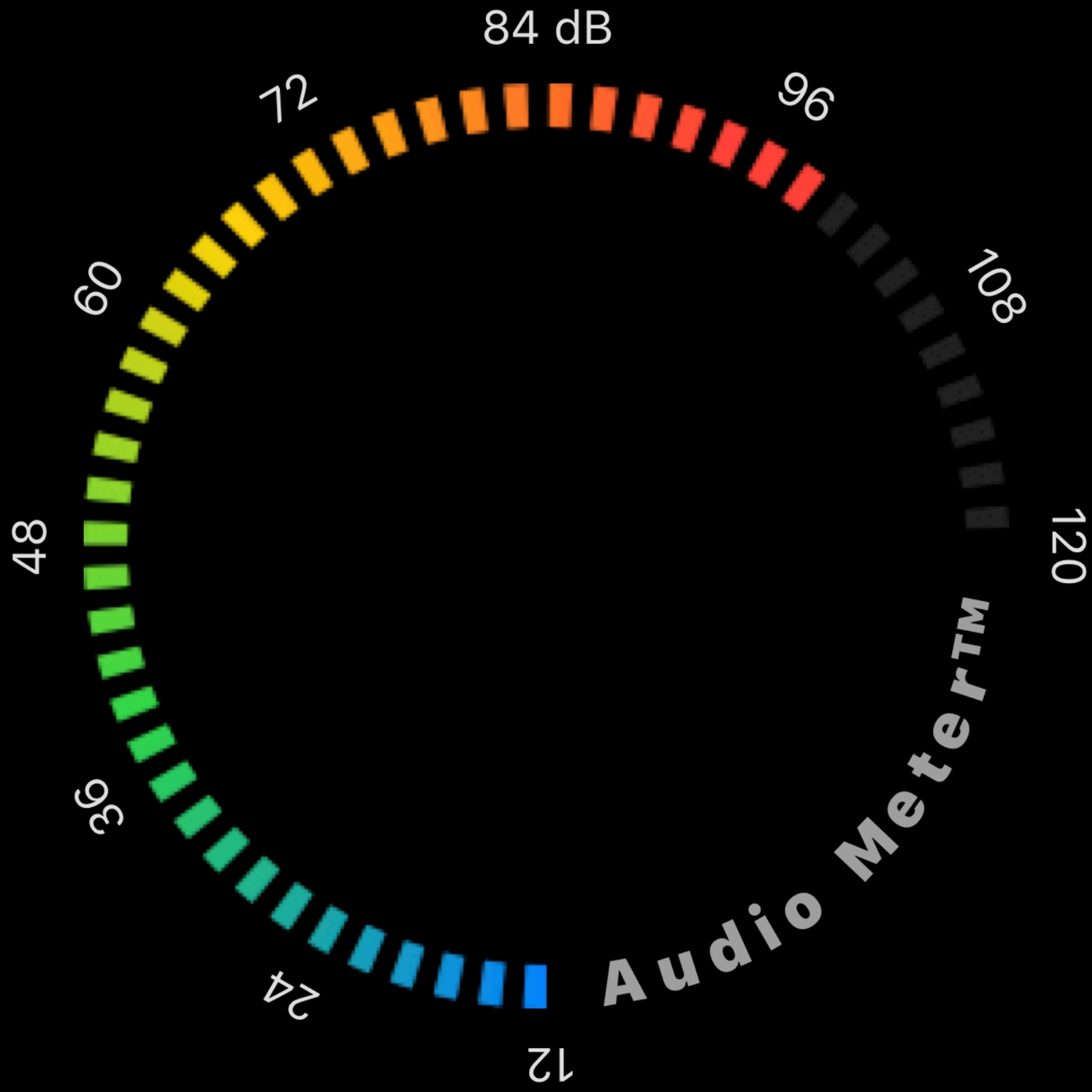


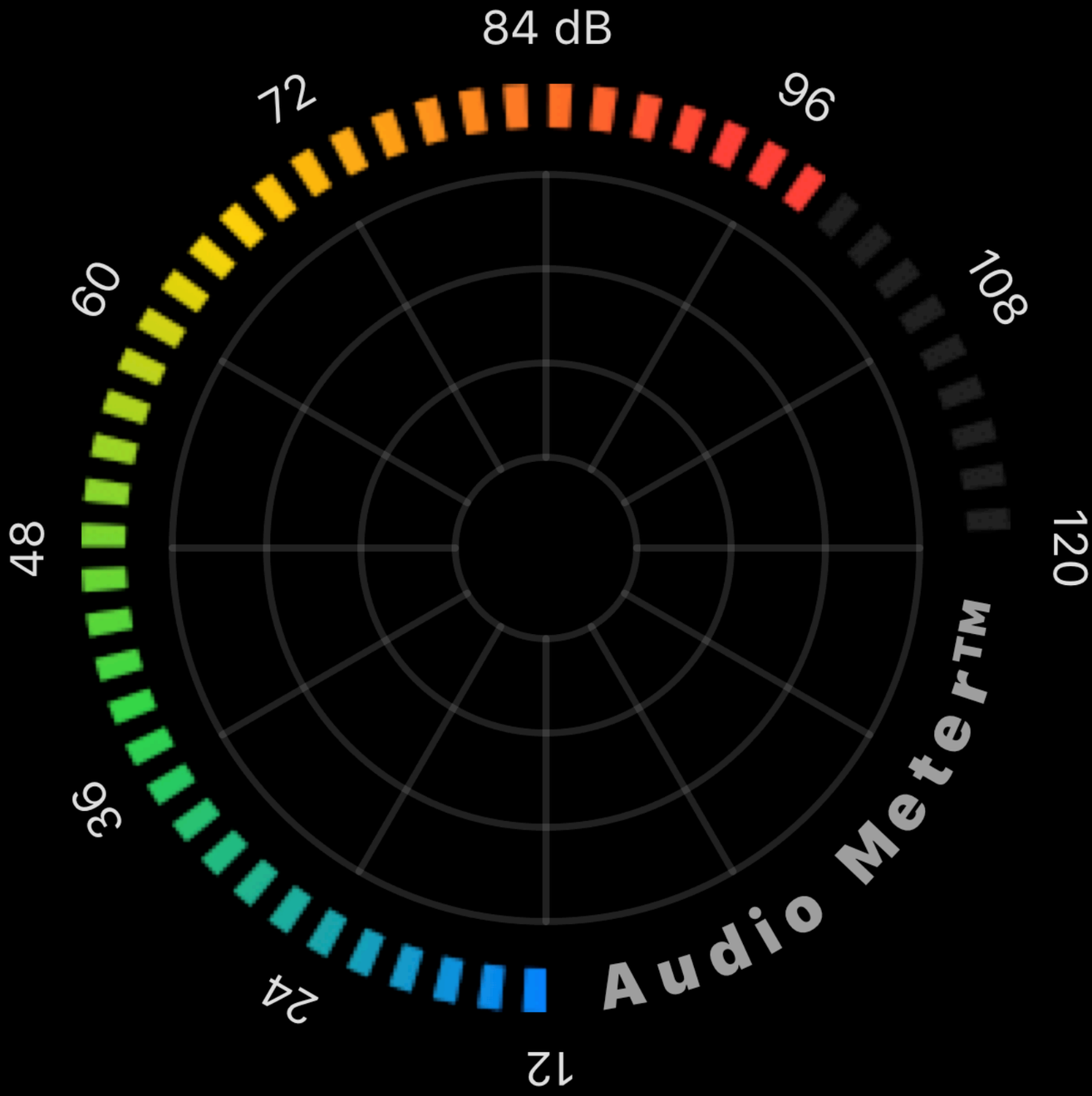

```
@ObservedObject audio = Audio()

var trimEnd: CGFloat {
    (audio.decibels - 12) / 108
}

var body: some View {
    Gauge(trim: 0..<trimEnd * 0.75)
}
```

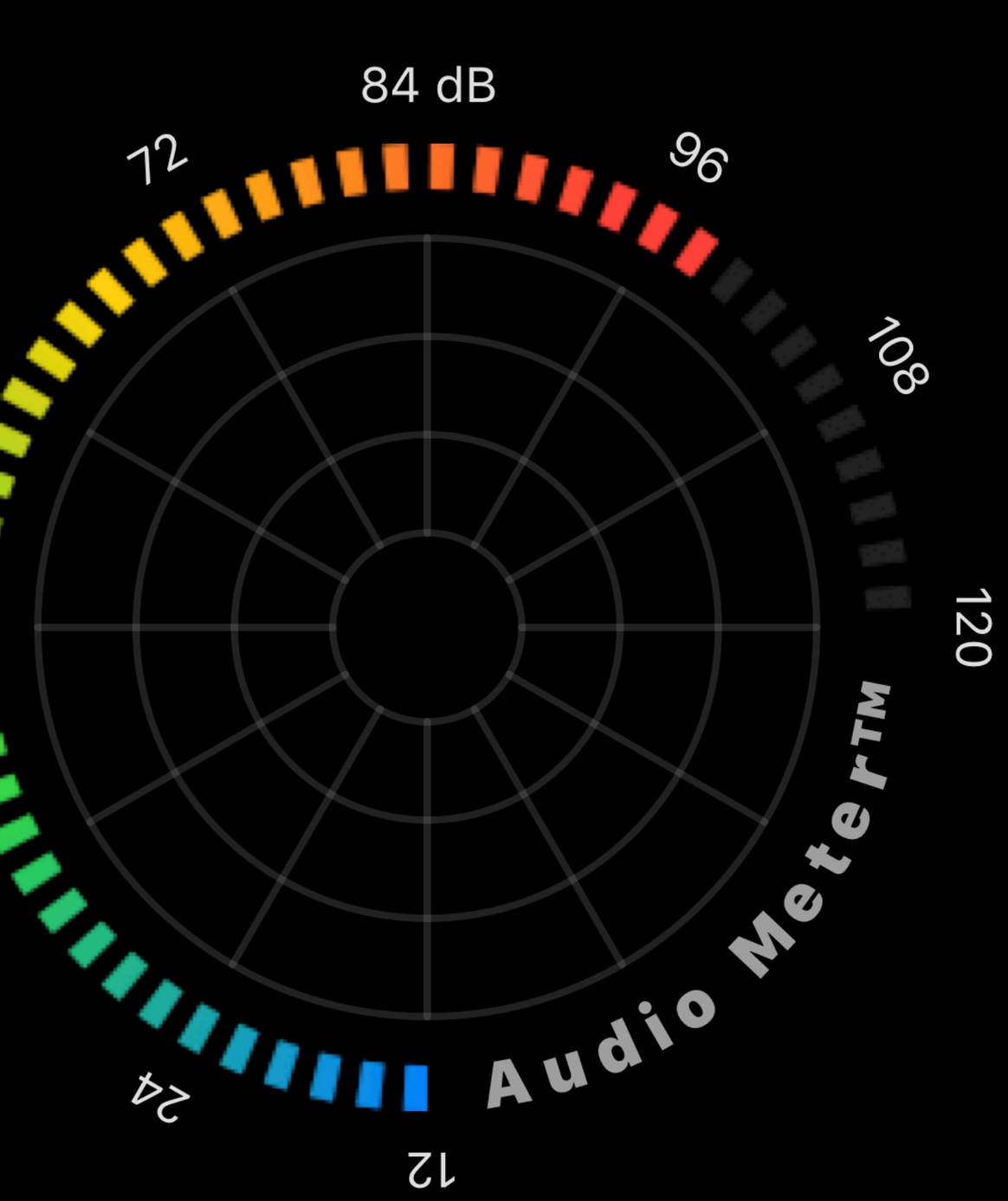

84 dB
96
108
120
12
24
36
48
60
72
Audio MeterTM



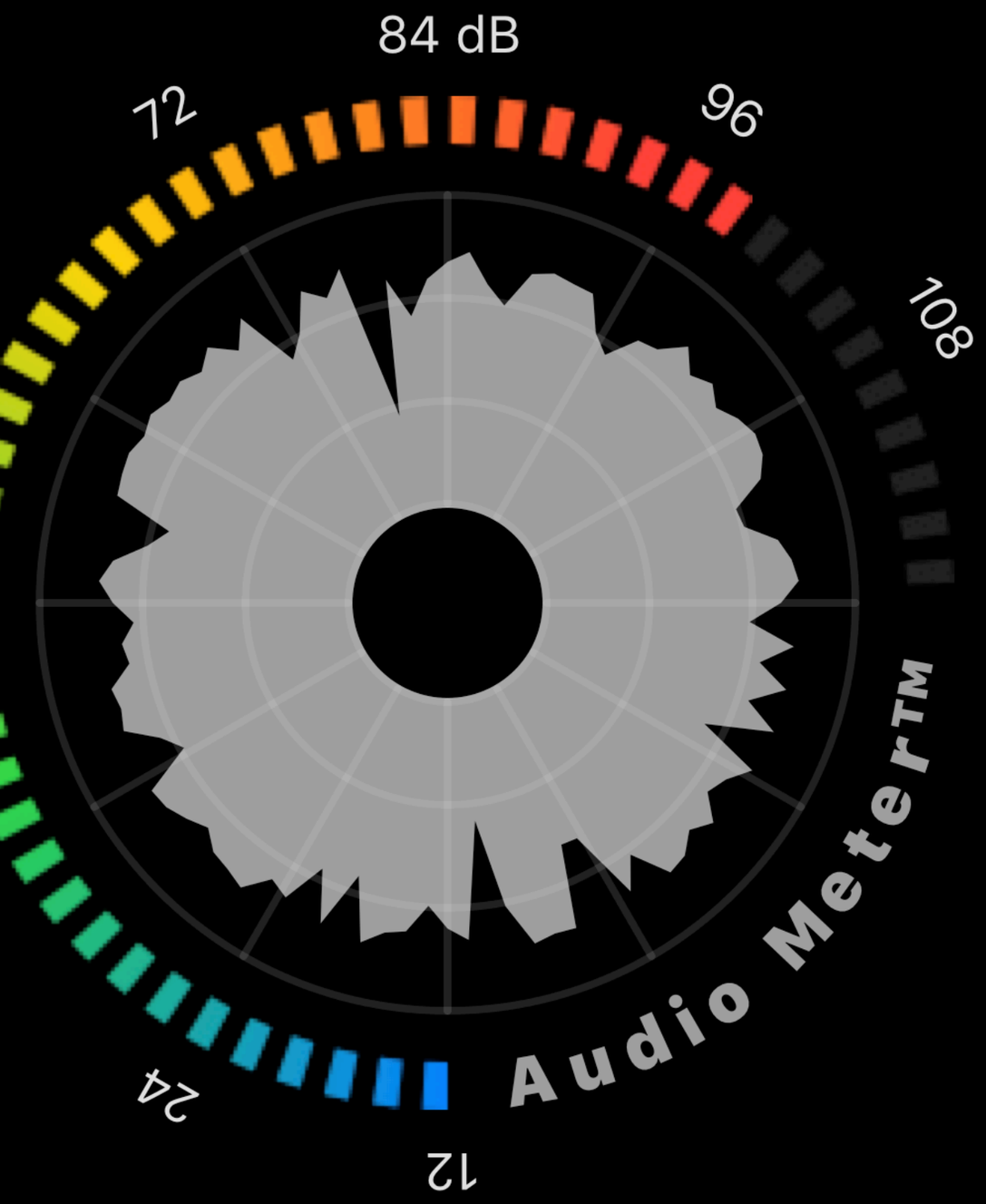

```swift
import AVFoundation
import Accelerate

let engine = AVAudioEngine()
try engine.start()
let format = node.inputFormat(forBus: 0)
engine.installTap(onBus: 0, buffer: 1024, format: values) {
    let values = Array(
        UnsafeBufferPointer(start: $0.floatChannelData![0],
                            count: Int($0.frameLength))
    )
    let rms = vDSP.rootMeanSquare(values)
    let decibels = 20 * log10(rms / /* Reference */)
    DispatchQueue.main.async {
        self.decibels = decibels
    }
}
```


84 dB
96
108
120
12
24
36
48
60
72
Audio Meter™

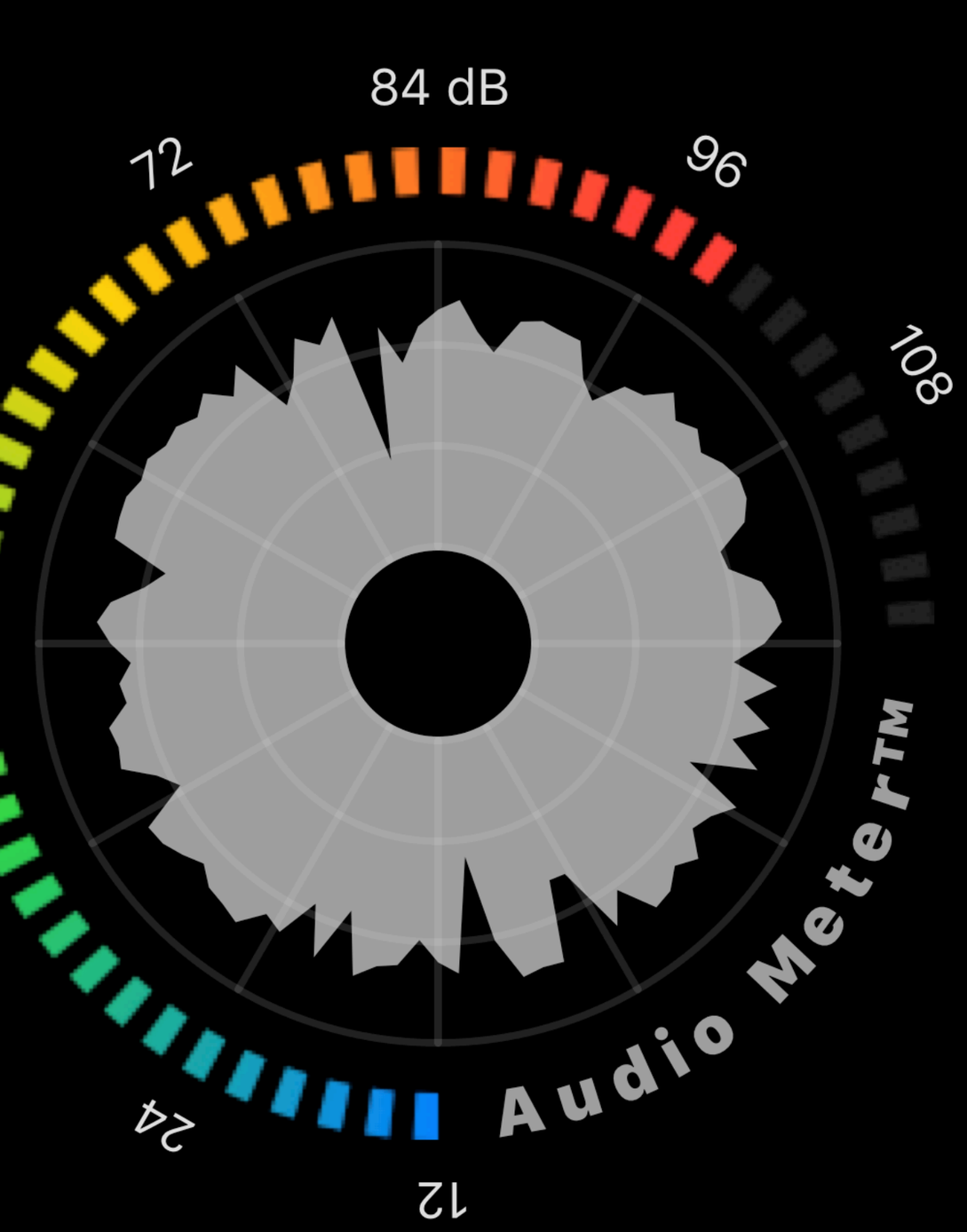

84 dB
72
96
60
108
48
120
36
24
12
Audio Meter™



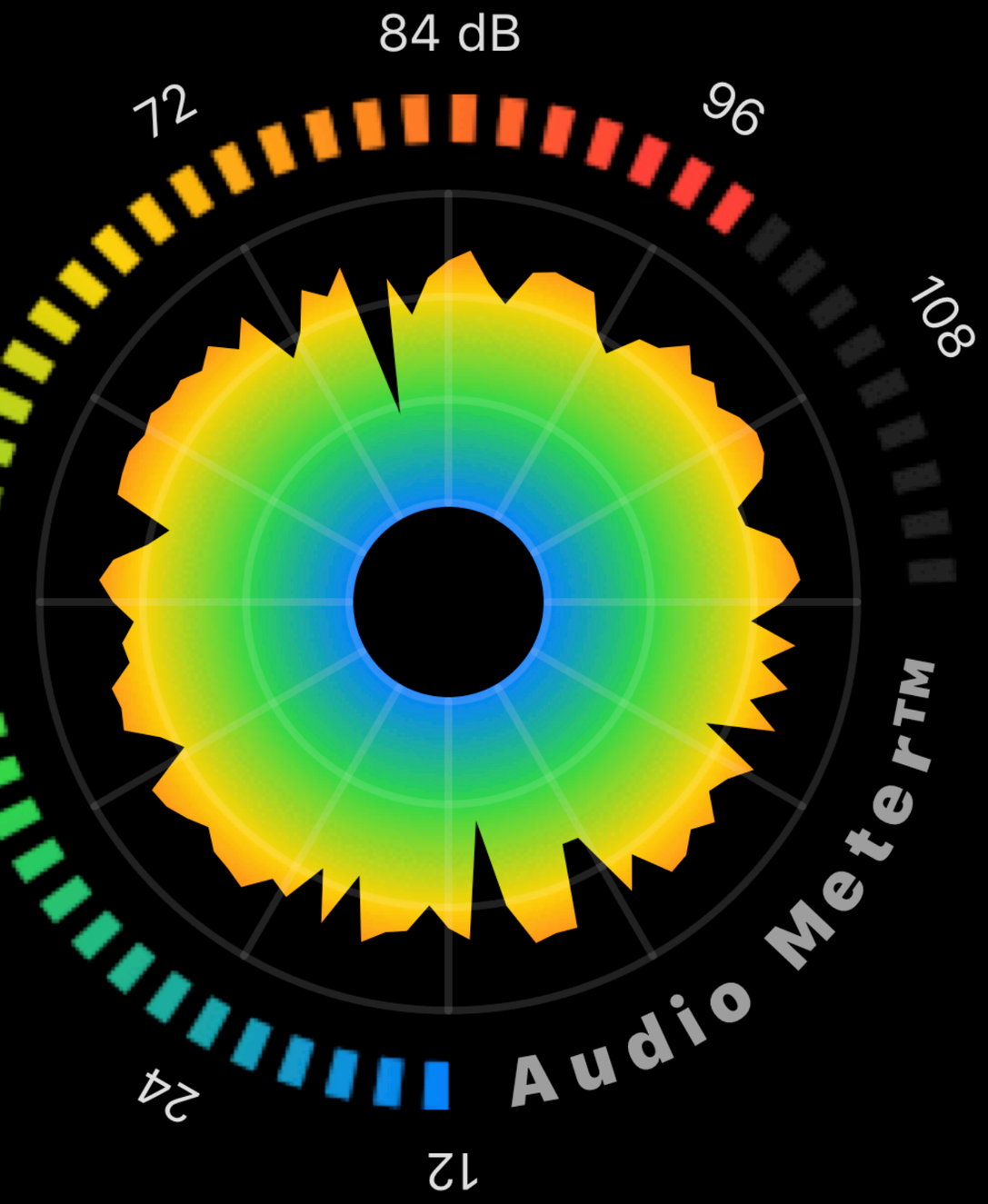

```
struct Graph: Shape {
    var values: [(Int, CGFloat)]

    func path(in: CGRect) -> Path {
        ...
    }
}
```



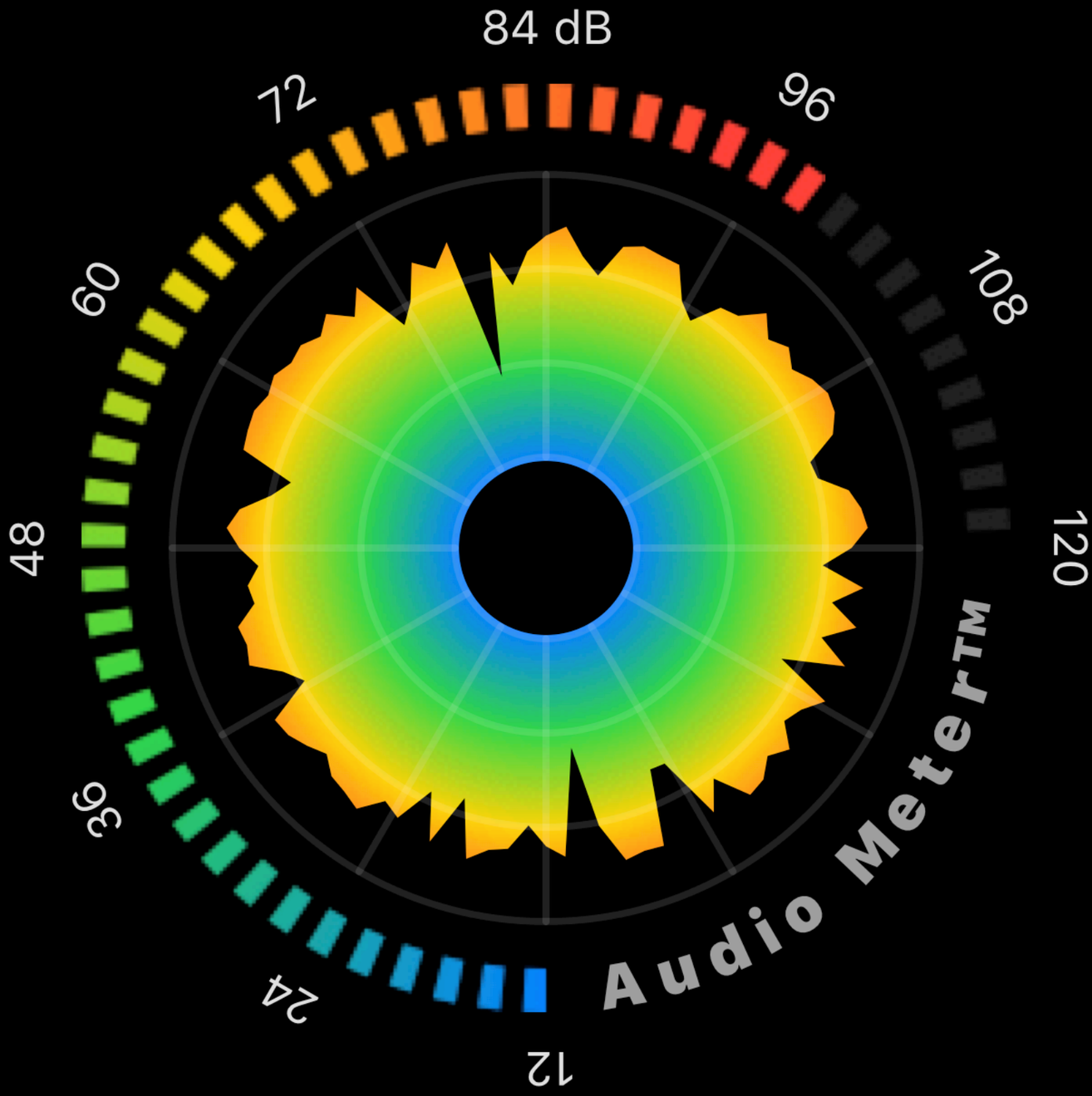

```
Path { path in
    let r = max(rect.width, rect.height) / 2
    let c = CGPoint(x: rect.midX,
                    y: rect.midY)
    for (index, value) in values {
        if index == 0 {
            path.move(to: /* Point */)
        } else {
            path.addLine(to: /* Point */)
        }
    }
    path.closeSubpath()
}
```

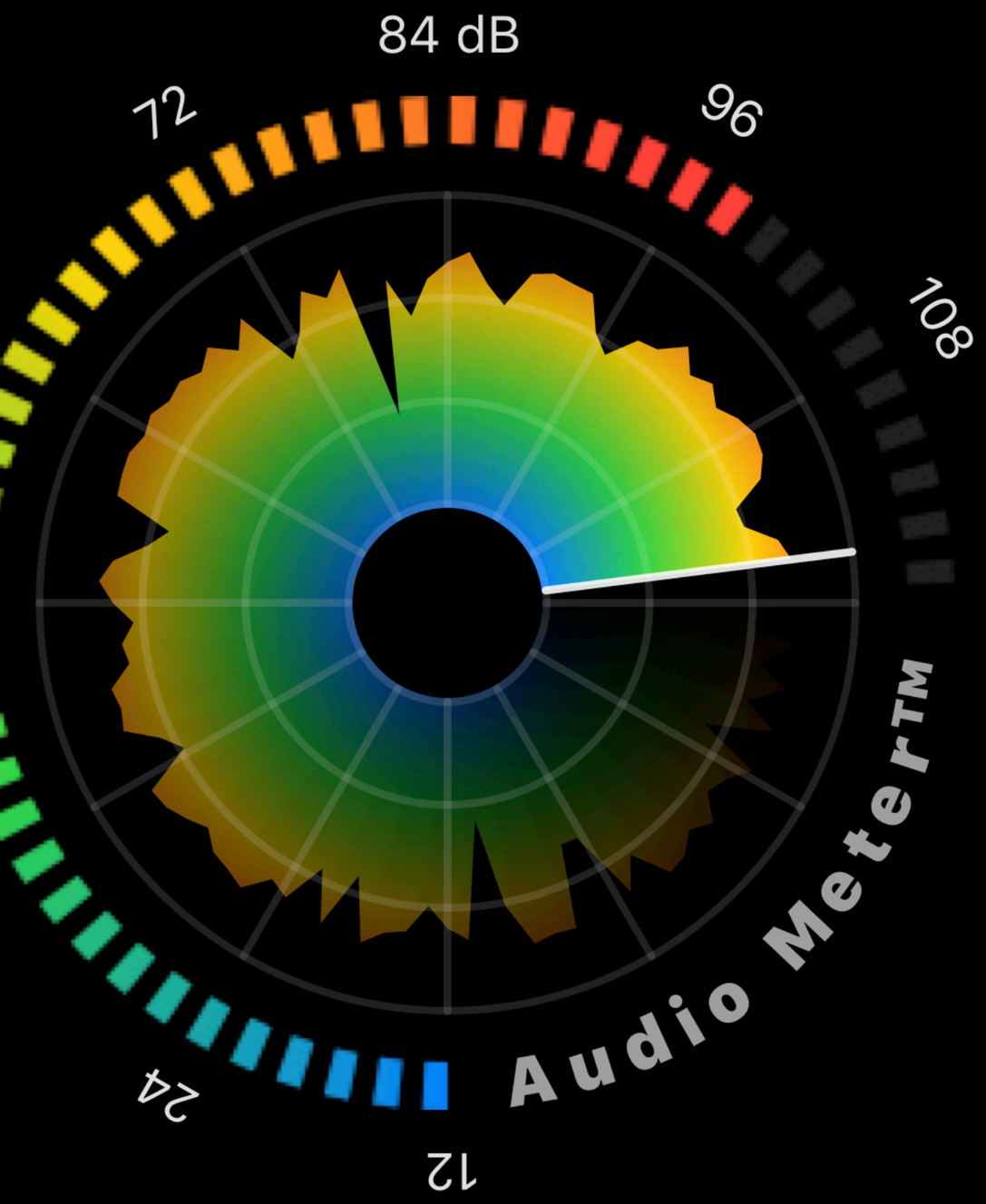



```
Path { path in
    let r = max(rect.width, rect.height) / 2
    let c = CGPoint(x: rect.midX,
                    y: rect.midY)
    for (index, value) in values {
        if index == 0 {
            path.move(to: /* Point */)
        } else {
            path.addLine(to: /* Point */)
        }
    }
    path.closeSubpath()
}
```

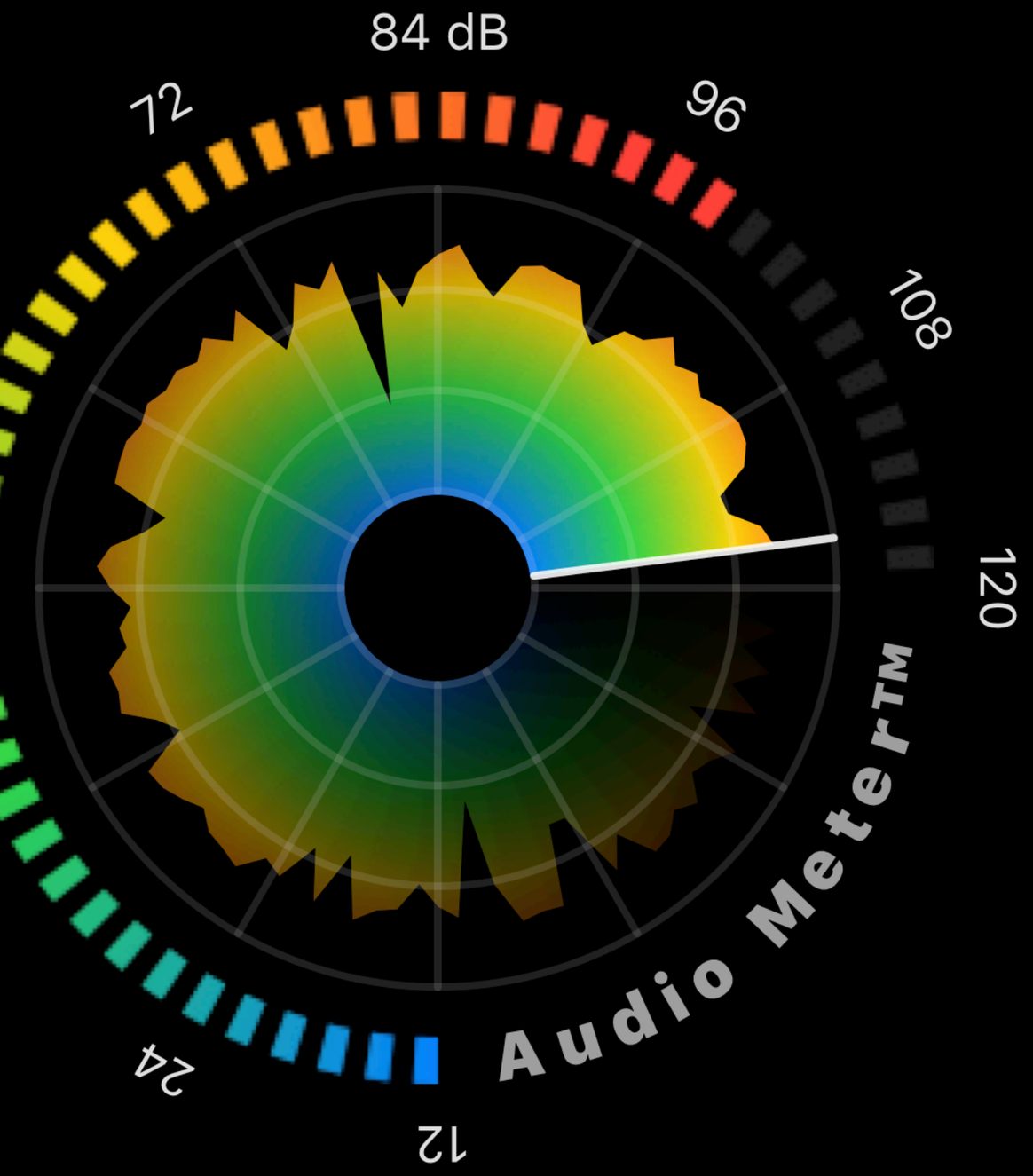



```
Graph(values: ...)
    .maskContent(using:
        RadialGradient(
            gradient: ...,
            center: .center,
            startRadius: 50,
            endRadius: 300)
    )
```

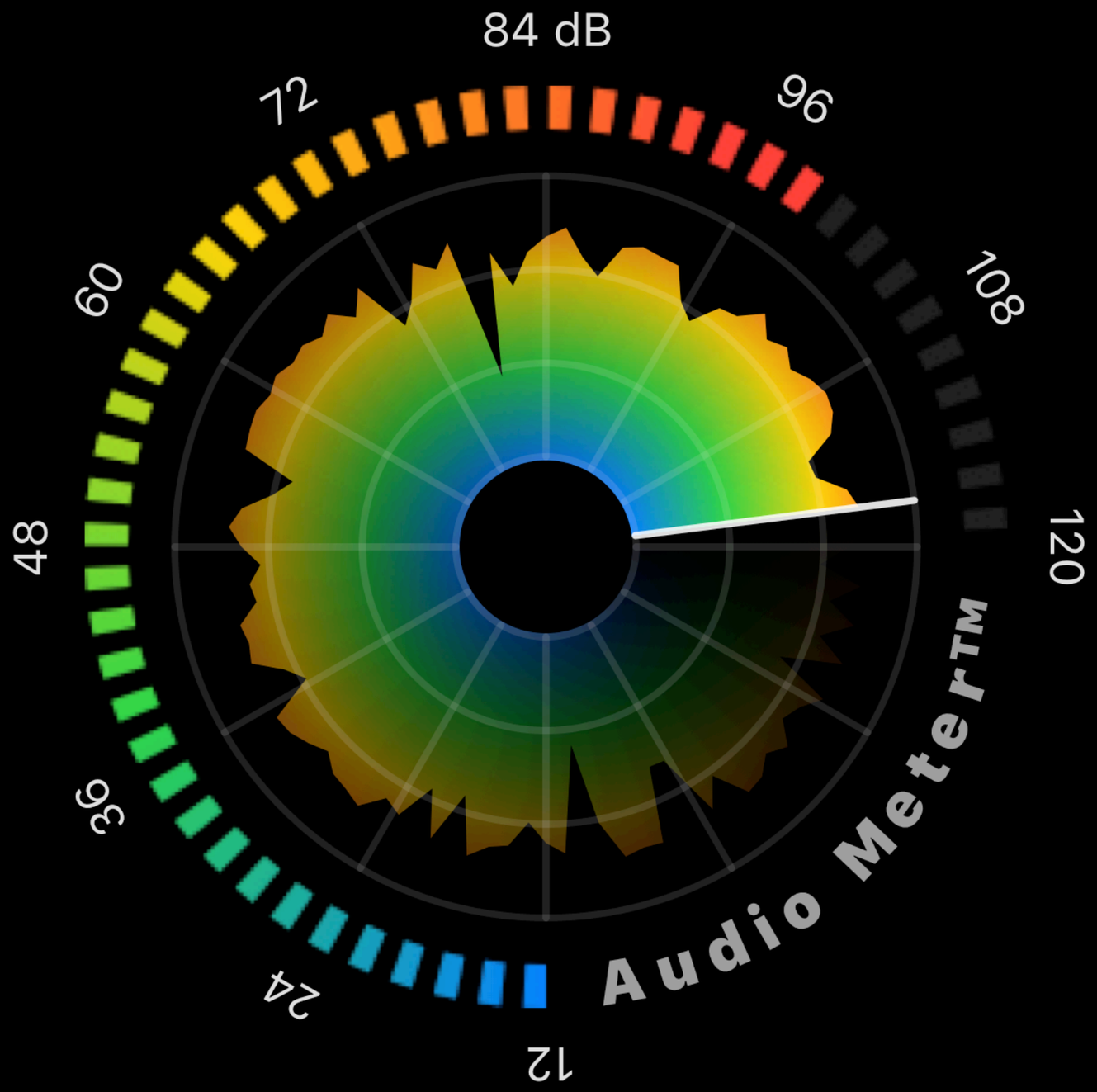

84 dB
72
96
60
108
48
120
36
24
12
Audio Meter™




```
ZStack {
    Graph()
        .maskContent(using:
            fadeGradient()
                .rotationEffect(/* Angle */)
        )

    ScannerLine()
        .rotationEffect(/* Angle */)
}
```

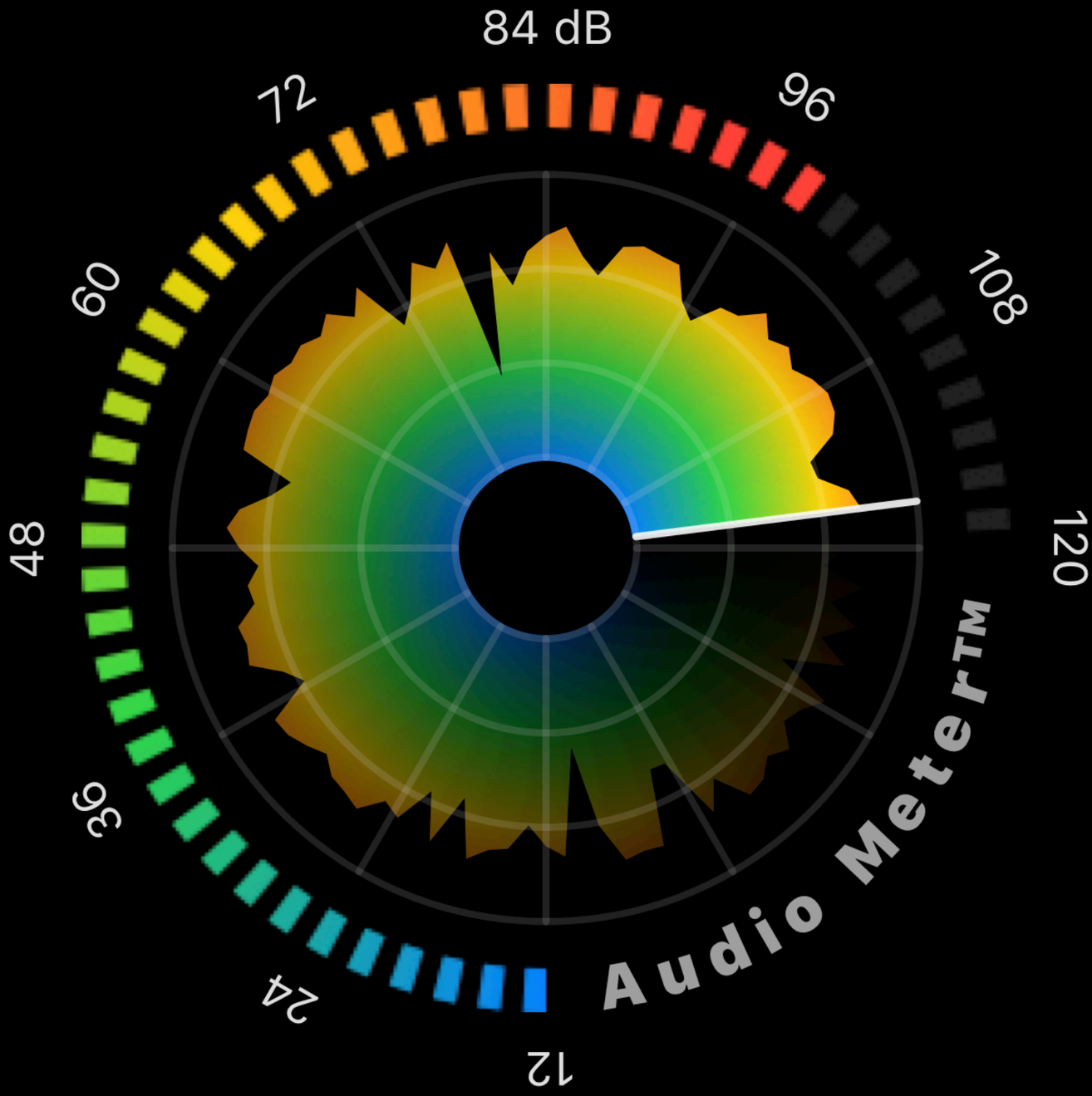



```
ZStack {
    Graph()
        .maskContent(using:
            fadeGradient()
                .rotationEffect(/* Angle */)
        )

    ScannerLine()
        .rotationEffect(/* Angle */)
}
.animation(
    .linear(duration:
        audio.sampleDuration)
    )
)
```




```
@State var timing: Double = 0

Gauge()
    .animation(.easeOut($timing))
```

Timing: 0.0


84 dB
72
96
60
108
48
120
36
24
12
Audio Meter™

Swift with Majid	by	Majid Jabrayilov
SwiftUI for Mac	by	Sarah Reichelt
Swift Talk	by	Florian Kugler, Chris Eidhof
Blog	by	Pavel Zak
SwiftUI Lab	by	Javier
Hacking with Swift	by	Paul Hudson
Splash	by	John Sundell
Blog	by	Ole Begemann

Prototyping Custom UI in SwiftUI

Tobias
Due
Munk

dotSwift

February 3
2020