Sample Code

Adding a depth effect to text in visionOS

Create text that expands out of a window using stacked SwiftUI text views.

Download

visionOS 2.0+ | Xcode 16.0+

Overview

This sample app demonstrates how to stack multiple text views so the text appears to pop out of its window. At launch, the app applies a spring animation to the views in the stack, causing the text to pop out, like the scene in the following image:

Play

Initialize the text

The app's `DepthTextView` creates and stores a `Text` instance as one of its properties:

```
struct DepthTextView: View {
    let text = Text("Hello World").font(.extraLargeTitle)

    // ...
}
```

By passing `extraLargeTitle` to the `font(_:)` modifier, it makes the text appear bigger. The main view creates text variations from this property in its methods and computed properties.

Create the foremost text view

The `textFrontView` property creates a variation of the `text` property, to act as the view that pops out the most, by changing its offset along the z-axis:

```
var textFrontView: some View {
    text.offset(z: Double(layerSpacing * layers) * animationProgress)
}
```

The sample achieves this by passing the largest value into the `offset(z:)` modifier. The offset value, excluding the `animationProgress` value that controls the animation, is the product of the `layerSpacing` and `layers` properties:

```
/// The number of text layers that extend from the window along its z-axis.
let layers = 5

/// The number of points between each layer.
let layerSpacing = 100
```

Generate a series of text layers along the z-axis

The `textMiddleViews` property makes several variations of the `text` property to form the middle layers between the shadow and the front of the stack. It does this by creating a `ForEach` instance to generate the text views:

```
var textMiddleViews: some View {
    ForEach(1..<layers, id: \.self) { layer in
        let layerPercent = Double(layer) / Double(layers)
        let maximumOffset = Double(layerSpacing * layers)
        let maximumOpacity = 1.0
```

```
        text
            .offset(z: maximumOffset * layerPercent * animationProgress)
            .opacity(maximumOpacity * layerPercent)
    }
}
```

The property dynamically applies a unique offset along the z-axis to each text view by passing a larger value to the `offset(z:)` modifier with each iteration. This offset creates a stacking effect that shows the text extending outward from the view window.

The `layers` property controls the number of iterations:

```
let layers = 5
```

Generate a shadow version of the text view

The `textShadowView` property makes a version of the `text` property that acts as a shadow by adding a blur effect and modifying its style and transparency:

```
var textShadowView: some View {
    /// The width, in points, of the blur effect relative to the text's edges.
    let blurRadius: CGFloat = 12
    let maximumOpacity = 0.6

    return text
        .foregroundStyle(.black)
        .blur(radius: blurRadius)
        .opacity(maximumOpacity * animationProgress)
}
```

The `blur(radius:opaque:)` view modifier applies a Gaussian blur to the text view, and the `opacity(_:)` view modifier makes the view semitransparent.

> **Note**
>
> This property doesn't change the offset along the z-axis from `0.0`, which places the shadow view in the same plane as the window that contains the `ZStack`.

Stack the text with animations

The `DepthTextView` arranges its view properties along its z-axis by adding a `ZStack` as the main view of its `body` property:

```
var body: some View {
    // Create a stack of the same text view with different opacities and
    // positions along the z-axes, starting with a blurry shadow version.
    ZStack {
        textShadowView
        textMiddleViews
        textFrontView
    }
    .onAppear(perform: animateWithSpringEffect)
}
```

The `textShadowView`, `textMiddleViews`, and `textFrontView` properties each create a version of the text. The sample adds an animation effect to its `ZStack` by passing its `animateWithSpringEffect` method to the stack's `onAppear(perform:)` view modifier.

The `animateWithSpringEffect` method animates the transition through the `animationProgress` property from `0.0` to `1.0` by:

1. Creating a `Spring` instance.
2. Creating an `interpolatingSpring` animation with that spring.
3. Passing the spring animation to the `withAnimation(_:_:)` method.

```
func animateWithSpringEffect() {
    /// A spring coefficient for a spring animation effect,
    /// in newtons per meter.
    let stiffness: Double = 200

    /// The damping factor of a spring animation's effect,
    /// in newton-seconds per meter.
    let damping: Double = 10

    let spring = Spring(stiffness: stiffness, damping: damping)
    let springAnimation = Animation.interpolatingSpring(spring).delay(1.0)

    // Animate the text popping out of the window with a spring effect.
    withAnimation(springAnimation) { animationProgress = 1.0 }
}
```

The `animateWithSpringEffect()` method creates a spring animation with `stiffness` and `damping` properties, as well as a one-second delay. It animates the `animationProgress` from 0.0 to 1.0, resulting in a springlike pop effect as the text enters the view.

See Also

Drawing text

{} Displaying text in visionOS

Create styled text in a window using SwiftUI.