Sample Code

Creating 2D shapes with SwiftUI

Draw two-dimensional shapes in your visionOS app with SwiftUI shapes or with your custom shapes.

Download

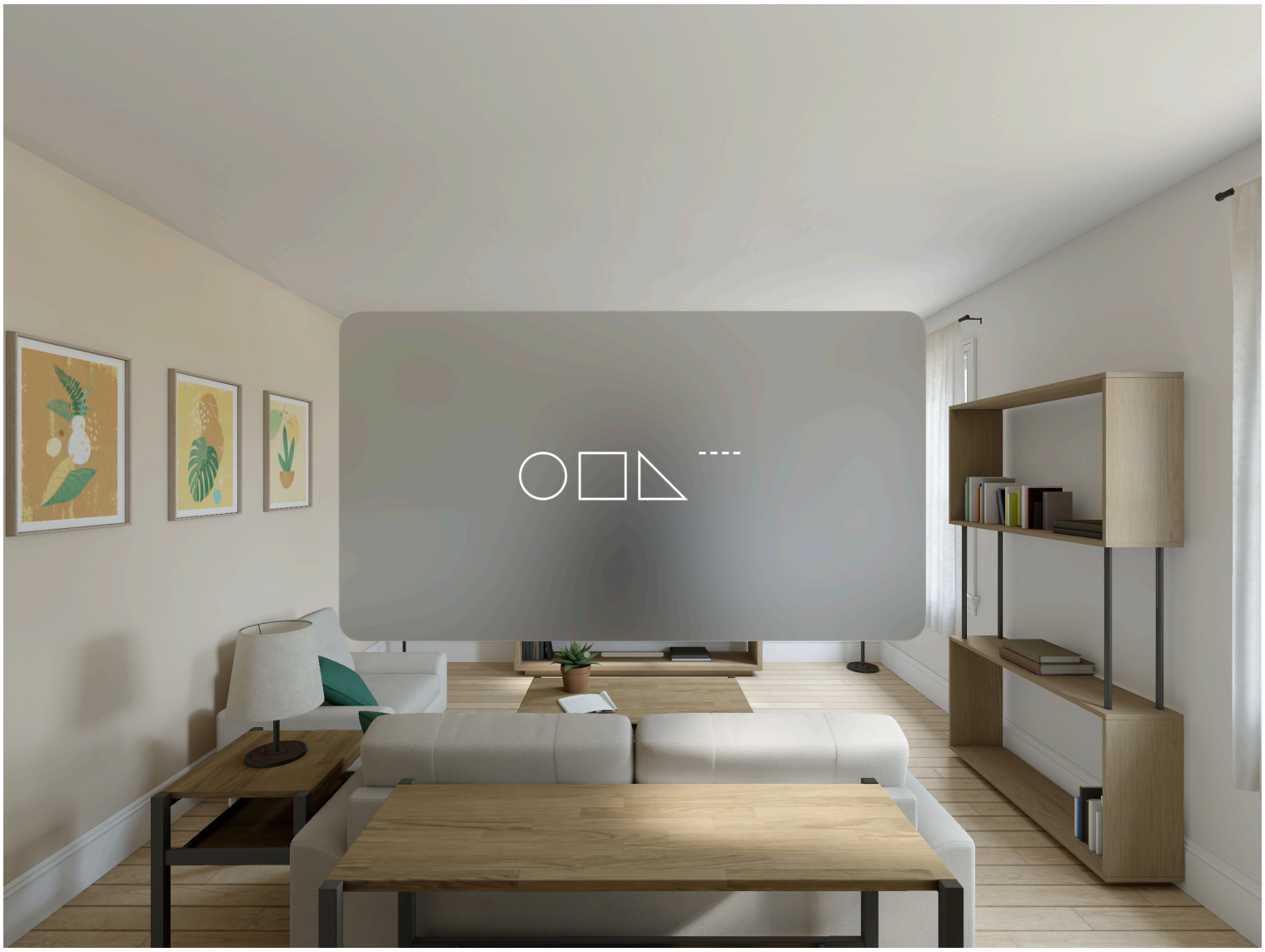
visionOS 2.0+ | Xcode 16.0+

Overview

This sample code project demonstrates how to create and display 2D shapes in your visionOS app, using SwiftUI views that adopt the Shape protocol, including your custom types.

The following image shows the main view of the app, which draws four shapes:

- A circle
- A rectangle
- A triangle
- A dashed line



SwiftUI defines the `Circle` and `Rectangle` types, and the sample defines the custom types `Triangle` and `Line`.

Add existing shapes from SwiftUI to a view

The app's main view draws the outline of a circle and a rectangle by adding an instance of `Circle` and `Rectangle`, respectively.

```
struct ShapesView: View {
    var body: some View {
        /// The gap between each shape.
        let spaceBetweenShapes: CGFloat = 30.0

        /// The width and height for the frame of each shape view.
        let shapeScale: CGFloat = 100.0

        /// A width for all the lines in each shape.
        let strokeWidth: CGFloat = 5.0
```

```
        // ...

        HStack(spacing: spaceBetweenShapes) {
            Circle()
                .stroke(lineWidth: strokeWidth)
                .frame(width: shapeScale, height: shapeScale)
            Rectangle()
                .stroke(lineWidth: strokeWidth)
                .frame(width: shapeScale, height: shapeScale)

            // ...
        }
    }
}
```

The `stroke(_:lineWidth:)` method sets the thickness of the line that the shape uses to draw itself, and the `frame(width:height:alignment:)` method puts the shape inside an invisible view.

Create custom shape types

This sample defines custom shape types for lines and triangles that draw themselves by adopting the `Shape` protocol and implementing a custom `Path` method.

The `Line` type stores two endpoints and draws a line between them in its `path(in:)` method.

```
import SwiftUI

struct Line: Shape {
    let endPoint1: CGPoint
    let endPoint2: CGPoint

    // ...

    func path(in bounds: CGRect) -> Path {
        /// The drawing path for the triangle shape.
        var path = Path()

        // Draw the line between the two endpoints.
        path.move(to: endPoint1)
        path.addLine(to: endPoint2)
```

```
        return path
    }
```

The `Triangle` type stores three vertices for a triangle and draws a line between each vertex in its custom `path(in:)` method.

```
import SwiftUI

struct Triangle: Shape {
    let vertex1: CGPoint
    let vertex2: CGPoint
    let vertex3: CGPoint

    // ...

    func path(in bounds: CGRect) -> Path {
        /// The drawing path for the triangle shape.
        var path = Path()

        // Start at the first vertex.
        path.move(to: vertex1)

        // Draw the triangle's first two sides.
        path.addLine(to: vertex2)
        path.addLine(to: vertex3)

        // Draw the triangle's third side by returning to the first vertex.
        path.closeSubpath()

        return path
    }
}
```

The shapes view adds a triangle by creating three `CGPoint` instances and passing them to the `Triangle` initializer.

```
struct ShapesView: View {
    var body: some View {
        /// The gap between each shape.
        let spaceBetweenShapes: CGFloat = 30.0
```

```
        /// The width and height for the frame of each shape view.
        let shapeScale: CGFloat = 100.0

        /// A width for all the lines in each shape.
        let strokeWidth: CGFloat = 5.0

        /// The upper-leading corner of the triangle.
        let vertex1 = CGPoint(x: 0.0, y: 0.0)

        /// The lower-trailing corner of the triangle.
        let vertex2 = CGPoint(x: shapeScale, y: shapeScale)

        /// The lower-leading corner of the triangle.
        let vertex3 = CGPoint(x: 0.0, y: shapeScale)

        // ...

        HStack(spacing: spaceBetweenShapes) {
            // ...

            Triangle(vertex1, vertex2, vertex3)
                .stroke(lineWidth: strokeWidth)
                .frame(width: shapeScale, height: shapeScale)

            // ...
        }
    }
}
```

Each vertex uses SwiftUI coordinates. The origin of this coordinate system is in the upper-left corner, and the y-axis increases in a downward direction.

Add customizations to shape types

To draw a line with dashes, add a `dashed(_:)` method to the `Line` type, which the sample puts in an extension.

```
extension Line {
    func dashed(_ width: CGFloat,
                _ dashPattern: [CGFloat]? = nil) -> some Shape {
        let pattern = dashPattern ?? [width]
```

```
        let style = StrokeStyle(lineWidth: width, dash: pattern)

        return stroke(style: style)
    }
}
```

The method creates an instance of a line, applying a stroke style to it.

The shapes view calls this method to create a dashed line based on the stroke width for all the shapes in the view.

```
        /// The gap between each shape.
        let spaceBetweenShapes: CGFloat = 30.0

        /// The width and height for the frame of each shape view.
        let shapeScale: CGFloat = 100.0

        /// A width for all the lines in each shape.
        let strokeWidth: CGFloat = 5.0

        // ...

        /// A pattern for the dashed line.
        let strokePattern = [3 * strokeWidth, 2 * strokeWidth]

        HStack(spacing: spaceBetweenShapes) {
            // ...

            Line(shapeScale)
                .dashed(strokeWidth, strokePattern)
                .frame(width: shapeScale, height: shapeScale)
        }
    }
}
```

In this sample, the view draws each line segment one-and-a-half times longer than the gaps between adjacent line segments.

See Also

Related samples

{} Creating 3D entities with RealityKit

Display a horizontal row of three-dimensional shapes in your visionOS app, using predefined mesh and white material.