Sample Code

Displaying a stereoscopic image

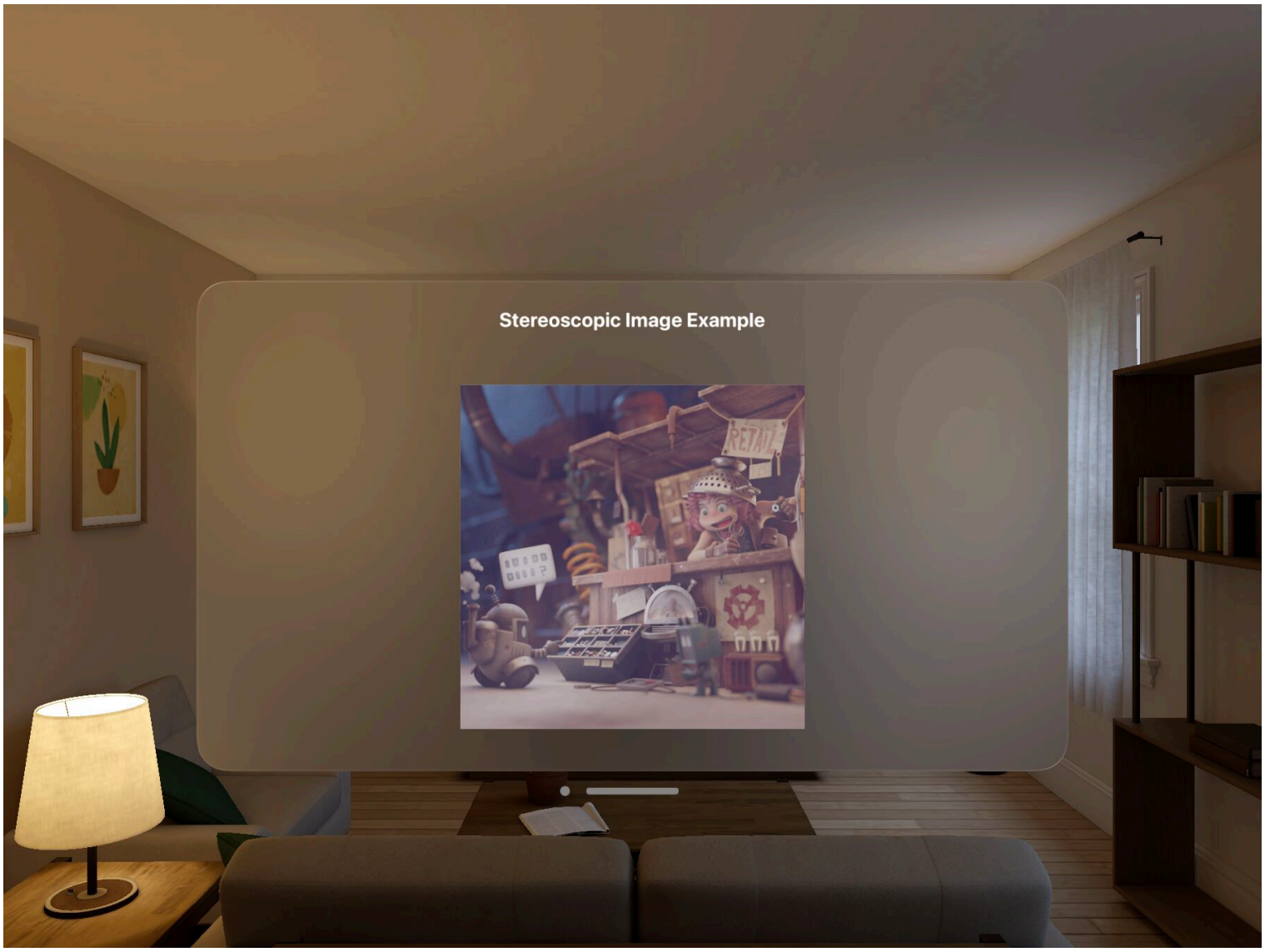
Build a stereoscopic image by applying textures to the left and right eye in a shader graph material.

Download

visionOS 2.0+ | Xcode 16.0+

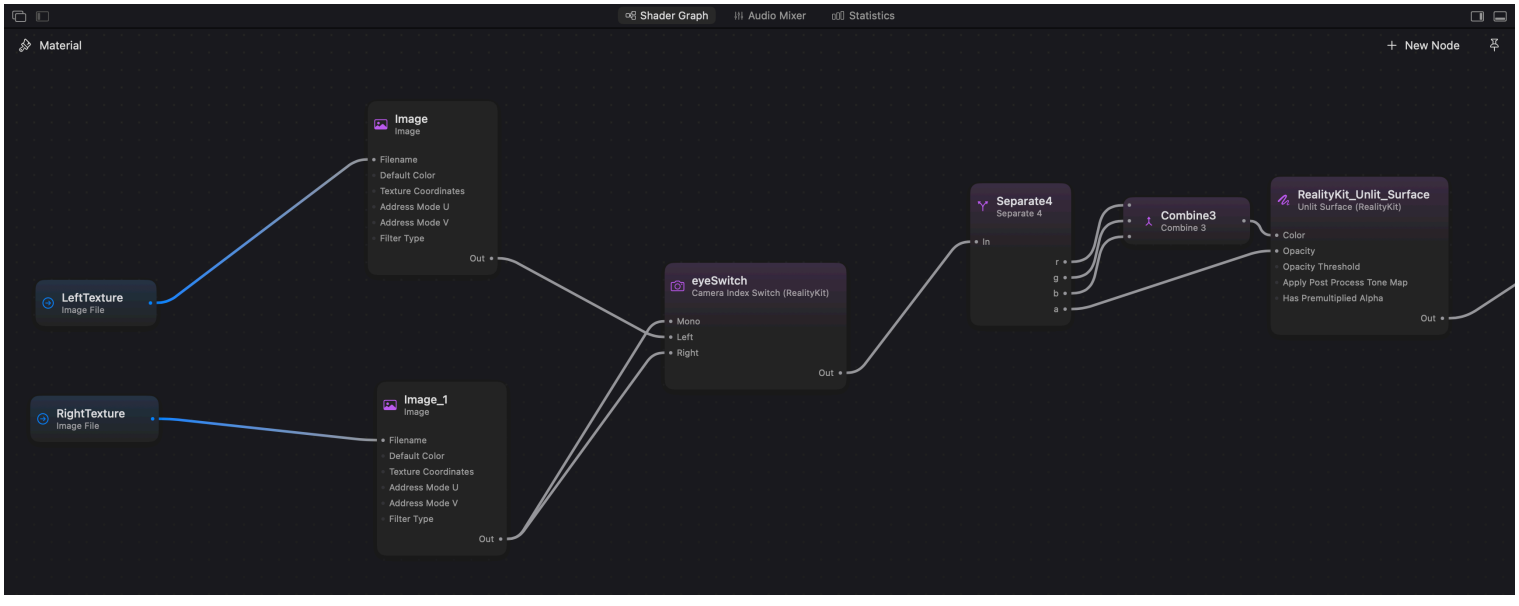
Overview

This sample demonstrates how to create a stereoscopic image with shader graph materials in visionOS. You can apply the textures independently to the left and the right eye. The following image shows a stereoscopic image displaying a pair of left and right images that correspond to the angle of vision of a person's eyes:



Set up the shader graph

The sample sets up the shader graph for the stereoscopic material with two image files: `LeftTexture` and `RightTexture`. The sample applies these image files to the `Camera Index Switch (RealityKit)` node, to set up a dedicated texture for both the left and the right eye, like in the following image:



Load the stereoscopic image as an entity

To create the stereoscopic image, the sample creates the `StereoImageCreator` class, which contains a private method to load the left and right textures. The app sets these textures as parameters to the shader graph material:

```
@MainActor private func applyTextureToEntity(box: ModelEntity, material: inout Shade
    do {
        /// The name of the resource for the left texture.
        let leftFileName = "Shop_L"

        /// The name of the resource for the right texture.
        let rightFileName = "Shop_R"

        /// The left texture for the shader graph material.
        let leftTexture = try await TextureResource(named: leftFileName)

        /// The right texture for the shader graph material.
        let rightTexture = try await TextureResource(named: rightFileName)

        /// The parameter name to modify the left texture in the shader graph.
        let leftParameter = "LeftTexture"

        /// The parameter name to modify the right texture in the shader graph.
        let rightParameter = "RightTexture"

        // Set the textures into the shader graph material.
        try material.setParameter(name: leftParameter, value: .textureResource(leftT
        try material.setParameter(name: rightParameter, value: .textureResource(righ

        // Apply the results to the material and assign to the box's material.
        box.model?.materials = [material]
    } catch {
        // Handle any errors that occur.
        assertionFailure("\(error)")
    }
}
```

The method uses a do-catch block to handle any potential errors that may occur when loading and applying the image files to the material.

The `StereoImageCreator` class also contains the `createImageEntity()` method, which constructs a `ShaderGraphMaterial` and creates a box entity with the shader graph material.

```
@MainActor public func createImageEntity() async -> ModelEntity? {
    /// The full path to the material primitive in the usda file.
    let materialRoot: String = "/Root/Material"

    /// The filename of the material.
    let materialName: String = "StereoscopicMaterial"

    /// The shader graph material for the stereoscopic image.
    guard var material = try? await ShaderGraphMaterial(named: materialRoot, from: m
        print("Failed to load shader graph material.")
        return nil
    }

    // Generates the model entity in the shape of a box and applies the shader graph
    let box = ModelEntity(
        mesh: .generateBox(size: size),
        materials: [material]
    )

    // ...
}
```

The app then modifies the z-axis scale to compress the box into a flat plane, which serves as a placeholder for the image. Lastly, the app calls the `applyTextureToEntity(box: material:)` method to load the left and right textures onto the material and apply the results to the box:

```
@MainActor public func createImageEntity() async -> ModelEntity? {
    // ...

    /// The size of the box in three dimensions.
    let size: Float = 0.3

    /// The z-axis scale to compress the box into an image placeholder.
    let zScale: Float = 1E-3

    // Apply the z-axis scale to compress the box into a flat plane.
    box.scale.z = zScale

    // Load the textures and apply the result to the box.
    await applyTextureToEntity(box: box, material: &material)
```

```
    return box
}
```

Present the stereoscopic image with text

The `StereoImage` view consists of a vertical stack that displays a title and a reality view. Within the reality view, it initializes the `StereoImageCreator` instance, calls the `createImageEntity()` method to create the stereoscopic image, and adds the entity to the reality view. `RealityView` displays your RealityKit content inline in true 3D space. To display the stereoscopic image inline in the window, set the frame's depth to 0:

```
VStack(spacing: spacing) {
    // ...

    RealityView { content in
        /// The creator instance of `StereoImageCreator`.
        let creator = StereoImageCreator()

        /// The image entity that `StereoImageCreator` generates.
        guard let entity = await creator.createImageEntity() else {
            print("Failed to create the stereoscopic image entity.")
            return
        }

        // Add the image entity to the `RealityView`.
        content.add(entity)
    }
    // Set the frame to 0 so the origin of the RealityView exists on the same plane
    .frame(depth: .zero)
}
```

See Also

Building materials

{} Generating procedural textures

Display a 3D model that generates procedural textures in a reality view.

{} Implementing adjustable material

Update the adjustable parameters of a 3D model in visionOS.