Sample Code

Implementing adjustable material

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

Download

visionOS 2.0+ | Xcode 16.0+

Overview

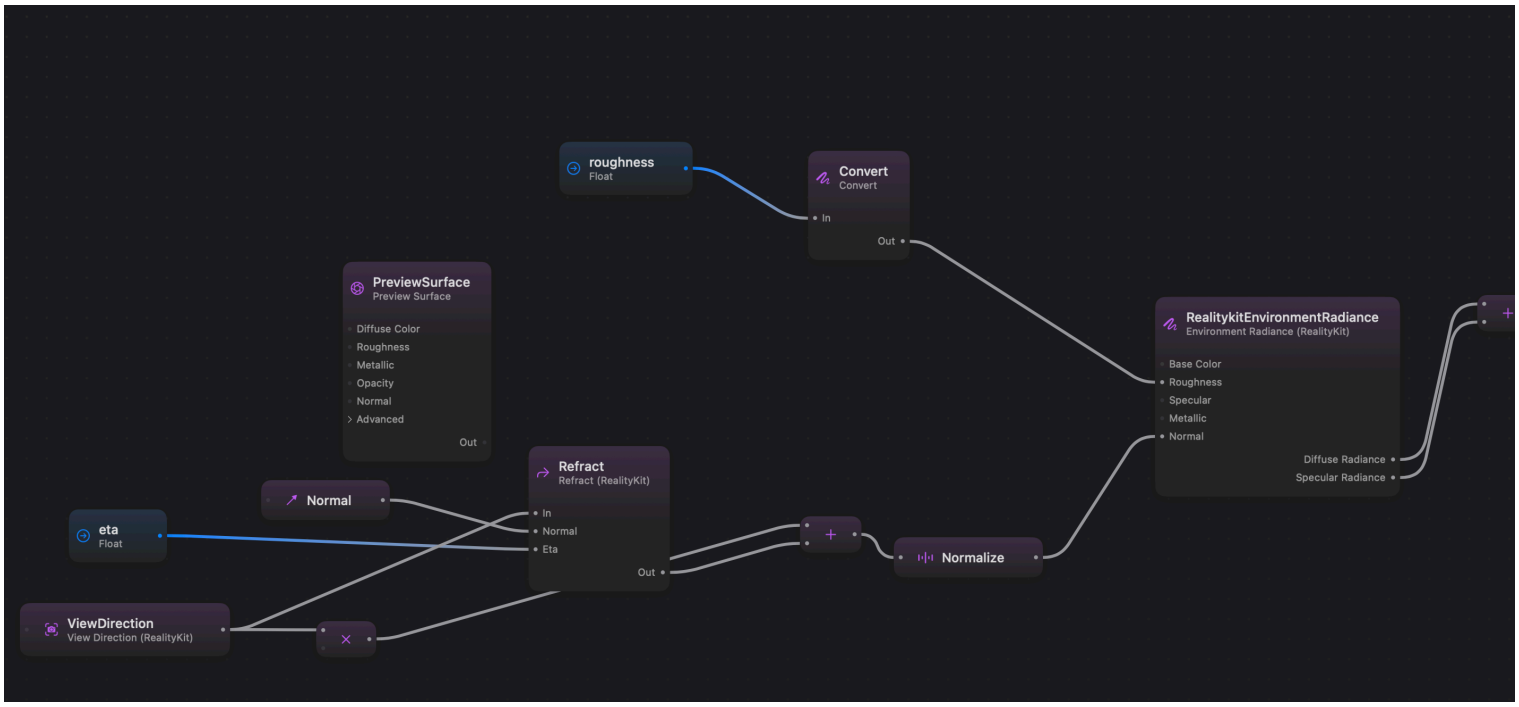
This sample app creates and displays a 3D dragon model within visionOS, using Reality Composer Pro. While the sample focuses on modifying the dragon's adjustable glass material (such as its roughness and refraction), you can also modify the adjustables for other material instances to allow dynamic customization for your visionOS app.

The app splits into two parts in its main view:

- A 3D shape with a glass material
- Two sliders to adjust modifiables

Set up the shader graph

The shader graph for the dragon model includes two modifiable parameters: `roughness` and `eta`. These control the model's surface roughness and refraction, respectively, as the following image shows:



These parameters connect through various nodes, ultimately feeding into a `RealitykitEnvironmentRadiance` node. This setup allows dynamic adjustment of the dragon's appearance.

> **Note**
>
> The parameter `eta` represents η, which symbolizes the index of refraction. For more information, see `Refract (RealityKit)`.

Update the component material

To handle updates to the dragon model, the app creates the `updateMaterials(_:)` method, which employs a recursive approach to update all child entities that the parent entity manages. Then the app applies the `update` closure to each material within the component, to update those materials:

```
import RealityKit

extension Entity {
    public func updateMaterials(_ update: (inout Material) -> Void) {
        // Call recursive method to all child entities.
        for child in children {
            child.updateMaterials(update)
        }

        // Apply the new values to the component material.
```

```
        if var comp = components[ModelComponent.self] {
            comp.materials = comp.materials.map { material in
            var copy = material
            update(&copy)
            return copy
        }
            components.set(comp)
        }
    }
}
```

Set up the main view

To control the material properties of the 3D glass model, the app initializes variables for the default roughness value, eta parameter for refraction, and entity instance of the dragon model in the `GlassView` structure:

```
import SwiftUI
import RealityKit

struct GlassView: View {
    /// The default roughness value of the material.
    @State var roughness: Float = 0.2

    /// The default eta parameter for refraction.
    @State var eta: Float = 0.5

    /// The entity to store the dragon model.
    @State var entity: Entity?

    // ...
}
```

Within the main body of `GlassView`, the app creates a reality view within an `HStack`. The app loads the model by its filename and adjusts its scale and position, then adds the model to the RealityKit content:

```
var body: some View {
    HStack {
        RealityView { content in
            do {
```

```
                // Load the model from the filename.
                let fileName = "DragonGlass.usda"
                let entity = try await Entity(named: fileName)

                // Adjust the scale and position of the model.
                let scale: Float = 2
                let position: SIMD3<Float> = [0.1, -0.15, 0.0]
                entity.scale *= scale
                entity.position += position

                // Add the entity to the `RealityView`.
                content.add(entity)

                // Set `entity` to the dragon model.
                self.entity = entity
            } catch {
                print("Error loading model: \(error)")
            }
        } update: { _ in

            // ...

        }
    }
}
```

The view enables real-time material updates using the optional `update()` closure of the reality view. The app calls `updateMaterials(_:)` on the entity to modify the shader graph material parameters:

```
var body: some View {
    HStack {
        RealityView { content in

            // ...

        } update: { _ in
            entity?.updateMaterials({ material in
                // Obtain the current material of the dragon model.
                guard var mat = material as? ShaderGraphMaterial else { return }

                do {
                    try mat.setParameter(name: "roughness", value: .float(roughness)
```

```
                    try mat.setParameter(name: "eta", value: .float(eta))
                } catch {
                    print("Error setting parameters: \(error)")
                }

                // Update the dragon model's material.
                material = mat
            })
        }

        // ...
    }
}
```

Within the closure that the app passes into the method, the app tries to set the `roughness` and `eta` parameter into the `ShaderGraphMaterial` to update the dragon model.

The view incorporates a `Divider` structure to separate the reality view, which houses the dragon model, from the `VStack`, which houses sliders that enable interactive adjustment of the `roughness` and `eta` values, to modify the model's visual appearance:

```
var body: some View {
    HStack {
        // ...

        Divider()

        // Set up a slider in the window to update the `roughness`
        // and `eta` parameter to a value between 0 and 1.
        VStack {
            HStack {
                Text("Roughness"); Spacer(); Text("\(roughness)")
            }
            Slider(value: $roughness, in: 0...1)

            Divider()

            HStack {
                Text("ETA:"); Spacer(); Text("\(eta)")
            }
            Slider(value: $eta, in: 0...1)
        }.frame(maxWidth: 200)
    }.padding()
```

```
}
```

See Also

Building materials

{} Generating procedural textures

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

{} Displaying a stereoscopic image

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