Sample Code

Responding to gestures on an entity

Respond to gestures performed on RealityKit entities using input target and collision components.

Download

visionOS 2.0+ | Xcode 16.3+

Overview

Your app responds to `RealityView` gesture events received from SwiftUI by adding an `InputTargetComponent` and a `CollisionComponent` to your entities. Add a `HoverEffectComponent` to the entity so it highlights as the gaze intersects the collision shape. The input target component marks the entity as participating in the event system. The system uses the collision component to test if the gaze vector intersects the entity. With both components attached, entities receive events. With the hover effect attached the entity visually appears ready to receive events.

Attach components to an entity to process events

The sample defines the `ActiveComponent`, which keeps track of the `active` state of the entity.

```
public class ActiveComponent: Component {
    public var active: Bool = false
}
```

This sample has one entity, a cube that's `0.1` units in each direction. The sample creates the entity then adds the components.

```
var cube = ModelEntity(mesh: .generateBox(size: 0.1),
                       materials: [SimpleMaterial(color: .orange, isMetallic: false)

cube.components.set(InputTargetComponent())
cube.components.set(CollisionComponent(shapes: [ShapeResource.generateBox(size: SIMD
cube.components.set(HoverEffectComponent())
cube.components.set(ActiveComponent())
```

The sample has a `SpatialEventGesture` attached to the `RealityView`. As the person interacting with the app looks around and pinches, the system uses the input target component and the collision component to determine intent. The system considers all entities in the scene because the sample calls `targetedToAnyEntity`. When the person pinches on the cube the system invokes the gesture's `onEnded` block, which toggles the `active` flag.

```
.gesture(SpatialEventGesture()
    .targetedToAnyEntity()
    .onEnded { value in
        value.entity.components[ActiveComponent.self]?.active.toggle()
    })
```

The `attachments:` block creates an attachment with the name of the `cube` entity.

```
attachments: {
    Attachment(id: cube.id) {
        Text("\(cube.name)")
            .padding()
            .glassBackgroundEffect(in: RoundedRectangle(cornerRadius: 5.0))
            .tag(cube.id)
    }
}
```

The attachment's `id` is set to the ID of the `cube` so that it's easy to find in the `update:` block of `RealityView`. In the `update:` block, the sample finds the `ActiveComponent` from the cube and then finds the attachment using the ID of the cube. If the active value is `true` the code adds a `BillboardComponent` to the attachment. The system ensures entities with a `Billboard Component` always face the person. The `RealityView update` block adds the attachment entity as a subentity of the cube and sets the position to `[0.0, 0.1, 0.0]`.

```
update: { content, attachments in
    guard let component = cube.components[ActiveComponent.self] else { return }
```

```
    guard let attachmentEntity = attachments.entity(for: cube.id) else { return }
    if component.active {
        attachmentEntity.components.set(BillboardComponent())
        cube.addChild(attachmentEntity)
        attachmentEntity.setPosition(SIMD3<Float>(0.0, 0.1, 0.0),
                                     relativeTo: cube)
    } else {
        cube.removeChild(attachmentEntity)
    }
}
```

See Also

Scene content

{} Hello World

Use windows, volumes, and immersive spaces to teach people about the Earth.

{} Enabling video reflections in an immersive environment

Create a more immersive experience by adding video reflections in a custom environment.

{} Creating a spatial drawing app with RealityKit

Use low-level mesh and texture APIs to achieve fast updates to a person's brush strokes by integrating RealityKit with ARKit and SwiftUI.

{} Generating interactive geometry with RealityKit

Create an interactive mesh with low-level mesh and low-level texture.

{} Combining 2D and 3D views in an immersive app

Use attachments to place 2D content relative to 3D content in your visionOS app.

{} Transforming RealityKit entities using gestures

Build a RealityKit component to support standard visionOS gestures on any entity.

Models and meshes

Display virtual objects in your scene with mesh-based models.

Materials, textures, and shaders

Apply textures to the surface of your scene's 3D objects to give each object a unique appearance.

- Anchors

  Lock virtual content to the real world.

- Lights and cameras

  Control the lighting and point of view for a scene.

- Content synchronization

  Synchronize the contents of entities locally or across the network.

- Audio

  Create personalized and realistic spatial audio experiences.

- Videos

  Present videos in your RealityKit experiences.

- Images

  Present images and spatial scenes in your RealityKit experiences.