Sample Code

Animating entity rotation with a system

Rotate an entity around an axis using a Component and a System.

Download

visionOS 2.0+ | Xcode 16.3+

Overview

You can use the Entity Component System (ECS) from RealityKit to apply logic to any number of entities in a scene. This sample code provides a component. The system queries scene for all entities that have that component and then rotates those entities based on the speed and axis in the component.

For more information about the ECS, see Understanding the modular architecture of RealityKit, and for more information about systems, see Implementing systems for entities in a scene.

Add the rotation component to the entity

After creating the entity in `ContentView` the sample adds an instance of the `RotationComponent` to the entity:

```
telescopeScene.components[RotationComponent.self] = RotationComponent()
```

By default, the rotation speed is `0.0` and the axis of rotation is the x-axis.

Enable SwiftUI to control the animation

The `ContentView` type has two `@State` variables that control the animation:

```
@State var axis: RotationAxis = .x
@State var rotating: Bool = false
```

- `RotationAxis` is an enumeration with one case for each ordinal direction.
- The `rotating` value sets the `speed` to `0.0` when `rotating` is `false`, and `1.0` when `rotating` is `true`.

Update the RealityKit state on SwiftUI state changes

The system calls the `RealityView` `update:` block when the `rotating` or `speed` values change.

```
update: { context in
  telescope?.components[RotationComponent.self]?.rotationAxis = axis
  telescope?.components[RotationComponent.self]?.speed = rotating ? 1.0 : 0.0
}
```

Update the rotation of the entity on every frame

The framework calls the `RotationSystem` function `update(context: SceneUpdateContext)` for each frame. The sample includes a query to quickly find entities that have the component:

```
static let rotationQuery = EntityQuery(where: .has(RotationComponent.self))
```

This function uses `rotationQuery` to find relevant entities. The function then sets the orientation of the entities based on the values in the attached component:

```
for entity in context.entities(matching: Self.rotationQuery, updatingSystemWhen: .re
  guard let component: RotationComponent = entity.components[RotationComponent.self]
  if component.speed == 0 { continue }
  entity.setOrientation(simd_quatf(angle: component.speed * Float(context.deltaTime)
                                   axis: component.axis),
                        relativeTo: entity)
}
```

While the component's `speed` value isn't zero, the system animates the entity's rotation around `axis`.

The `relativeTo:` argument specifies what coordinate system the rotation acts in. Specifying `entity` here rotates the entity around its own coordinate system. In this sample there is only one entity. In a more complex scene many interesting effects are achieved by rotating in a different coordinate system.

See Also

System configuration

Implementing systems for entities in a scene

Apply behaviors and physical effects to the objects and characters in a RealityKit scene with the Entity Component System (ECS).

`protocol System`

An object that affects multiple entities in every update of a RealityKit scene.

`struct SystemUpdateCondition`

A condition which causes a system to update.

`struct SceneUpdateContext`

An object that contains information about the scene to update.