Sample Code

Transforming entities between RealityKit coordinate spaces

Move an entity between a volumetric window and an immersive space using coordinate space transformations.

Download

visionOS 2.0+ | Xcode 16.2+

Overview

visionOS has three scene types: Windows, Volumes, and Immersive Spaces.

The article describes how you can move entities between these different types by transforming an entity from one coordinate space to another.

> **Note**
>
> For more information about the different types of spaces available in visionOS, watch the WWDC24 session, Dive deep into volumes and immersive spaces.

One example of wanting to do this could be a shopping app that displays content within a volumetric window. A person may want to move that content outside of the window, and into their surroundings (an immersive space), to compare multiple items with each other. The person using this app would expect its size to remain consistent, and have no disconnect between spaces.

RealityKit has two named coordinate spaces:

- `Entity.CoordinateSpaceReference.scene`
- `Entity.CoordinateSpaceReference.immersiveSpace`

The scene coordinate space has its origin at the center-back of the volumetric window and is a reference to the entity's container window scene. The immersive coordinate space has its origin at the point on the ground below you while the immersive space is open.

The sample code seamlessly moves an entity from a volumetric window to an immersive space using the `transformMatrix(relativeTo:)` method.

> **Note**
>
> Alternatively you could use `transform(from:to:)` method as shown in Dive deep into volumes and immersive spaces

The sample code creates a cube in a volumetric window and moves the cube to an immersive space and back.

Create the volumetric and immersive views

Create the volumetric reality view `VolumetricView` in the `WindowGroup` and the immersive reality view `ImmersiveView` in an `ImmersiveSpace`:

```
var body: some Scene {
    WindowGroup(id: "VolumetricView") {
        VolumetricView()
            .environment(appModel)
            .onChange(of: appModel.showImmersiveSpace) { _, newValue in
                handleImmersiveSpaceStateChange(showImmersiveSpace: newValue)
            }
    }
    .windowStyle(.volumetric)
    .defaultSize(width: 1.0, height: 1.0, depth: 1.0, in: .meters)

    ImmersiveSpace(id: "ImmersiveSpace") {
        ImmersiveView()
            .environment(appModel)
            .preferredSurroundingsEffect(.dark)
    }
}
```

The width, height and depth of the volumetric window is set to 1 meter. When the immersive space is open, the surroundings become dark so that the virtual content is in focus and when it is hidden, the surroundings gradually become brighter again.

Create a cube in the volumetric window

Create an entity with a cube mesh in the volumetric window and set the material to the `volumetricMaterial` which is blue in color. The blue color indicates that it is part of the volumetric window. Make the cube a subentity of the `volumeRootEntity` which is an entity at the root of the volumetric window.

```
private func createCube() {

    appModel.volumeCube.components.set([
        ModelComponent(
            mesh: .generateBox(size: 0.1, cornerRadius: 0.01),
            materials: [appModel.volumetricMaterial]),
        InputTargetComponent(allowedInputTypes: .indirect),
        HoverEffectComponent(),
        CollisionComponent(shapes: [
            ShapeResource.generateBox(size: [0.1, 0.1, 0.1])
        ])
    ])

    // Make the volume cube a subentity of the volumetric window's root.
    appModel.volumeRootEntity.addChild(appModel.volumeCube)
}
```

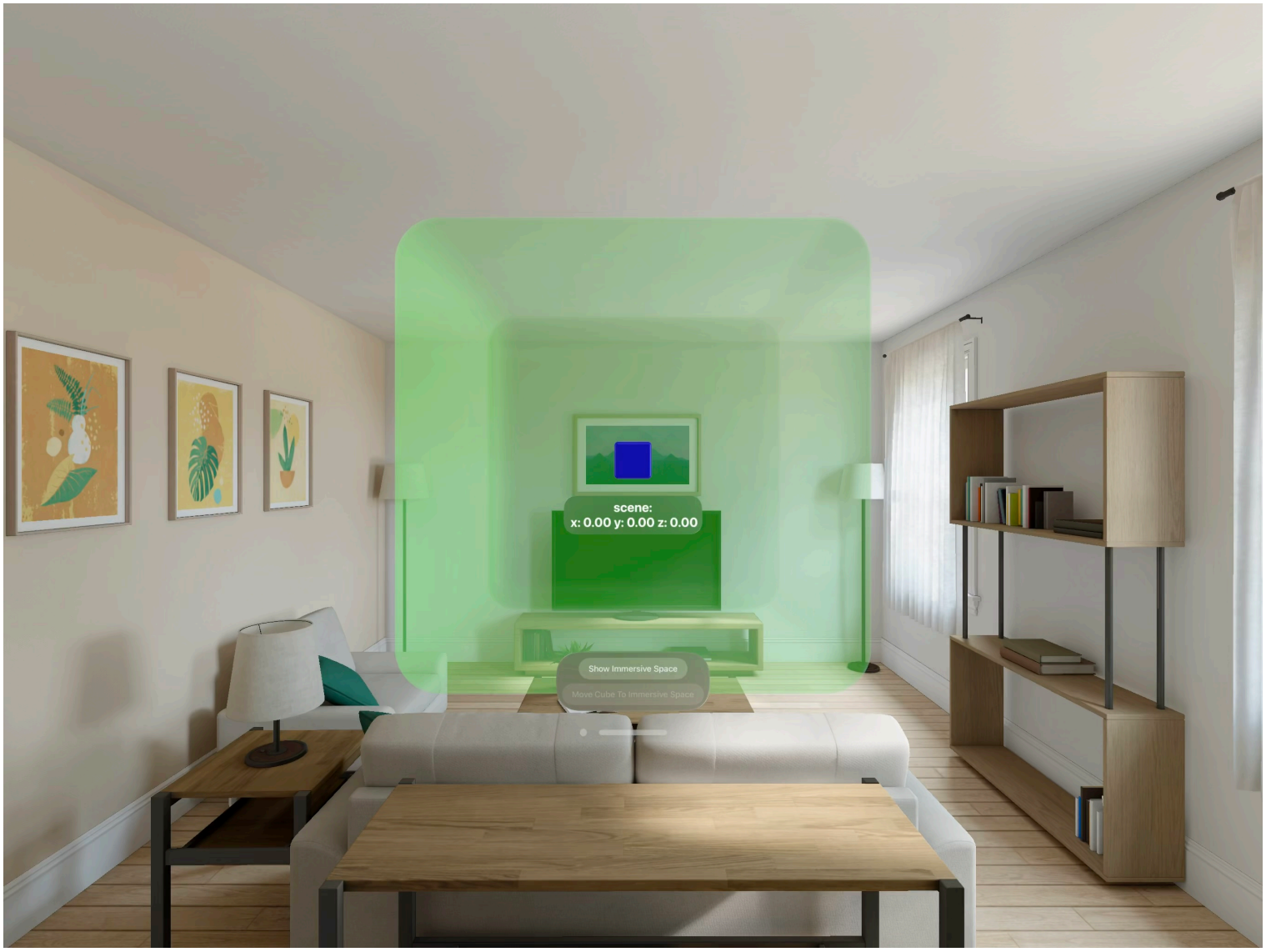
Create a cube representing the bounds of the volumetric window

Create an entity with a cube mesh with the size of 1 meter to represent the bounds of the volumetric window. Set the material to the `volumetricWindowCubematerial` which is set to a transparent green color.

```
private func createVolumetricWindowCube() -> ModelEntity {
    var volumetricWindowCubematerial = SimpleMaterial(
        color: #colorLiteral(red: 0.0, green: 1.0, blue: 0.0, alpha: 0.4),
        roughness: 0.5,
        isMetallic: false)

    volumetricWindowCubematerial.faceCulling = .front

    let volumetricWindowCube = ModelEntity(
```

```
        mesh: .generateBox(size: 1.0, cornerRadius: 0.1),
        materials: [volumetricWindowCubematerial])

    return volumetricWindowCube
}
```

The `faceCulling` property of the material is set to `front` so that the bounds of the volume and cube inside it can be seen clearly.



Play

Move the cube from the volumetric window to the immersive space

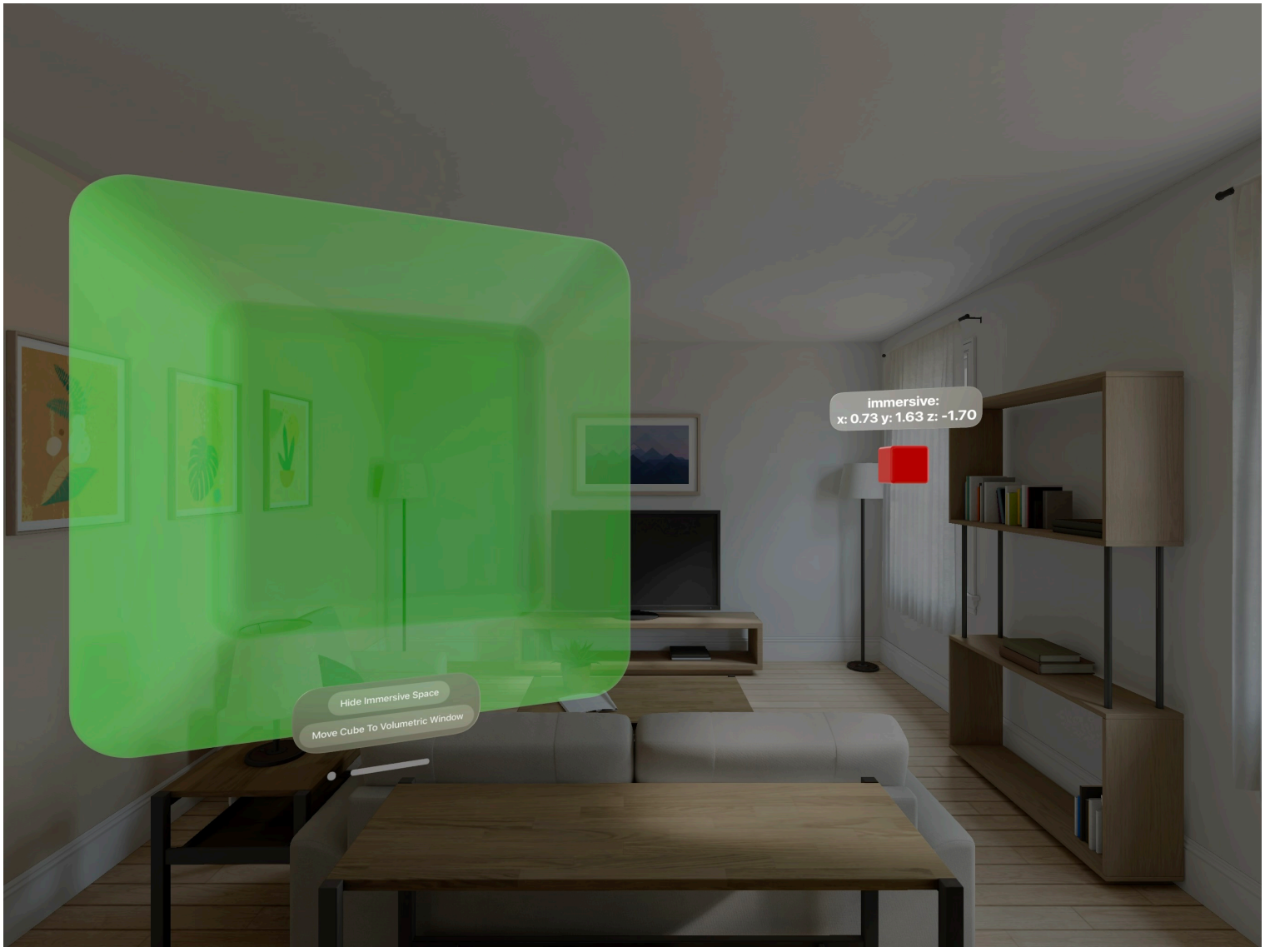
Get the transform matrix of the entity relative to the open immersive space calling the `transformMatrix(relativeTo:)` method.

> **Note**
>
> When using the the `Entity.CoordinateSpaceReference.immersiveSpace` as the value for the argument in the `transformMatrix(relativeTo:)` method, you will need to ensure that an immersive space is open and the immersive view has a `RealityView` in it. Otherwise the method will return nil.

Make the cube a subentity of the `immersiveSpaceRootEntity`, which is an entity at the root of the immersive space. Set the transform matrix of the entity relative to it's container entity calling the `Entity/setTransformMatrix(_:relativeTo:)` method.

```
private func moveCubeFromVolumetricWindowToImmersiveSpace() {

    // Only move the cube if the immersive space is open.
    guard appModel.immersiveSpaceIsShown else { return }

    // Record the cube's transform before moving it to the immersive space.
    // The app uses this information to move the cube back to the volumetric window.
    appModel.volumeCubeLastSceneTransformEntity.transform = appModel.volumeCube.tran

    // Get the transformation matrix of the cube relative to the immersive space.
    let cubeToSpaceMatrix = appModel.volumeCube.transformMatrix(relativeTo: .immersi

    // Add the cube as a child of the immersive space's root entity.
    appModel.immersiveSpaceRootEntity.addChild(appModel.volumeCube)

    // Set the transformation matrix of the cube relative to the immersive space.
    appModel.volumeCube.setTransformMatrix(cubeToSpaceMatrix ?? matrix_identity_floa
                                           relativeTo: appModel.volumeCube.parent)

    // Change the material to the immersive space material.
    appModel.volumeCube.components[ModelComponent.self]?.materials = [
        appModel.immersiveSpaceMaterial
    ]

    // Disable the scene transform attachment as the volume cube is in the immersive
    appModel.sceneAttachmentEntity?.isEnabled = false

    appModel.cubeInImmersiveSpace = true
}
```

The method records the cube's transform for use when the cube moves from the immersive space back to the volumetric window. The color of the cube changes from blue to red when the material changes to the `immersiveSpaceMaterial`. Disable the `sceneAttachmentEntity`, as the translation value with respect to the scene coordinate space is the same as the translation value with respect to the immersive space. The sample code allows you to move the cube between the volumetric window and the immersive space by either tapping on the button on the volumetric window or by double-tapping the cube itself.



Play ⊙

Move the cube back to the volumetric window

To move the entity back to the volumetric window, set the transform of the cube to the previously recorded value before it moved to the immersive space.

The sample code uses the `Entity/move(to:relativeTo:duration:timingFunction:)-905k` method to smoothly animate the transformation to the previously recorded transform of the cube relative to the immersive space:

```
func moveCubeFromImmersiveSpaceToVolumetricWindow() {

    // Get the transformation matrix of the volume cube's previously
    // recorded transform relative to the immersive space.
    let cubeToSpaceMatrix =
        volumeCubeLastSceneTransformEntity.transformMatrix(
            relativeTo: .immersiveSpace)

    // Create a transform from the transformation matrix.
    let cubeToSpaceTransform = Transform(
        matrix: cubeToSpaceMatrix ?? matrix_identity_float4x4)

    // Move the volume cube to its previously recorded transform
    // over a period of time.
    volumeCube.move(
        to: cubeToSpaceTransform,
        relativeTo: nil,
        duration: getDurationBasedOnDistance(durationMaxLimit: 2.0),
        timingFunction: .easeOut)
}
```

The above code uses the `transformMatrix(relativeTo:)` method to get the transformation matrix of the previously recorded value relative to the immersive space. The transformation matrix creates a `Transform` for specifying the target transform in the `Entity/move(to:relativeTo:duration:timingFunction:)-905k` method.

Make the cube a subentity of the volumetric window's root

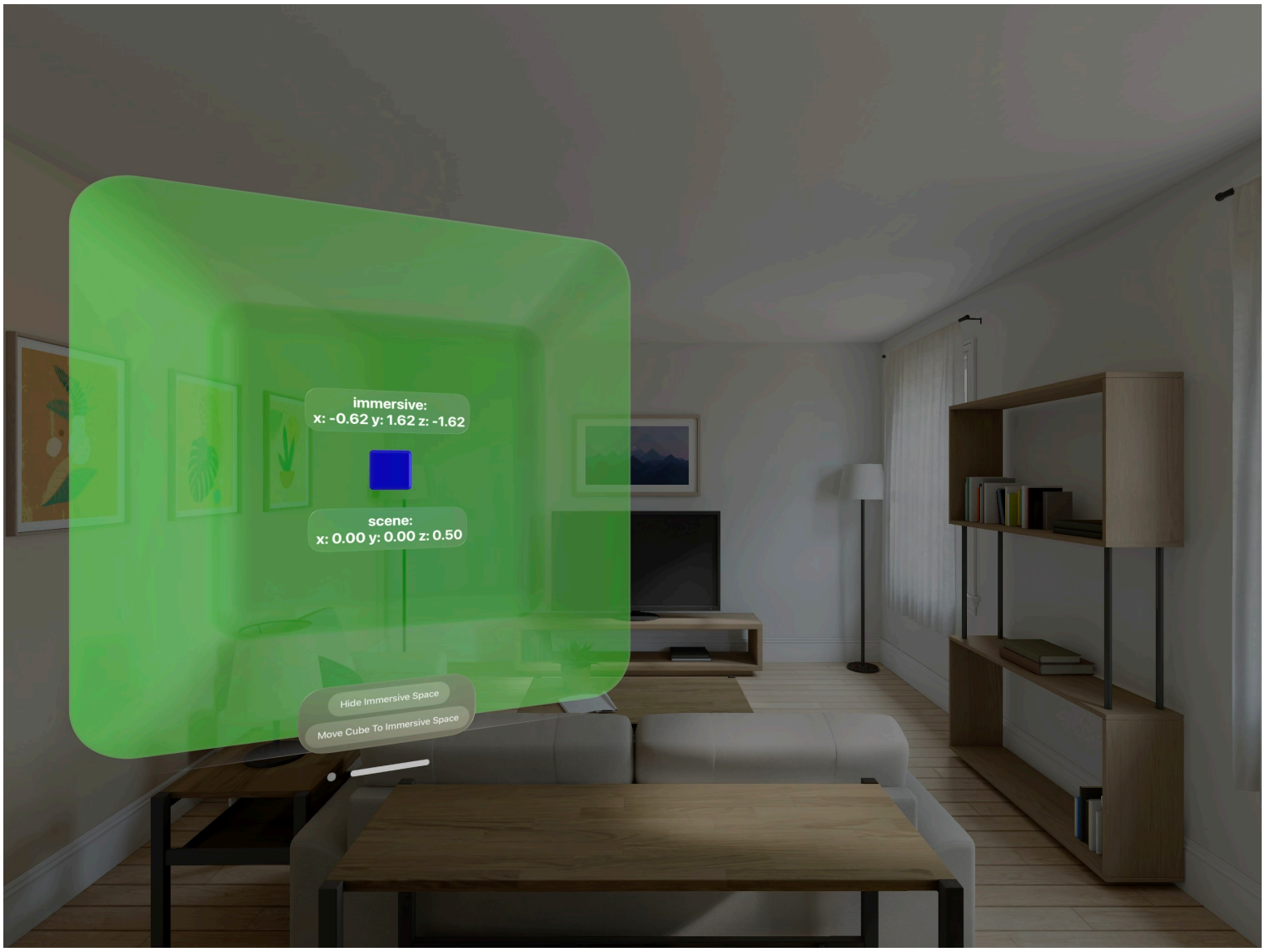
When the move animation finishes, make the cube a subentity of the volumetric window's root entity and set the transform to the previously recorded value:

```
func makeCubeSubEntityOfVolumeRoot() {

    // Make the volume cube a subentity of the volumetric window's root.
    appModel.volumeRootEntity.addChild(appModel.volumeCube)

    // Set the transform to the last recorded transform.
    appModel.volumeCube.transform =
        appModel.volumeCubeLastSceneTransformEntity.transform
```

```
    // Set the material to the volumetric material.
    appModel.volumeCube.components[ModelComponent.self]?.materials = [
        appModel.volumetricMaterial
    ]

    // Update the scene and immersive space transforms.
    appModel.updateTransform(relativeTo: .scene)
    appModel.updateTransform(relativeTo: .immersiveSpace)

    // Enable the scene transform attachment because the cube is in the
    // volumetric window.
    appModel.sceneAttachmentEntity?.isEnabled = true

    appModel.cubeInImmersiveSpace = false
}
```

The cube is made a subentity of the `volumeRootEntity` as it needs to move from the immersive space into the volumetric space. If this isn't done, the cube remains in the immersive space and won't move along with the volumetric window.

The color of the cube changes from red back to blue, indicating that it's in the volumetric window. The scene and immersive space transforms update to the latest values after completing the move animation. The scene attachment is enabled as the scene and immersive space transform values are different when the cube is in the volumetric window.



Play ⊙

See Also

Using a matrix

`func transformMatrix(relativeTo: Entity?) -> float4x4`

Gets the 4 x 4 transform matrix of an entity relative to the given entity.

`func setTransformMatrix(float4x4, relativeTo: Entity?)`

Sets the transform of the entity relative to the given reference entity using a 4x4 matrix representation.