Sample Code

Presenting images in RealityKit

Create and display spatial scenes in RealityKit

Download

visionOS 26.0+ | Xcode 26.0+

Overview

RealityKit apps can easily display images in 3D space using `ImagePresentationComponent`, which can display traditional 2D and *spatial photos* as well as generate and display *spatial scenes* — which represents the content of an existing image in three dimensions.

Spatial scenes are different from *spatial photos*. A *spatial photo* presents two separate 2D images, one to each eye, to create the illusion of a three dimensional view. *Spatial scenes*, on the other hand, generate textured 3D geometry from either a *spatial photo* or a regular 2D image.

This sample app demonstrates how to use `ImagePresentationComponent` and `ImagePresentationComponent.Spatial3DImage` to convert an existing 2D image to a 3D spatial scene, and how to present the 2D and 3D versions of the image using `RealityView` in a SwiftUI app.

Choose viewing modes

Image presentation components can present images in several modes. Your apps can choose to use any or all of these modes.

`mono`
: Shows an image from a single point of view.

`spatial3D`
: Shows a *spatial scene* from the source image.

`spatial3DImmersive`
: Shows a *spatial scene* from the source image and displays it in immersive mode.

`spatialStereo`

Shows an image as a *spatial photo*.

`spatialStereoImmersive`
: Shows an image as *spatial photo* and displays it in immersive mode.

This sample displays images using `mono` and `spatial3D` viewing modes.

Create an entity to present the image

To display an image, create an `ImagePresentationComponent` and attach it to an entity in your scene. You can attach it to any entity, and you'll also want to attach it to one that has no visual representation in your scene. This sample creates an empty `Entity` property called `content Entity` in the `AppModel` class for that purpose.

```
var contentEntity: Entity = Entity()
```

In the `RealityView` `make` closure, the app calls an asynchronous method to create an `Image PresentationComponent` and adds it to `contentEntity`.

```
await appModel.createImagePresentationComponent()
```

The `createImagePresentationComponent` function creates an `ImagePresentation Component.Spatial3DImage` from a 2D image, then creates an image presentation component with that image and attaches it to the `contentEntity`:

```
func createImagePresentationComponent() async {
    guard let imageURL else {
        print("ImageURL is nil.")
        return
    }
    spatial3DImageState = .notGenerated
    spatial3DImage = nil
    do {
        spatial3DImage = try await ImagePresentationComponent.Spatial3DImage(content
    } catch {
        print("Unable to initialize spatial 3D image: \(error.localizedDescription)'
    }

    guard let spatial3DImage else {
        print("Spatial3DImage is nil.")
        return
```

```
    }

    let imagePresentationComponent = ImagePresentationComponent(spatial3DImage: spat
    contentEntity.components.set(imagePresentationComponent)
    if let aspectRatio = imagePresentationComponent.aspectRatio(for: .mono) {
        imageAspectRatio = CGFloat(aspectRatio)
    }
}
```

The `createImagePresentationComponent` method stores the `ImagePresentationComponent/aspectRatio` of the newly created ImagePresentationComponent in the App `Model`.

The app implements an `onChange(of:perform:)` modifier for `aspectRatio` in the `AppModel` to ensure that the `UIWindowScene` size matches the image.

```
.onChange(of: appModel.imageAspectRatio) { _, newAspectRatio in
    guard let windowScene = sceneDelegate.windowScene else {
        print("Unable to get the window scene. Resizing is not possible.")
        return
    }

    let windowSceneSize = windowScene.effectiveGeometry.coordinateSpace.bounds.size

    //  width / height = aspect ratio
    // Change ONLY the width to match the aspect ratio.
    let width = newAspectRatio * windowSceneSize.height

    // Keep the height the same.
    let size = CGSize(width: width, height: UIProposedSceneSizeNoPreference)

    UIView.performWithoutAnimation {
        // Update the scene size.
        windowScene.requestGeometryUpdate(.Vision(size: size))
    }
}
```

Manage image presentation

In the `update` closure of the `RealityView`, the app retrieves the presentation screen size of the image presentation component using the entity's `observable` property. This ensures that update

is called when the `presentationScreenSize` changes.

```
guard let presentationScreenSize = appModel
    .contentEntity
    .observable
    .components[ImagePresentationComponent.self]?
    .presentationScreenSize, presentationScreenSize != .zero else {
        print("Unable to get a valid presentation screen size from the content entit
        return
}
```

The app sets the z axis position of the `contentEntity` to 0.0. This displays the image presentation component flush against the background.

```
let originalPosition = appModel.contentEntity.position(relativeTo: nil)
appModel.contentEntity.setPosition(SIMD3<Float>(originalPosition.x, originalPositior
```

To display the image at an appropriate size, the app wraps a `RealityView` inside a `GeometryReader3D`:

```
GeometryReader3D { geometry in
    RealityView { content in
```

In the `make` and `update` closure of the `RealityView`, the app converts the geometry reader's frame bounds into the scene's coordinate space:

```
let availableBounds = content.convert(geometry.frame(in: .local), from: .local, to:
```

Then, the app calls the `scaleImagePresentationToFit` method which scales the image to fit into the geometry reader's frame bounds:

```
scaleImagePresentationToFit(in: availableBounds)
```

The `scaleImagePresentationToFit` method calculates x and y scale values to preserve the aspect ratio of the presented image at the current `presentationScreenSize`, and sets those scale values as the content entity's scale:

```
func scaleImagePresentationToFit(in boundsInMeters: BoundingBox) {
    guard let imagePresentationComponent = appModel.contentEntity.components[ImagePr
        return
    }

    let presentationScreenSize = imagePresentationComponent.presentationScreenSize
    let scale = min(
        boundsInMeters.extents.x / presentationScreenSize.x,
        boundsInMeters.extents.y / presentationScreenSize.y
    )

    appModel.contentEntity.scale = SIMD3<Float>(scale, scale, 1.0)
}
```

Generate a spatial scene

It can take several seconds to generate a spatial scene. To preserve the user experience, you have two options:

- You can generate the spatial scene first and then add it to the image presentation component.
- Alternatively, you can can add it to the image presentation component first and then generate the spatial scene afterwards.

If you create the spatial scene before adding it to the component, the generated spatial scene appears as soon as you add it. If you add a 2D or stereo image to the component first and then generate the spatial scene later, the component presents a conversion UI like the one in the Photos app. This indicates that it's generating the spatial scene

> **Note**
>
> Generation and viewing of spatial scenes is not supported in the simulator.

Play ⦿

This sample adds the images to the component first, then generates the spatial scene on a button press. It does that by first declaring an enumeration in the app data model to represent the current status of the displayed image.

```
enum Spatial3DImageState {
    case notGenerated
    case generating
    case generated
}
```

The app currently displays a 2D image in an `ImagePresentationComponent`. When the viewer clicks the Show as 3D button for the first time, it checks to see if the spatial 3D image has been generated, and returns if it has to avoid doing unnecessary work.

```
guard spatial3DImageState == .notGenerated else {
    print("Spatial 3D image already generated or generation is in progress.")
    return
```

```
}
```

The viewing mode of the image presentation component changes to `spatial3D`, calls `generate()` on the spatial 3D image it displays, and sets the image state to `.generated` so it knows not to generate it again:

```
guard var imagePresentationComponent = contentEntity.components[ImagePresentationCom
    print("ImagePresentationComponent is missing from the entity.")
    return
}
// Set the desired viewing mode before generating so that it will trigger the
// generation animation.
imagePresentationComponent.desiredViewingMode = .spatial3D
contentEntity.components.set(imagePresentationComponent)

// Generate the Spatial3DImage scene.
spatial3DImageState = .generating
try await spatial3DImage.generate()
spatial3DImageState = .generated
```