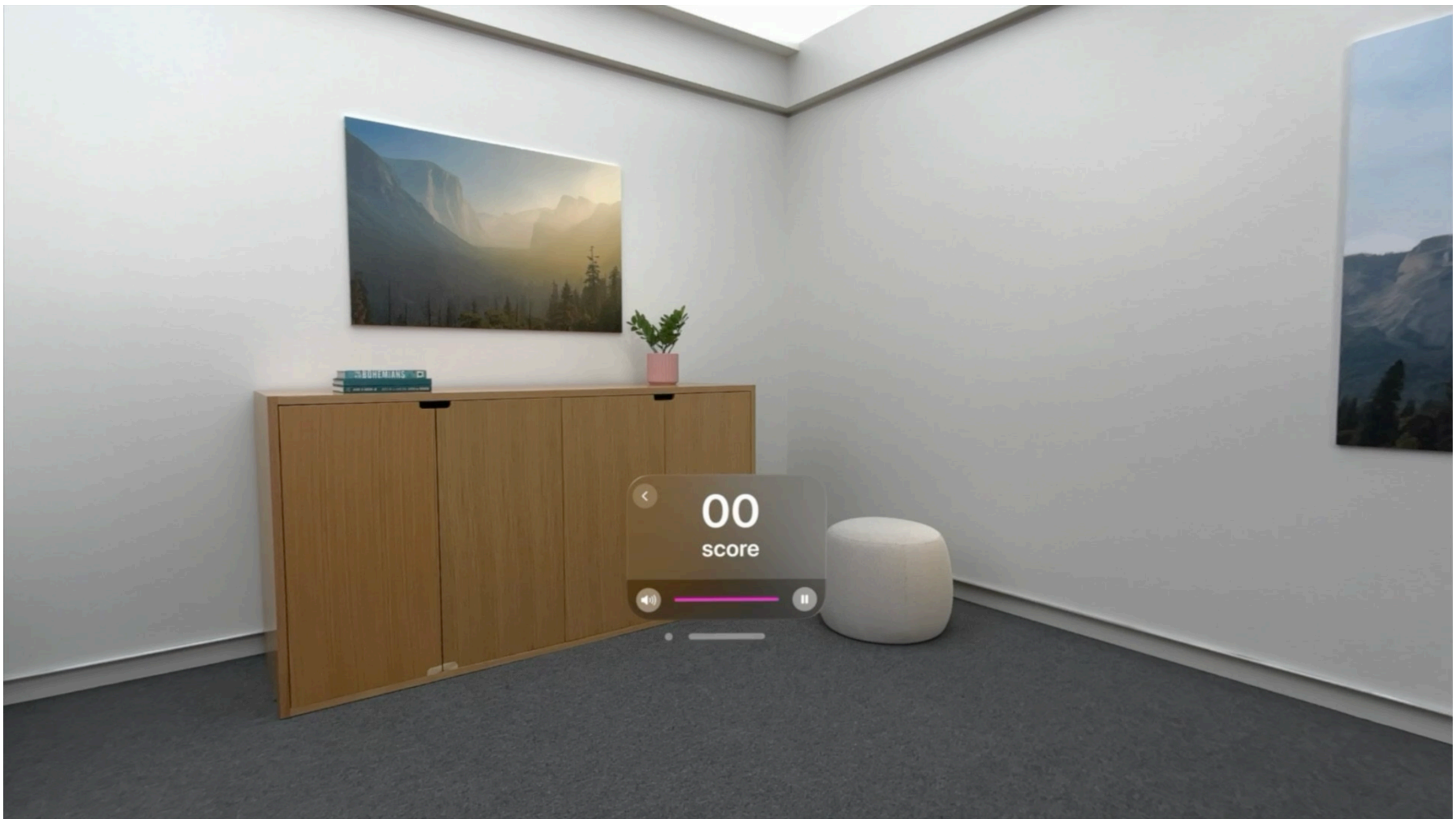
Article

Improving accessibility support in your visionOS app

Update your code to ensure everyone can access your app’s content in visionOS.

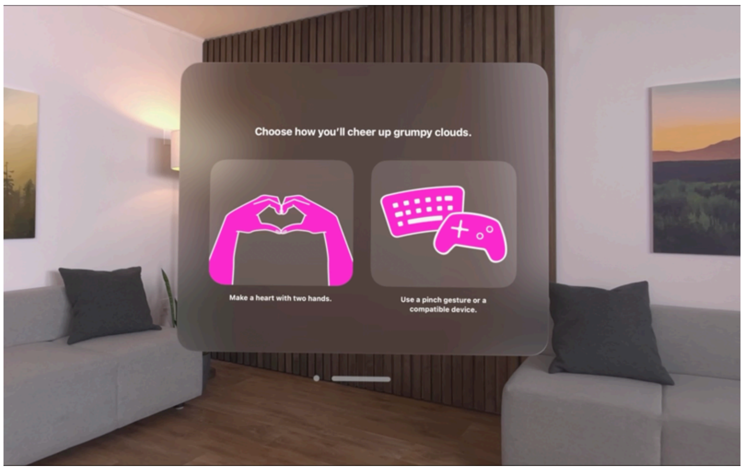
Overview

visionOS is an immersive platform that supports people of all abilities. Even though experiences incorporate stunning visual content and hand- and eye-tracking technologies, people can engage with content in other ways. In fact, the platform supports people in many different situations, including those who are blind, have low vision, have limited mobility, or have limb differences. With the help of assistive technologies, people can interact with all of your app’s content.

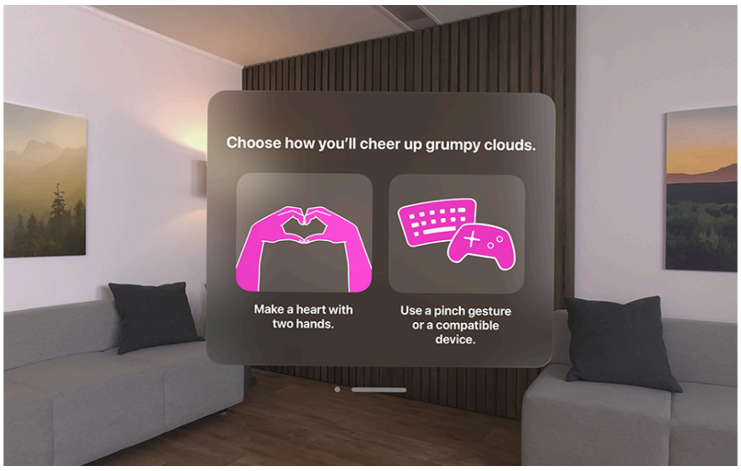


Play ⊙

During development, enable VoiceOver and other assistive features and test your app's accessibility support. Make sure people can navigate your app's interface intuitively, and that all of the necessary elements are present. Improve the descriptive information for those elements to communicate their intended purpose. And make sure your app adapts to changing conditions, such as changes to the Dynamic Type setting while your app is running.


Choose how you'll cheer up grumpy clouds.
Make a heart with two hands.
Use a pinch gesture or a compatible device.

Default font size


Choose how you'll cheer up grumpy clouds.
Make a heart with two hands.
Use a pinch gesture or a compatible device.

Increased font size

For general information about supporting accessibility, see Accessibility. For design guidance, see Human Interface Guidelines > Accessibility.

Add accessibility traits to RealityKit entities

VoiceOver and other assistive technologies rely on the accessibility information that your app's views and content provide. SwiftUI and UIKit provide default information for the standard system views, but RealityKit doesn't provide default information for the entities in your scenes.

To configure the accessibility information for a RealityKit entity, add an instance of `AccessibilityComponent` to the entity. Use this component to specify the same values you specify for the rest of your app's views. The following example shows how to create this component and add it to an entity:

```
var accessibilityComponent = AccessibilityComponent()
accessibilityComponent.isAccessibilityElement = true
accessibilityComponent.traits = [.button]
accessibilityComponent.label = "Sports car"
accessibilityComponent.value = "Parked"
accessibilityComponent.systemActions = [.activate]
myEntity.components[AccessibilityComponent.self] = accessibilityComponent
```

People can use VoiceOver to initiate specific types of actions on your entities. Assign a value to the `systemActions` property of your component if your entity supports the incrementing or decrementing of its value, or supports activation with a gesture other than a standard tap. You don't need to set a system action if you let people interact with the entity using a standard single-tap gesture.

The following example uses the content of a `RealityView` to determine when activation events occur on the view's entities. After subscribing to the view's activation events, the code sets up an asynchronous task to handle incoming events. When a new event occurs, the task executes the custom code to handle a collision.

```
activationSubscription = content.subscribe(to: AccessibilityEvents.Activate.self,
                              on: nil, componentType: nil) { activation in
    Task {
        try handleCollision(for: activation.entity, gameModel: gameModel)
    }
}
```

Add support for Direct Gesture mode

When VoiceOver is active in visionOS, people use hand gestures to navigate your app's interface and inspect elements. To prevent your app's code from interfering with VoiceOver interactions, the system doesn't deliver hand input to your app during this time. However, a person can perform a special VoiceOver gesture to enable Direct Gesture mode, which leaves VoiceOver enabled but restores hand input to your app.

Add VoiceOver announcements to your code to communicate the results of meaningful events. VoiceOver speaks these announcements at all times, but they are particularly useful when Direct Gesture mode is on. The following example posts an announcement when a custom gesture causes an interaction with a game piece:

```
AccessibilityNotification.Announcement("Game piece hit").post()
```

Provide alternatives to input that involves physical movement

Reduced mobility can affect a person's ability to interact with your app's content. When designing your app's input model, avoid experiences that require specific body movements or positions. For example, if your app supports custom hand gestures, add menu commands for each gesture so someone can enter them using a keyboard or assistive device.

Some assistive technologies let people interact with your app using only their eyes. Using these technologies they can select, scroll, long press, or drag items in your interface. Even if you support other types of interactions, give people a way to access all of your app's behavior using only these interactions.

Avoid head-anchored content

Some assistive technologies allow people to navigate or view your app's interface using head movements. As the person's head moves, the assistive technology focuses on the item directly in front of them. Content that follows the movements of the person's head interferes with the behavior of these assistive technologies.

When designing your interface, place content in windows or anchor it to locations other than the virtual camera. If you do need head-anchored content, provide an alternative solution when relevant assistive technologies are in use. For example, you might move head-anchored content to an anchor point that doesn't follow the person's head movements.

To determine when to change the anchoring approach for your content, check the `accessibilityPrefersHeadAnchorAlternative` environment variable in SwiftUI, or get the `prefersHeadAnchorAlternative` property. This environment variable is `true` when an assistive technology is in use that conflicts with head-anchored content. Adapt your content to use alternate anchoring mechanisms at that time.

Limit motion effects in your content

Motion effects on any immersive device can be jarring, even for people who aren't sensitive to motion. Limit the use of motion effects that incorporate rapid movement, bouncing or wave-like movement, zooming animations, multi-axis movement, spinning, or rotations. When the person wearing the device is sensitive to motion effects, eliminate the use of these effects altogether.

The Reduce Motion system setting lets you know when to provide alternatives for all of your app's motion effects. Access this setting using the `accessibilityReduceMotion` environment variable in SwiftUI or with the `isReduceMotionEnabled` property in UIKit. When the setting is `true`, provide suitable alternatives for motion effects or eliminate them altogether. For example, show a static snapshot of the ocean instead of a video.

For more information, see Human Interface Guidelines > Motion.

Include captions for audio content

For people who are deaf or hard of hearing, provide high-quality captions for your app's content. Captions are a necessity to some, but are practical for everyone in certain situations. For example, captions are useful to someone watching a video in a noisy environment. Remember to include captions not just for text and dialogue, but also for music and sound effects in your content. For Spatial Audio content, include information in your captions that indicates the direction of various sounds.

AVKit and AVFoundation provide built-in support for displaying captioned content. These frameworks configure the font, size, color, and style of the captions automatically according to the person's accessibility settings. For example, the frameworks adopt the current Dynamic Type setting when displaying text.

If you have a custom video engine, check the `isClosedCaptioningEnabled` accessibility setting to determine when to display captions. To get the correct appearance information for your captioned content, adopt Media Accessibility in your project. This framework provides you with the optimal font, color, and opacity information to apply to captioned text and images.

See Also

Design

- Designing for visionOS

  When people wear Apple Vision Pro, they enter an infinite 3D space where they can engage with your app or game while staying connected to their surroundings.

- Adopting best practices for privacy and user preferences

  Minimize your use of sensitive information and provide a clear statement of what information you do use and how you use it.