Article

Positioning and sizing windows

Influence the initial geometry of windows that your app presents.

Overview

visionOS and macOS enable people to move and resize windows. In some cases, your app can use scene modifiers to influence a window's initial geometry on these platforms, as well as to specify the strategy that the system employs to place minimum and maximum size limitations on a window. This kind of configuration affects both windows and volumes, which are windows with the `volumetric` window style.

Your ability to configure window size and position is subject to the following constraints:

- The system might be unable to fulfill your request. For example, if you specify a default size that's outside the range of the window's resizability, the system clamps the affected dimension to keep it in range.
- Although you can change the window's content, you can't directly manipulate window position or size after the window appears. This ensures that people have full control over their workspace.
- During state restoration, the system restores windows to their previous position and size.

> **Note**
>
> Windows in iPadOS occupy the full screen, or share the screen with another window in Slide Over or Split View. You can't programmatically affect window geometry on that platform.

Specify initial window position

macOS visionOS

In macOS, the first time your app opens a window from a particular scene declaration, the system places the window at the center of the screen by default. For scene types that support multiple simultaneous windows, the system offsets each additional window by a small amount to avoid fully obscuring existing windows.

You can override the default placement of the first window in macOS by applying the `defaultPosition(_:)` scene modifier to indicate where to place the window relative to the screen bounds. For example, you can request that the system place a new window in the bottom trailing corner of the screen.

```
@main
struct MyApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
        .defaultPosition(.bottomTrailing)
    }
}
```

The system aligns the point in the window that corresponds to the specified `UnitPoint` with the point in the screen that corresponds to the same unit point. You can use a built-in unit point, like `bottomTrailing` in the above example, or define a custom one.

You can also use `defaultWindowPlacement(_:)` to place windows.

```
@main
struct MyApp: App {
    var body: some Scene {
        // ...

        Window("Status", id: "status") {
            StatusView()
        }
        .windowResizability(.contentSize)
        .defaultWindowPlacement { content, context in
            let displayBounds = context.defaultDisplay.visibleRect
            let size = content.sizeThatFits(.unspecified)
            let verticalOffset = 140
            // The system places the window 140 points from the bottom of the screer
            let position = CGPoint(
                x: displayBounds.midX - (size.width / 2),
                y: displayBounds.maxY - size.height - verticalOffset)
            return WindowPlacement(position: position, size: size)
        }
    }
}
```

Specify initial window size

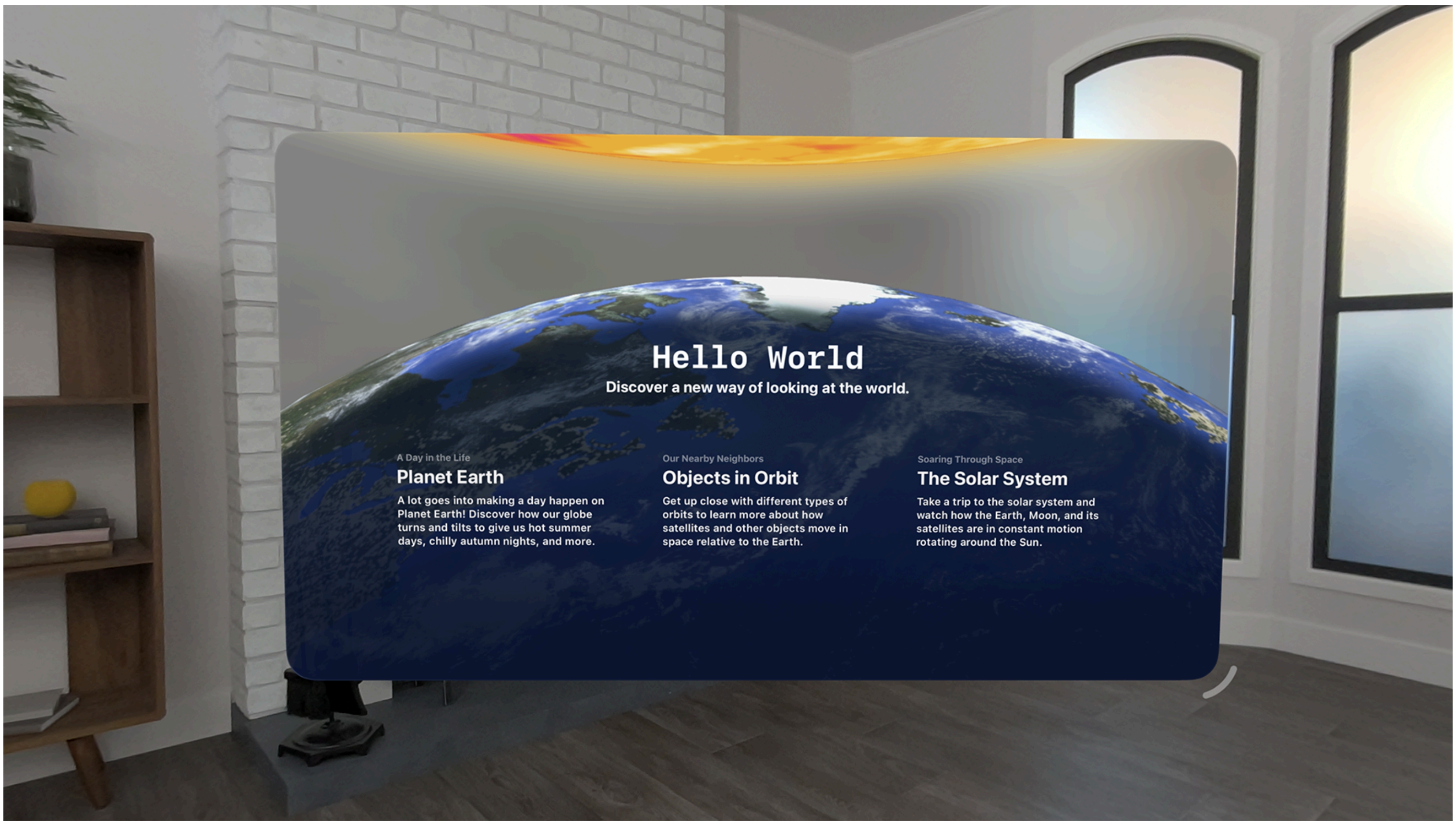
You can indicate a default initial size for a new window that the system creates from a `Scene` declaration by applying one of the default size scene modifiers, like `defaultSize(width:height:)`. For example, you can request that new windows that a `WindowGroup` generates occupy 600 points in the x-dimension and 400 points in the y-dimension.

```
@main
struct MyApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
        .defaultSize(CGSize(width: 600, height: 400))
    }
}
```

The system might clamp the actual size of the window, depending on both the window's content and resizability settings.

Specify window resizability

Both macOS and visionOS provide interface controls that enable people to resize windows within certain limits. For example, people can use the control that appears when they look at the corner of a visionOS window to resize a window on that platform.


Hello World
Discover a new way of looking at the world.
A Day in the Life
Planet Earth
A lot goes into making a day happen on Planet Earth! Discover how our globe turns and tilts to give us hot summer days, chilly autumn nights, and more.
Our Nearby Neighbors
Objects in Orbit
Get up close with different types of orbits to learn more about how satellites and other objects move in space relative to the Earth.
Soaring Through Space
The Solar System
Take a trip to the solar system and watch how the Earth, Moon, and its satellites are in constant motion rotating around the Sun.

Play ▷

You can specify how the system limits window resizability. The default resizability for all scenes is `automatic`. With that strategy, `Settings` windows use the `contentSize` strategy, where both the minimum and maximum window size match the respective minimum and maximum sizes of the content that the window contains. Other scene types use `contentMinSize` by default, which retains the minimum size restriction, but doesn't limit the maximium size.

You can specify one of these resizability strategies explicitly by adding the `windowResizability(_:)` scene modifier to a scene. For example, people can resize windows from the following window group to between 100 and 400 points in both dimensions because the frame modifier imposes those bounds on the content view:

```
@main
struct MyApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
                .frame(
                    minWidth: 100, maxWidth: 400,
                    minHeight: 100, maxHeight: 400)
        }
        .windowResizability(.contentSize)
    }
}
```

You can take this even further and enforce a specific size for a window with content that has a fixed size.

Specify a volume size

When you create a volume, which is a window with the `volumetric` style, you can specify the volume's size using one of the three-dimensional default size modifiers, like `defaultSize(width:height:depth:in:)`. The following code creates a volume that's one meter on a side:

```
WindowGroup(id: "globe") {
    Globe()
}
.windowStyle(.volumetric)
.defaultSize(width: 1, height: 1, depth: 1, in: .meters)
```

Although you can specify a volume's size in points, it's typically better to use physical units, like the above code, which specifies a size in meters. This is because the system renders a volume with fixed scaling rather than dynamic scaling, unlike a regular window, which means the volume appears more like a physical object than a user interface. For information about the different kinds of scaling, see Spatial layout.

See Also

SwiftUI

{} Canyon Crosser: Building a volumetric hike-planning app

Create a hike planning app using SwiftUI and RealityKit.

{} Hello World

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

Presenting windows and spaces

Open and close the scenes that make up your app's interface.

Adopting best practices for persistent UI

Create persistent and contextually relevant spatial experiences by managing scene restoration, customizing window behaviors, and surface snapping data.