Article

Diagnosing issues in the appearance of a running app

Inspect your running app to investigate issues in the appearance and placement of the content it displays.

Overview

At times your app's content may appear to be missing, out of place, or have an appearance that is incorrect. To identify and diagnose the cause of these issues, attach the debugger, reproduce the error, and then narrow down its root cause by inspecting changes in your interface, the code that is executing, and the state of variables. If you configure a scheme's run action for debugging by using the Debug executable checkbox in Info settings, the app will attach to the debugger automatically when the app uses the scheme. To attach the debugger to a process that is already running, choose Debug → Attach to Process, and select your app's process from the list.

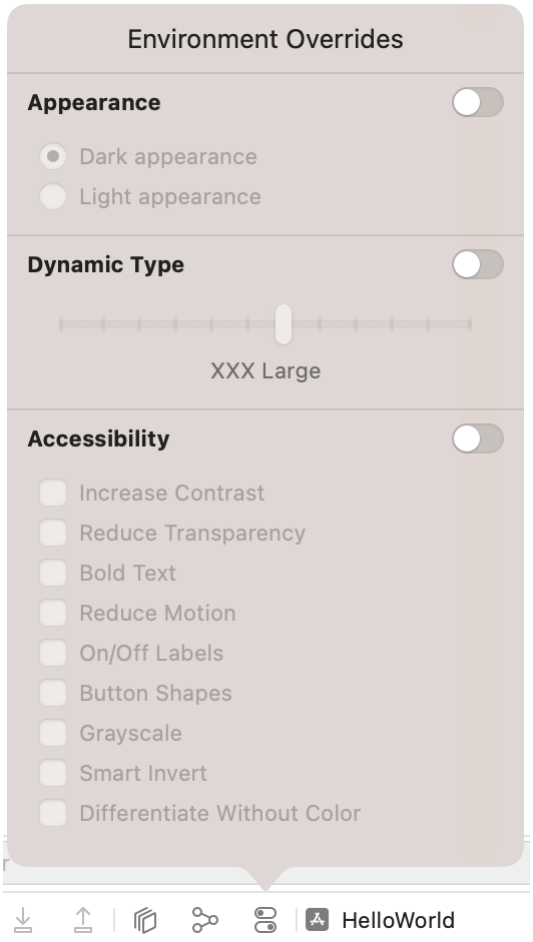
Temporarily override system settings to control your app's appearance and reveal problems that only occur when these settings are in effect, understand layout issues by visualize your views in stacked layers, and debug content in immersive space by adding visual overlays.

Adjust system configurations to identify their impact on your views

Some visual issues only arise when you configure the system using specific environmental settings. Xcode provides environmental overrides when targeting iOS, iPadOS, macOS, and tvOS apps to help you debug these issue. Use these environmental overrides to change the interface style, dynamic text size, and to induce the effects of other accessibility options, so you can understand the effect of these changes on the layout and visual appearance of your views.

To enable one or more of these overrides, click the Environment Overrides button on Xcode's debugging toolbar, toggle the switch next to the override category, and configure the controls

under the category heading.



Appearance
: View your app's content with a light or dark appearance. Select between light and dark appearances using the radio button.

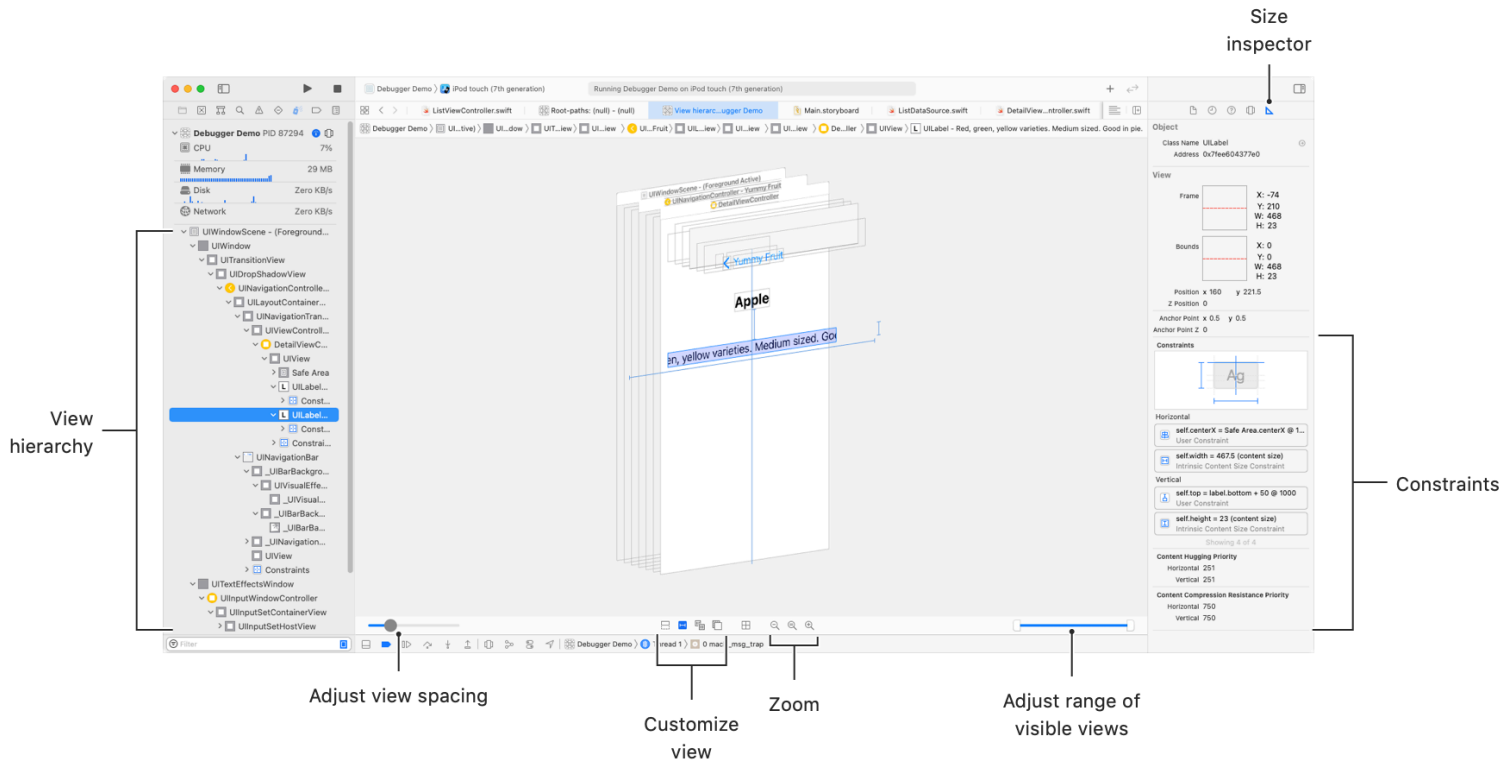
Dynamic Type
: View your app's content using different dynamic type sizes. Select the size using the slider. For more information on dynamic types, see `DynamicTypeSize`.

Accessibility
: View the effects various accessibility features have on your app's content. Click the checkboxes to toggle the accessibility features.

Resolve issues in the layout of your UIKit and SwiftUI views

Use the View debugger, available when targeting iOS, iPadOS, macOS, tvOS, and watchOS apps, to diagnose the reasons an item in your user interface is in the wrong position or is the wrong size. Set a breakpoint in your app after it presents the view, for example, in a `viewDidAppear:` method, then click the Debug View Hierarchy button in the debug bar when the debugger pauses on your breakpoint. Alternatively, just click the Debug View Hierarchy button after your app presents the view.

The debugger displays a 3D rendering of the current view on the canvas in the center, with a representation of the view hierarchy in the Debug navigator. Drag the view in any direction to see a 3D representation of the current view stack, and use the controls at the bottom of the canvas to adjust the views and the spacing between them.



Click to select a view in either the visual rendering or the view hierarchy in the Debug navigator, then inspect details in the Object inspector or Size inspector. Resolve your layout issue according to the type of layout:

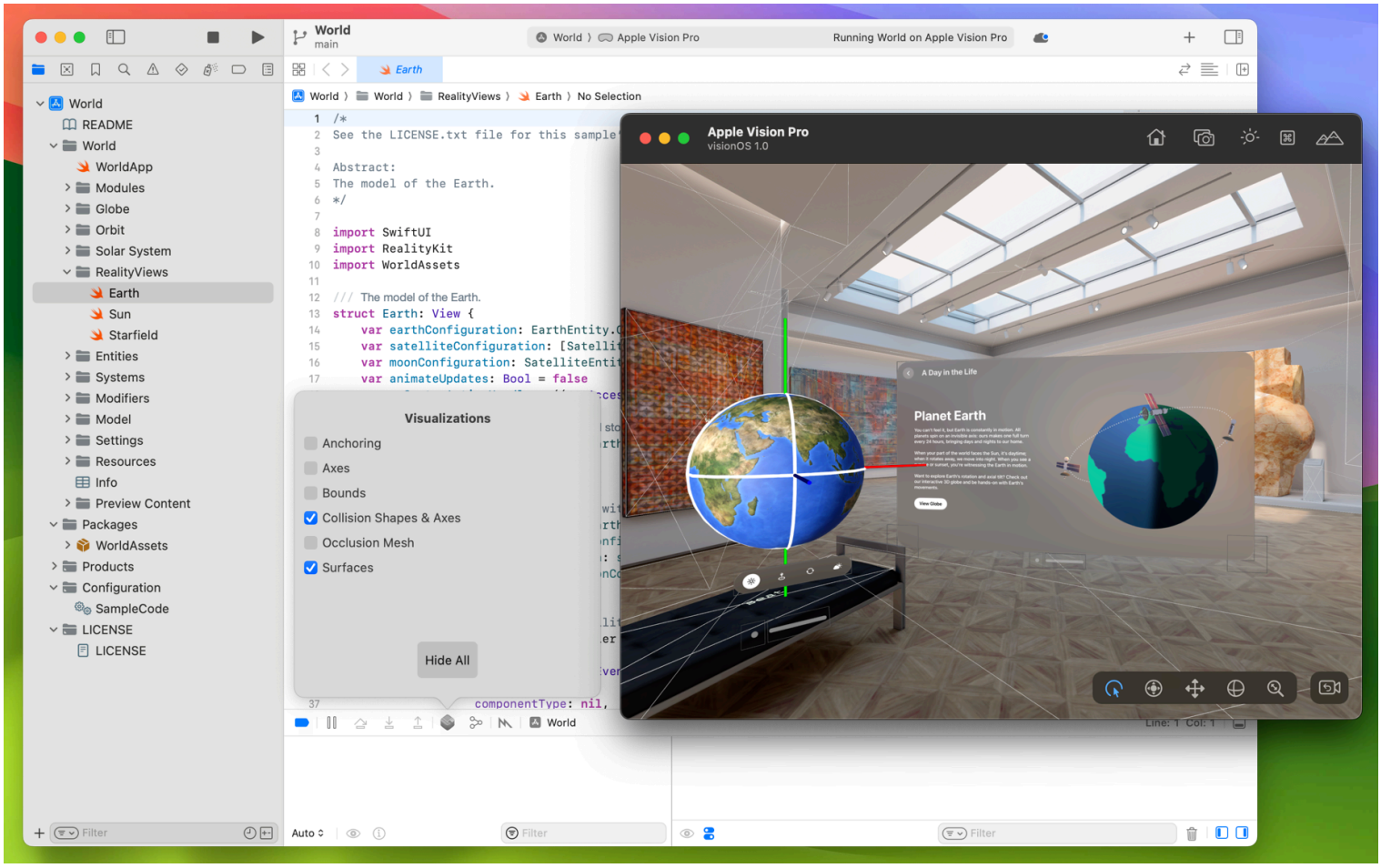
Frame-based layouts
: The view debugger shows the frames you specify in the Size inspector. If the size isn't what you expect, step through your code to diagnose issues with your frame calculations. Then fix and retest.

Views using Auto Layout
: The Size inspector shows constraints. Click a constraint to highlight it in the view debugger. Analyze the constraints that affect the misplaced or mis-sized view to diagnose the issue. Adjust your constraints, either in your code or in Interface Builder, and retest.

Views built with SwiftUI
: The Size inspector shows how SwiftUI resolves the size and placement of your view. With that information, analyze and adjust your SwiftUI code and retest.

Understand the relationships between objects in a immersive space

For visionOS apps with content in an immersive space, it's often helpful to see a visual representation of coordinate axes, bounding boxes, and other information that is normally invisible. Xcode's debugging tools include options to display this information in Simulator or on a device. Use them to ensure that your entities are located where you expect, and interacting with each other and the surroundings the way you anticipate. For example, if an entity isn't responding to events, enable Collision Shapes to confirm the presence of one, required for event handling, and indicate its boundary.



To annotate your content with an overlay, click the Debug Visualizations button on Xcode's debugging toolbar and select one or more of the options:

Anchoring
: Red, green and blue arrows indicating the x, y, and z axes of each anchor point and a yellow line betwen entities and their anchors. Use this to understand the placement of entities in relation to other real-world objects in the space.

Axes
: Red, green and blue arrows indicating the x, y, and z axes of each object in the space. Use this to understand the orientation of each object.

Bounds
: Green lines that indicate the bounding edge of each entity.

Collision Shapes and Axes

A gray outline around the collision shape of an entity and red, green, and blue arrows indicating the axes of the entity. Use this to understand issues in event detection.

Occlusion Mesh
: A multi-colored wireframe around real objects in the space. Use this to identify areas where virtual objects are correctly or incorrectly being hidden behind real objects.

Surfaces
: White border lines and diagonals marking each surface. Use this to understand the borders of surfaces that the system detects.

> **Note**
>
> The Anchoring, Axes, and Bounds options apply to the entities for the app you are debugging while the Collision Shapes & Axes option applies to all entities in Shared Space. The Occlusion Mesh and Surfaces options apply to objects the system detects in your surroundings.

See Also

Debugging strategies

- Diagnosing memory, thread, and crash issues early

  Identify runtime crashes and undefined behaviors in your app during testing using Xcode's sanitizer tools.

- Analyzing HTTP traffic with Instruments

  Measure HTTP-based network performance and usage of your apps.

- Detecting when your app contacts domains that may be profiling users

  Use Instruments to assess whether your app or its third-party SDKs connect to domains that may profile users.