Article

Analyzing the performance of your visionOS app

Use the RealityKit Trace template in Instruments to evaluate and improve the performance of your visionOS app.

Overview

To maintain the sense of immersion on Apple Vision Pro, the system attempts to provide the device displays with up-to-date imagery at a constant rate and respond to interactions with minimum latency. Any visual choppiness or delay in responsiveness interferes with the spatial experience. Higher power consumption over extended periods of time, or extreme power consumption over shorter periods of time, can trigger thermal mitigations that also impact the quality of the experience. It's important to minimize your app's use of system resources to ensure your app performs well on the platform. Many of the same best practices and optimization procedures you use developing for other Apple platforms apply when developing for visionOS as well. For more information about optimizing your app on other platforms, see Improving your app's performance.

To get useful information specific to rendering bottlenecks, high system power use, and other issues that effect the responsiveness of your visionOS app, profile your app with the RealityKit Trace template in Instruments. This template helps you identify:

- Complex content or content with frequent updates that cause the render server to miss deadlines and drop frames.
- Content and tasks that result in high system power use.
- Long running tasks on the the main thread that interfere with efficient processing of input events.
- Tasks running on other threads that don't complete in time to sync back to the main thread for view hierarchy updates.

Note

You can profile using a real device or a simulator, but to get the most accurate and actionable information, use a real device. Software and hardware differences between a simulator on your Mac and a real device prevent you from relying on timing information. Simulated devices are useful for quick iteration and improving performance aspects that aren't based on time.

Open a new trace document

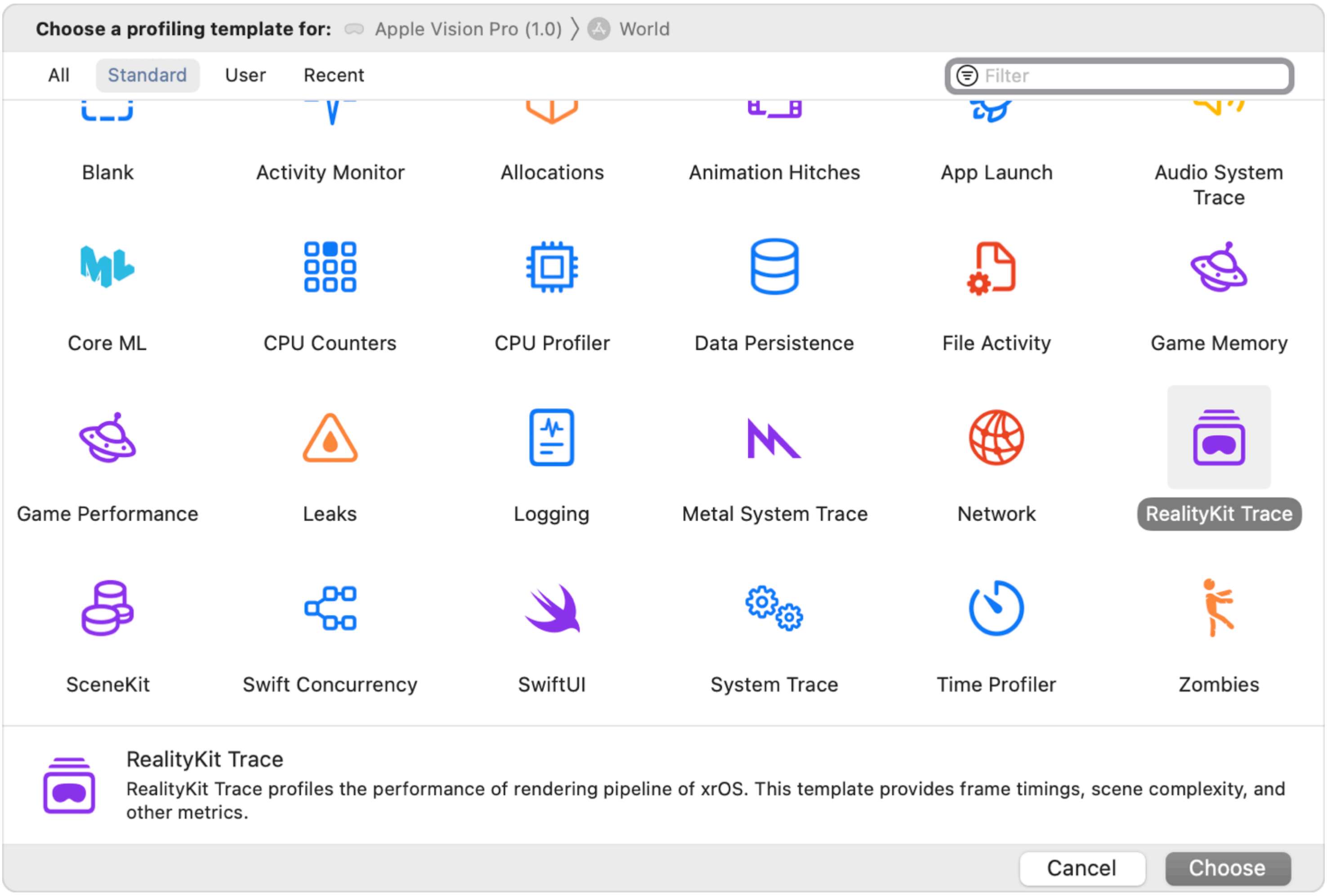
To create a new trace document:

1. Select your app's scheme and a visionOS run destination from the Xcode project window.

2. Choose Product > Profile.

3. Choose RealityKit Trace template

4. Select the Choose button.



RealityKit Trace

RealityKit Trace profiles the performance of rendering pipeline of xrOS. This template provides frame timings, scene complexity, and other metrics.

Alternatively, launch Instruments and choose a target app from the template selection dialog.

The RealityKit Trace template includes the following instruments:

RealityKit Frames
: Captures frame render times and lifespans for frames the visionOS render server generates. This instrument indicates when frames miss rendering deadlines and provides average CPU and GPU render rates.

RealityKit Metrics
: Captures comprehensive timing information from the entire render pipeline including rendering, commits, animations, physics, and spatial systems. This instrument identifies potential bottlenecks in your app’s process or in the render server as a result of your app’s content and indicates areas of moderate and high system power usage that require optimization.

Runloops
: Captures and displays Runloop execution details.

Time Profiler
: Profiles running threads on all cores at regular intervals for all processes.

Hangs
: Captures and displays periods of time when the main thread is unresponsive.

Metal Application
: Records Metal app events.

Consider adding other instruments to your trace for specific investigations. For example, you can use the Thermal State instrument to record device thermal states to check if thermal pressures are throttling performance.

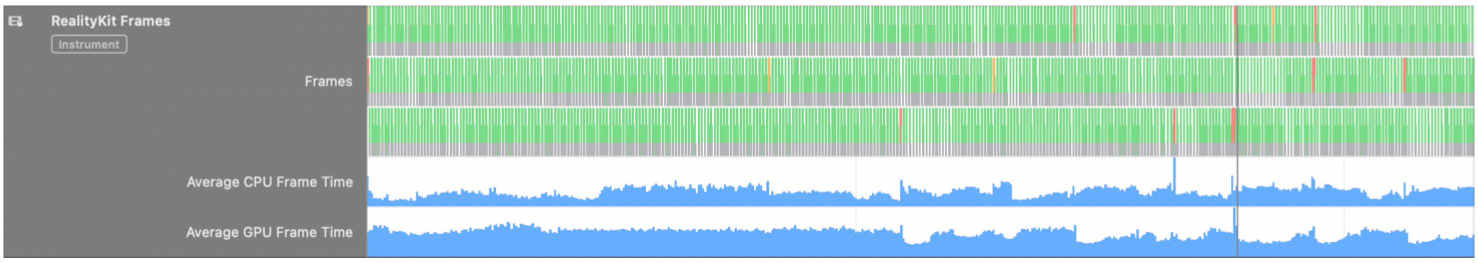
Profile your workflows

Click the record button at the top left of the window to start capturing profile data. Perform the actions in your app that you want to investigate. When you complete the actions, click the record button again to stop recording.

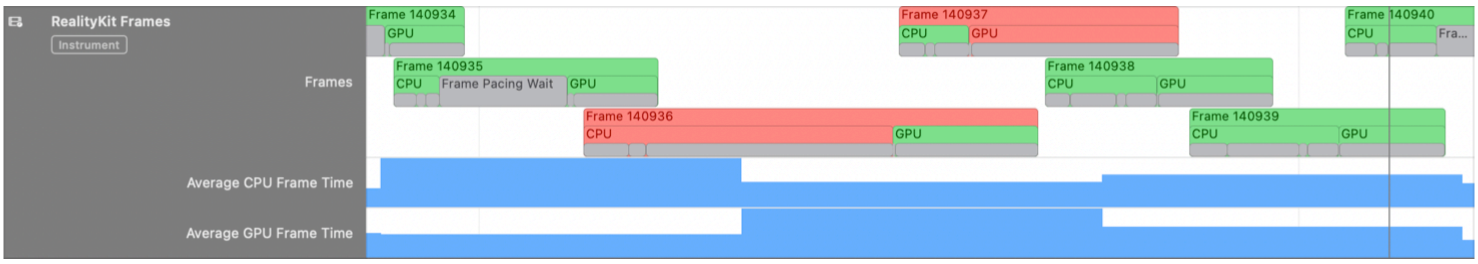
To investigate performance issues or analyze system power impact, profile your app in isolation to understand your app’s impact on system performance and ensure you get the most actionable information. For apps that run alongside other apps, profile your app again with those other apps running to understand how people experience your app in conjunction with other apps.

Inspect frame rendering performance

To maintain a smooth visual experience, the system tries to render new frames for the Apple Vision Pro at 90 frames per second (FPS). The system renders at other frame rates depending on the content it displays and the current surroundings. Each frame has a deadline for rendering based on the target frame rate. Not meeting these deadlines results in dropped frames. This creates a poor spatial experience overall. People tend to notice it in the visual performance of Persona and SharePlay experiences, video playback, and scrolling. The RealityKit Frames instrument displays the time spent rendering each frame in the Frames section of its timeline:



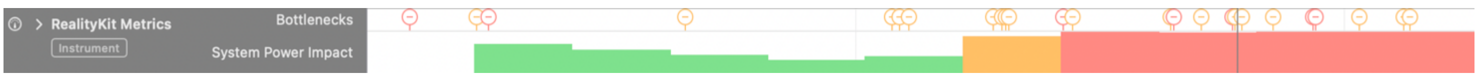
When you zoom out, you can identify areas with a high number of frame drops or with frames running close to the rendering deadline. The timeline uses green to identify frames that complete rendering before the deadline, orange for frames that complete rendering close to the deadline, and red for frames that don’t complete rendering that the renderer drops. Dropped frames contribute to a poor spatial experience, but frames that complete close to their rendering deadline indicate performance problems too. Hold the Option key and drag to zoom into a frame, or group of frames, to see their lifespan broken down in stages:



This provides you with insight into which portion of the rendering pipeline to investigate further. This timeline also includes sections that visualize the Average CPU Frame Time and Average GPU Frame Time to indicate the type of processing that computes the frames. A region of the timeline without a frame block indicates a period of time without changes to a person’s surroundings or app updates. The render server avoids computing new frames to send to the compositor during these periods which helps optimize power use.

Monitor system power usage

When thermal levels rise to levels that trigger thermal mitigations in the system, performance degrades and negatively impacts the responsiveness of your app. Optimize for power to avoid this negative impact. The timeline for the RealityKit Metrics instrument includes a System Power Impact section to identify areas of high power usage in your app:



If the timeline displays green, the tool considers your app’s impact on system power low enough to sustain. Regions that display orange or red indicate the system power usage could cause thermal levels to rise and trigger thermal mitigations. This decreases the availability of system resources, which can cause visual interruptions and responsiveness issues.

> **Note**
>
> If the render server can’t maintain the target frame rate of 90 FPS due to thermal pressure, it might reduce its frame rate in half. When this occurs, all frames in the frames track show up as missing their rendering deadlines. Other factors can cause reduced frame rate, including the complexity and frequency of the content the system is processing. Use the Thermal State instrument to determine if thermal conditions are causing the rate limiting or if it’s due to other factors.

Identify bottlenecks

The Bottlenecks section of the timeline for the RealityKit Metrics instrument contains markers that indicate high overhead in your app or the render server that contribute to dropped frames and high system power use. When you encounter either of these issues, check if the timeline identifies bottlenecks you can address. Double-click on any of the markers to display more information in the detail area at the bottom of the instruments window. If the detail area is hidden, choose View > Detail Area > Show Detail Area to reveal it. The render server encounters bottlenecks in either the CPU or GPU. The instrument categorizes bottlenecks by their severity and type.

To filter the bottlenecks listed in the detail area to a particular time period, drag inside the timeline to select the region. To see an outline view of the bottlenecks organized by severity and type, select Summary: RealityKit Bottlenecks from the menu at the top left of the detail area. Click the arrow button to the right of the severity or type in the outline view to show the list of bottlenecks in that category.

When you select a specific bottleneck, the extended detail provides recommendations for you to address the bottleneck – choose View > Show Extended Detail to reveal the extended detail if it’s hidden.

Explore the metrics that relate to bottlenecks

The trace provides additional information you can use to identify changes to make in your app to address these bottlenecks. Click the expansion arrow for the RealityKit Metrics instrument timeline to reveal graphs specific to each major category of work. Use the metrics associated with these graphs to determine which RealityKit feature has the biggest impact on high CPU frame times in the app process or in the render server. When interpreting these graphs, lower indicates better performance and power. The metrics represent values from all apps running, so profile with just your app running when trying to optimize for these metrics.

3D Render
: Metrics related to the cost of 3D RealityKit rendering in the render server. This includes the number of draw calls, triangles, and vertices from all apps.

Core Animation Render
: Metrics related to UI content rendering costs in the render server. This includes the total number of render passes, offscreen render passes, and translucent UI meshes from all apps.

Entity Commits
: Metrics related to the costs of entity commits in the app and the render server. This includes the number of RealityKit entities shared with the render server from all apps, as well as the number of updates received from all apps over certain intervals.

RealityKit Animations
: Metrics related to the cost of RealityKit animations in the app and the render server. This includes the number of skeletal animations, across all apps.

RealityKit Physics
: Metrics related to the cost of RealityKit physics simulations, collisions, and hit testing in the app process and render server. This includes the number of rigid body counts and colliders in use, as well as the type of physics shapes that the UI and other 3D content use, across all apps.

Spatial Systems
: Metrics related to the costs of spatial algorithms in the render server. This includes the number of custom anchors, across all apps.

> **Tip**
>
> The graphs for some sections combine several individual metrics. The heading indicates this by displaying a graph count. Click on the bottom of the timeline's heading and drag down to display individual graphs for each metric. For example, the 3D Render Timeline might display 13 Graphs in the heading; expanding that timeline exposes individual graphs for 3D Mesh Draw Calls, 3D Mesh Triangles, 3D Mesh Vertices, and the 10 additional metrics.

The timeline for your app's process helps summarize information from the instruments about your process and the work the render server completes for your process.

Choose an option from the pop-up in the timeline header to show different graphs in the timeline:

Runloops
: Time each thread spends waiting or busy.

Hangs
: Time the main thread is unresponsive.

Time Profile
: CPU usage and lifecycle status.

RealityKit System Times
: Overhead attributed to RealityKit systems.

When you select the timeline for your app’s process, you can choose instrument summaries and profile data to display in the detail area from the popup-button at its top-left:

To filter the information in the detail area by time, select periods of time in the timeline above.

Detect delays on the main thread

Select Hangs in your app’s process timeline to identify times in the trace that might have interaction delays. Use the RealityKit Metrics and Time Profiler summaries to better understand the work your app is doing. Choose the following options from the detail area pop-up menu:

Profile and Samples
: Shows information from the Time Profiler instrument to determine what your app is doing during a hang.

Summary
: RealityKit System CPU times: Shows minimum, maximum, and average times the CPU spends on various RealityKit system operations.

Optimize any 3D render updates, hit testing, and collision work you find. For more information about addressing hangs in your app, see Improving app responsiveness.

Manage audio overhead

Use the Audio Playback section of your process’s timeline to identify areas of high audio overhead. The system defaults to using spatial audio for your app when running on visionOS. It processes information in real time about your position, surroundings, and the current location of audio sources to generate an immersive audio experience. If you include too many concurrent audio sources that require the system to adapt audio sources to their location within a large space, the increased demand on system resources can lead to delays in the audio output.

To reduce the spatial audio work, limit:

- The number of concurrently playing audio sources
- The number of moving audio sources
- The size of the soundstage

Consider creating a pool of audio players to limit the maximum number of players your app uses. Place players on stationary entities, instead of moving entities, when appropriate. Initializing several audio players at the same time causes a high overhead that affects other aspects of the system, such as rendering performance. Consider the other tasks the system completes during these allocations and space them out over time. For more information, see Create a great spatial playback experience.

Profile custom materials for optimization

Use the Metal System Trace template to profile your custom materials in isolation before adding more visual effects to them. Examine the Metal GPU cost and try to optimize for GPU ALU Instructions and GPU Texture reads and writes per frame. For more information on using this template, see Analyzing the performance of your Metal app.

In order to use the Metal Application instrument to profile your custom materials accurately, set a fixed performance state:

1. Click and hold the record button and select Recording Options.
2. Select the options for the Metal Application instrument.
3. Set the Counter Set to Performance Limiters.
4. Set the Performance State to Minimum or Medium.

A person’s eye position, the distance from the person to the content, and the amount of content the app displays all impact the amount of GPU work done for custom materials from Reality Composer Pro. To compare traces, keep these factors as consistent as possible. For a custom material you use in a Fully Immersive Space, try to remove all other content from the scene while profiling to isolate the overhead of the material. For custom materials you use for other types of content, consider anchoring them to a person’s head in a test scene so the distance from the person remains fixed.

> Note
>
> Using an unlit or inexpensive custom material becomes even more important when your app uses Metal and the Compositor Services framework to present a fully immersive experience due to the number of pixels your app must render.

See Also

Performance

- Creating a performance plan for your visionOS app

  Identify your app’s performance and power goals and create a plan to measure and assess them.

- Reducing the rendering cost of your UI on visionOS

  Optimize your 2D user interface rendering on visionOS.

- Reducing the rendering cost of RealityKit content on visionOS

  Optimize your app’s 3D augmented reality content to render efficiently on visionOS.

- Understanding the visionOS render pipeline

  Compare how visionOS handles events and manages its rendering loop differently from other Apple platforms.