

[Xcode](#) / [Debugging](#) / Metal debugger

Metal debugger

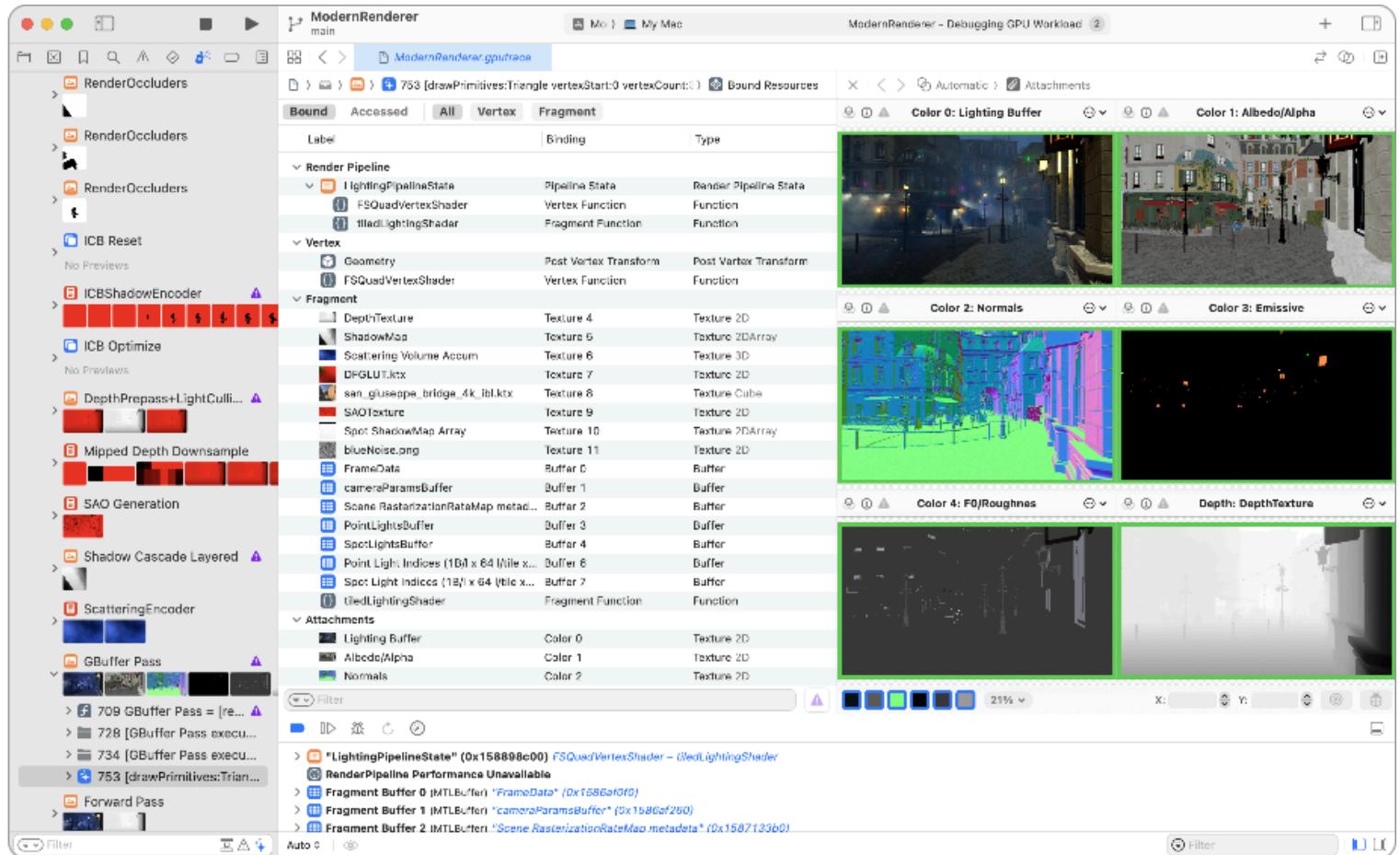
Debug and profile your Metal workload with a GPU trace.

Overview

The Metal debugger consists of a suite of tools for debugging and profiling your Metal app.

Unlike pausing at breakpoints during runtime, you can capture your Metal workload for multiple frames and then jump back and forth in time to explore the captured work. The Metal debugger enables you to explore the dependencies between passes, and offers insights for improving the performance of your app. You can also debug your shaders in draw commands and compute dispatches to fix sources of artifacts (see [Investigating visual artifacts](#)).

In addition, the Metal debugger displays your Metal workload on a profiling timeline and offers detailed statistics like performance counters and per-line shader profiling data. These tools can help you identify and eliminate performance bottlenecks in your app (see [Optimizing GPU performance](#)).



For additional information about the Metal debugger, see the following video sessions:

- [Metal Shader Debugging and Profiling](#)
- [Gain insights into your Metal app with Xcode 12](#)
- [Optimize Metal apps and games with GPU counters](#)
- [Discover Metal debugging, profiling, and asset creation tools](#)
- [Profile and optimize your game's memory](#)

Topics

Essentials

- 📄 **Capturing a Metal workload in Xcode**
Analyze your app's performance by configuring your project to use the Metal debugger.
- 📄 **Capturing a Metal workload programmatically**
Analyze your app's performance by invoking Metal's frame capture.
- 📄 **Replaying a GPU trace file**
Debug and profile your app's performance using a GPU trace file in the Metal debugger.

- Investigating visual artifacts
Discover, diagnose, and fix visual artifacts in your app with the Metal debugger.
- Optimizing GPU performance
Find and address performance bottlenecks using the Metal debugger.

Metal workload analysis

- Analyzing your Metal workload
Investigate your app's workload, dependencies, performance, and memory impact using the Metal debugger.
- Analyzing resource dependencies
Avoid unnecessary work in your Metal app by understanding the relationships between resources.
- Analyzing memory usage
Manage your Metal app's memory usage by inspecting its resources.
- Analyzing Apple GPU performance using a visual timeline
Locate performance issues using the Performance timeline.
- Analyzing Apple GPU performance using counter statistics
Optimize performance by examining counters for individual passes and commands.
- Analyzing Apple GPU performance with performance heat maps
Gain insights to SIMD group performance by inspecting source code execution.
- Analyzing Apple GPU performance using the shader cost graph
Discover potential shader performance issues by examining pipeline states.
- Analyzing non-Apple GPU performance using counter statistics
Optimize performance by examining counters for individual passes and commands.

Metal resource inspection

- Inspecting acceleration structures
Reveal ray intersection bottlenecks by examining your acceleration structures.
- Inspecting buffers
Confirm your buffer formats by examining buffer content.

- 📄 Inspecting pipeline states
Determine how your render and compute passes behave by examining their properties.
- 📄 Inspecting sampler states
Verify your sampler state configurations by examining their properties.
- 📄 Inspecting shaders
Improve your app's shader performance by examining and editing your shaders.
- 📄 Inspecting textures
Discover issues in your textures by examining their content.

Metal command analysis

- 📄 Inspecting the bound resources for a command
Discover issues by examining the bound resources at any point in an encoder.
- 📄 Inspecting the geometry of a draw command
Find problems in your app's vertex, object, or mesh function by examining the current geometry.
- 📄 Inspecting the attachments of a draw command
Discover attachment issues by inspecting individual pixels and samples.
- 📄 Debugging the shaders within a draw command or compute dispatch
Identify and fix problematic shaders in your app using the shader debugger.
- 📄 Analyzing draw command and compute dispatch performance with GPU counters
Identify issues within your frame capture by examining performance counters.
- 📄 Analyzing draw command and compute dispatch performance with pipeline statistics
Identify issues within your frame capture by examining pipeline statistics.

See Also

Graphics

- ☰ Metal developer workflows

Locate and fix issues related to your app's use of the Metal API and GPU functions.