Metal developer workflows

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

Overview

Metal comes with a comprehensive suite of advanced developer tools to help you debug and optimize your Metal apps.

Runtime diagnostics

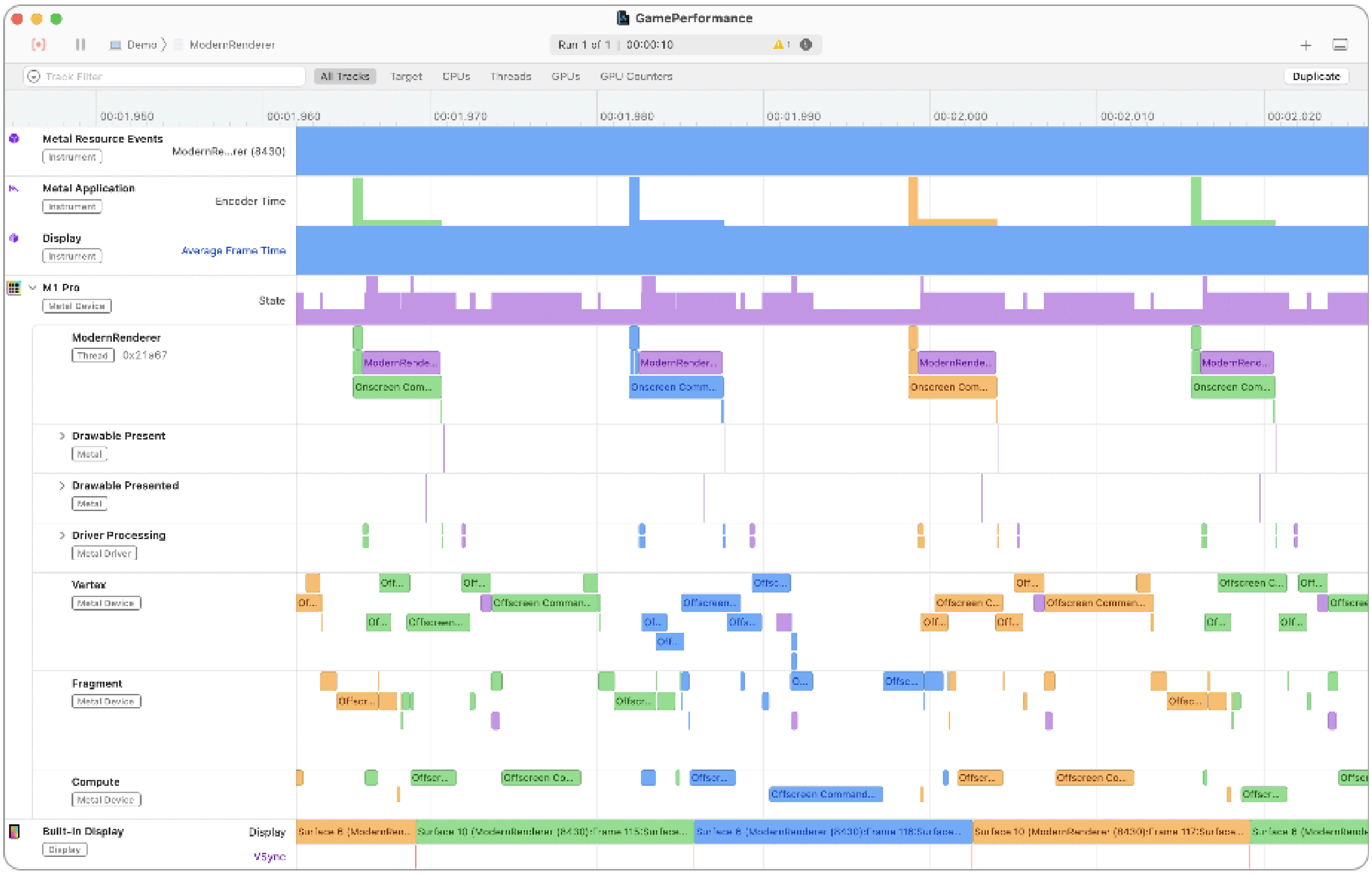
You can enable API Validation when running your app to check for incorrect Metal API usage. For more information, see Validating your app’s Metal API usage.

Enable Shader Validation when running your app to check for issues like out-of-bounds memory access, missing `useResource` calls, and stack overflows. For more information, see Validating your app’s Metal shader usage.

The Metal Performance HUD offers a visual overlay to catch performance issues while your app is running. For more information, see Monitoring your Metal app’s graphics performance.

Runtime performance analysis

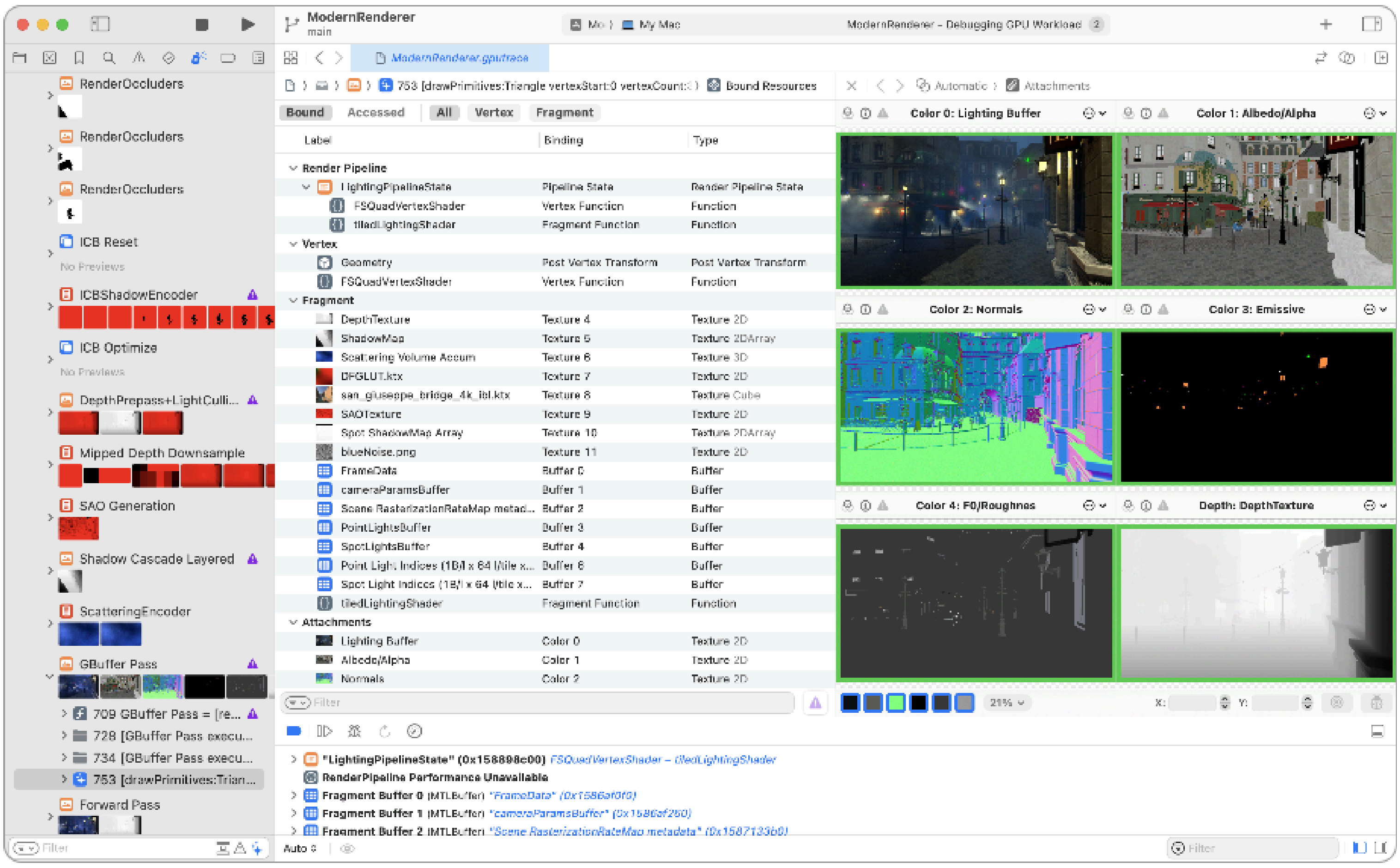
The Metal system trace tool in Instruments provides a visual timeline of the parallel work on the CPU and the GPU, and the memory usage of your Metal app.



You can begin profiling with the Game Performance template (see Analyzing the performance of your Metal app) or the Game Memory template (see Analyzing the memory usage of your Metal app).

Advanced Metal debugging and profiling

The Metal debugger in Xcode provides advanced tools for debugging and profiling your Metal app.



Label	Binding	Type
Render Pipeline		
LightingPipelineState	Pipeline State	Render Pipeline State
FSQuadVertexShader	Vertex Function	Function
tiledLightingShader	Fragment Function	Function
Vertex		
Geometry	Post Vertex Transform	Post Vertex Transform
FSQuadVertexShader	Vertex Function	Function
Fragment		
DepthTexture	Texture 4	Texture 2D
ShadowMap	Texture 5	Texture 2DArray
Scattering Volume Accum	Texture 6	Texture 3D
DFGLUT.ktx	Texture 7	Texture 2D
san_giuseppe_bridge_4k_ibl.ktx	Texture 8	Texture Cube
SAOTexture	Texture 9	Texture 2D
Spot ShadowMap Array	Texture 10	Texture 2DArray
blueNoise.png	Texture 11	Texture 2D
FrameData	Buffer 0	Buffer
cameraParamsBuffer	Buffer 1	Buffer
Scene RasterizationRateMap metad...	Buffer 2	Buffer
PointLightsBuffer	Buffer 3	Buffer
SpotLightsBuffer	Buffer 4	Buffer
Point Light Indices (1B/l x 64 l/tile x...	Buffer 6	Buffer
Spot Light Indices (1B/l x 64 l/tile x...	Buffer 7	Buffer
tiledLightingShader	Fragment Function	Function
Attachments		
Lighting Buffer	Color 0	Texture 2D
Albedo/Alpha	Color 1	Texture 2D
Normals	Color 2	Texture 2D

```
> "LightingPipelineState" (0x158898c00) FSQuadVertexShader – tiledLightingShader
  RenderPipeline Performance Unavailable
> Fragment Buffer 0 (MTLBuffer) "FrameData" (0x1586af0f0)
> Fragment Buffer 1 (MTLBuffer) "cameraParamsBuffer" (0x1586af250)
> Fragment Buffer 2 (MTLBuffer) "Scene RasterizationRateMap metadata" (0x1587123b0)
```

You can get summaries of your Metal workload with the Dependencies viewer and the Memory viewer, inspect individual resources, and selectively debug your shaders. For more information on debugging, see Investigating visual artifacts.

In addition, you can optimize your Metal app by drilling down performance bottlenecks with the Performance timeline and the per-line shader profiling results. For more information on profiling, see Optimizing GPU performance.

Learn more about Metal debugger.

Topics

Project preparation for debugging

Make your Metal app easier to debug by taking a few extra steps as you develop your code.

Building your project with embedded shader sources

Prepare to debug your project's shaders by including source code in the build.

Naming resources and commands

Enhance the debugging of your Metal app using labels and grouping.

Creating and using custom capture scopes

Capture specific GPU commands by using custom capture scopes.

Runtime diagnostics

Learn how to use runtime tools to discover potential issues in your Metal app.

- Inspecting live resources at runtime

  Validate your resources by viewing the contents of your textures and buffers while debugging your Metal app.

- Validating your app’s Metal API usage

  Catch runtime issues in your Metal app using API Validation.

- Validating your app’s Metal shader usage

  Catch common shader runtime issues using Shader Validation while your app is running.

- Monitoring your Metal app’s graphics performance

  Catch performance issues using the Metal Performance HUD while your app runs.

- Customizing the Metal Performance HUD

  Modify the appearance of your Metal heads-up display to monitor your graphics performance.

- Understanding the Metal Performance HUD metrics

  Learn what each of the metrics reported by the heads-up display indicates.

- Gaining performance insights with the Metal Performance HUD

  Catch potential performance issues while your app runs using the Metal heads-up display.

- Generating performance reports with the Metal Performance HUD

  Record your app’s performance using the heads-up display.

Counters

Get references for the set of performance metrics available across Metal tools.

- Finding your Metal app’s GPU occupancy

  Understand the GPU usage for executing shaders by using occupancy.

- Reducing shader bottlenecks

  Identify and minimize congestion points in a GPU’s subsystems by checking its limiter and utilization counters.

Measuring the GPU’s use of memory bandwidth

Check whether your Metal app correctly reads and writes to memory by measuring the GPU’s memory bandwidth.

See Also

Graphics

Metal debugger

Debug and profile your Metal workload with a GPU trace.