Article

Analyzing your Metal workload

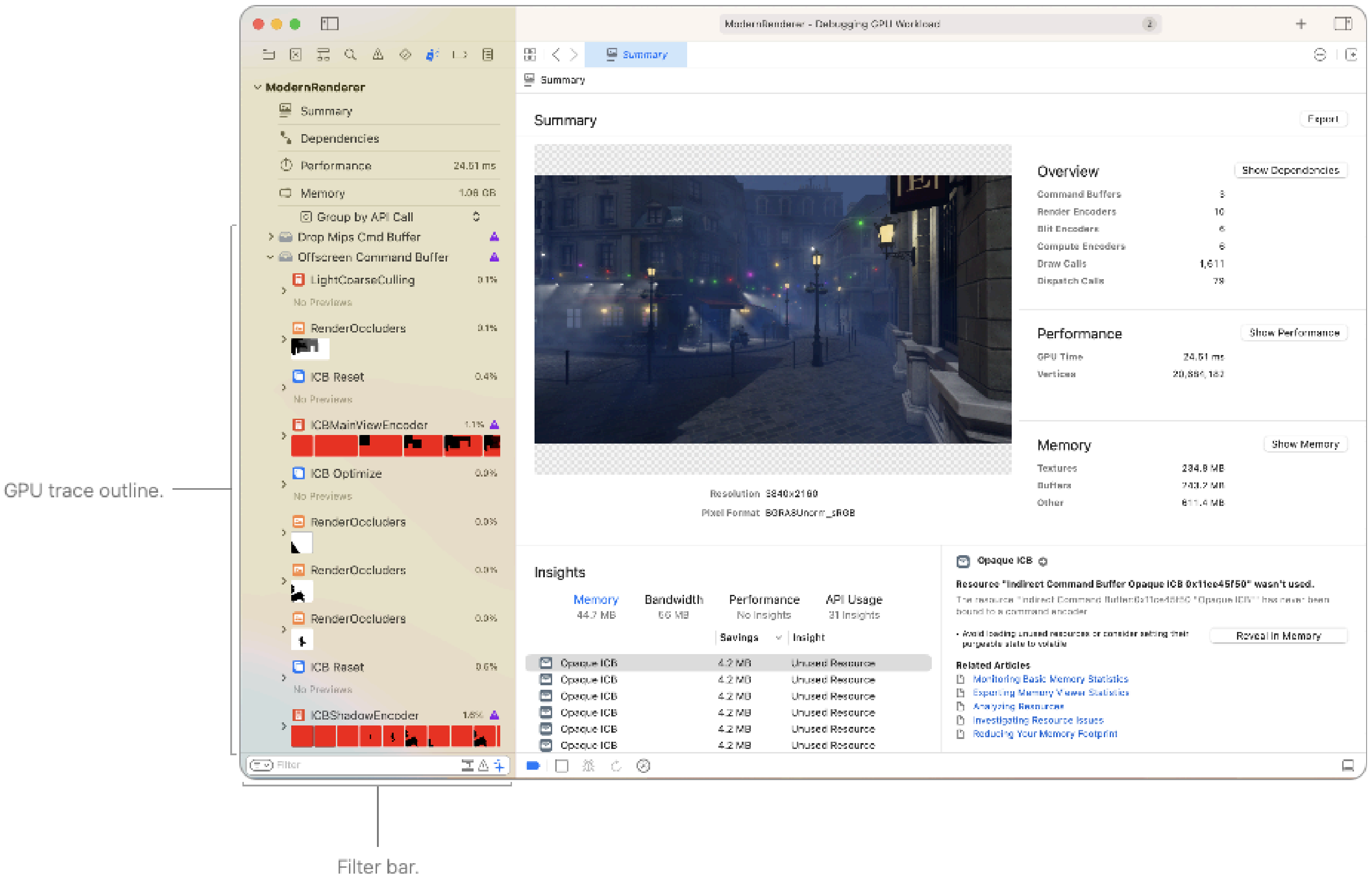
Investigate your app’s workload, dependencies, performance, and memory impact using the Metal debugger.

Overview

The Metal debugger provides useful tools for analyzing many aspects of how your app uses the GPU.

After capturing a Metal workload (see Capturing a Metal workload in Xcode) or replaying a GPU trace (see Replaying a GPU trace file), the Metal debugger presents the Summary viewer. The Summary viewer includes a preview of the last presented drawable in your app on the left, several statistics on the right, and a list of automatically generated recommendations, called Insights, at the bottom.

To the left of the Summary viewer is the Debug navigator, which enables quick access to top-level information about your Metal workload. You can explore all of the GPU commands in your capture, as well as all of the pipeline states it uses.



ModernRenderer
- Summary
- Dependencies
- Performance 24.51 ms
- Memory 1.08 GB
- Group by API Call
- Drop Mips Cmd Buffer
- Offscreen Command Buffer
  - LightCoarseCulling 0.1% — No Previews
  - RenderOccluders 0.1%
  - ICB Reset 0.4% — No Previews
  - ICBMainViewEncoder 1.1%
  - ICB Optimize 0.0% — No Previews
  - RenderOccluders 0.0%
  - RenderOccluders 0.0%
  - RenderOccluders 0.0%
  - ICB Reset 0.0% — No Previews
  - ICBShadowEncoder 1.6%

Summary

Resolution 3840x2160
Pixel Format BGRA8Unorm_sRGB

Overview

Command Buffers	3
Render Encoders	10
Blit Encoders	6
Compute Encoders	6
Draw Calls	1,611
Dispatch Calls	78

Performance

GPU Time	24.51 ms
Vertices	20,884,182

Memory

Textures	234.8 MB
Buffers	243.2 MB
Other	611.4 MB

Insights

Memory	Bandwidth	Performance	API Usage
44.7 MB	66 MB	No Insights	31 Insights

	Savings	Insight
Opaque ICB	4.2 MB	Unused Resource
Opaque ICB	4.2 MB	Unused Resource
Opaque ICB	4.2 MB	Unused Resource
Opaque ICB	4.2 MB	Unused Resource
Opaque ICB	4.2 MB	Unused Resource
Opaque ICB	4.2 MB	Unused Resource

Opaque ICB

Resource "Indirect Command Buffer Opaque ICB 0x11ce45f50" wasn't used.

The resource "Indirect Command Buffer0x11ce45f50 "Opaque ICB"" has never been bound to a command encoder

- Avoid loading unused resources or consider setting their purgeable state to volatile

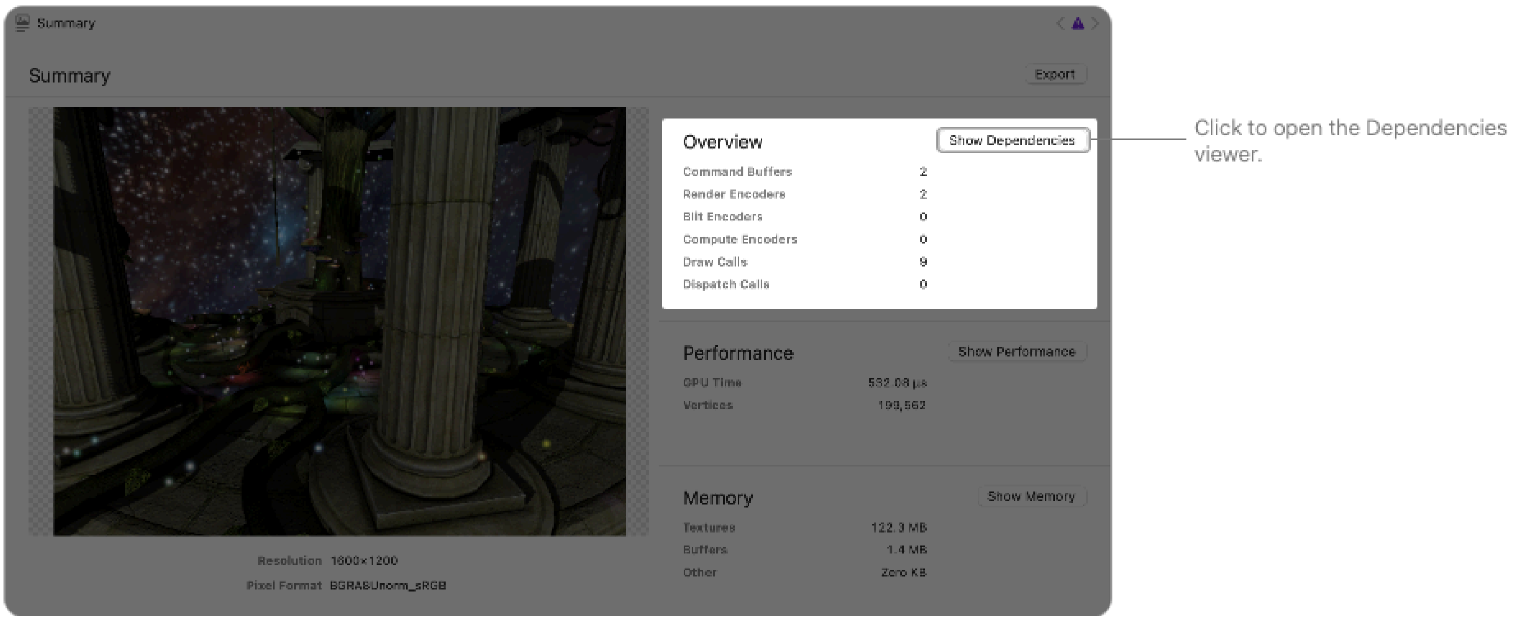
Related Articles

- Monitoring Basic Memory Statistics
- Exporting Memory Viewer Statistics
- Analyzing Resources
- Investigating Resource Issues
- Reducing Your Memory Footprint

View statistics for your Metal workload

The Metal debugger automatically calculates several statistics and displays them on the right side of the Summary viewer.

The Overview section includes the number of command buffers, command encoders, and draw commands or compute dispatches. It takes CPU time to encode commands, and GPU time to execute them. If your app is creating too many command buffers or command encoders that start to have a significant impact on performance, consider reducing the workload. You can click the Show Dependencies button to open the Dependencies viewer and learn how your command encoders connect with one another (see Analyzing resource dependencies).



Summary

Overview

Command Buffers	2
Render Encoders	2
Blit Encoders	0
Compute Encoders	0
Draw Calls	9
Dispatch Calls	0

Performance

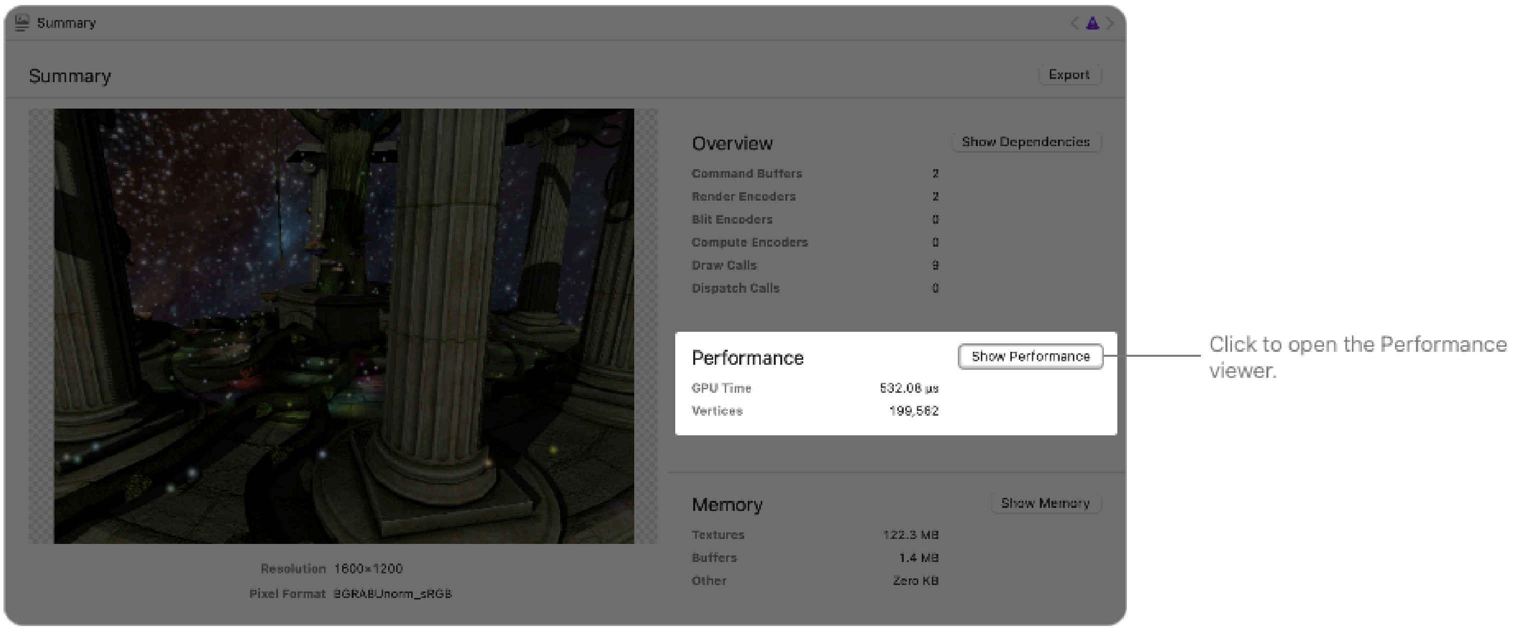
GPU Time	532.08 µs
Vertices	199,562

Memory

Textures	122.3 MB
Buffers	1.4 MB
Other	Zero KB

Resolution 1600×1200
Pixel Format BGRA8Unorm_sRGB

The Performance section includes the total GPU time and the number of vertices. If your workload has a high GPU time, consider optimizing its performance (see Optimizing GPU performance). You can also click the Show Performance button to display the Performance timeline and discover which aspects of your Metal workload are taking the most time (see Analyzing Apple GPU performance using a visual timeline).



Summary

Overview

Command Buffers	2
Render Encoders	2
Blit Encoders	0
Compute Encoders	0
Draw Calls	9
Dispatch Calls	0

Performance

GPU Time	532.08 µs
Vertices	199,562

Memory

Textures	122.3 MB
Buffers	1.4 MB
Other	Zero KB

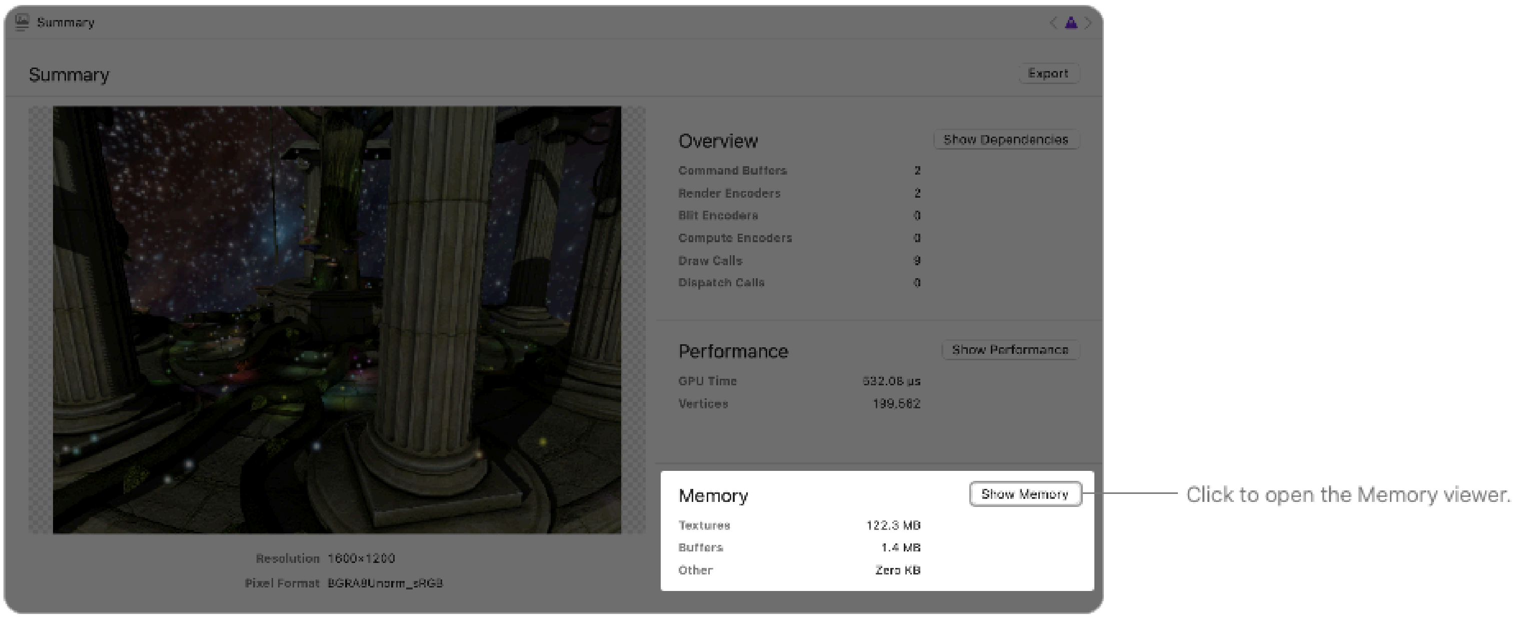
Resolution 1600×1200
Pixel Format BGRA8Unorm_sRGB

Note

If you don't select the Profile after Replay checkbox in the Metal Capture popover (see Capturing a Metal workload in Xcode) or the Profile GPU Trace checkbox in the Replay window (see Replaying a GPU trace file), the performance section doesn't show any statistics. To see the statistics, click the Profile button and wait for profiling to finish.

Performance data not available

The Memory section includes a brief overview of GPU memory for various resource types, such as textures and buffers. If your Metal workload is using a large amount of memory, consider optimizing the memory use. You can also click the Show Memory button to open the Memory viewer and discover which aspects of your Metal workload are using the most memory (see Analyzing memory usage).



Overview

Command Buffers	2
Render Encoders	2
Blit Encoders	0
Compute Encoders	0
Draw Calls	9
Dispatch Calls	0

Performance

GPU Time	532.08 µs
Vertices	199,562

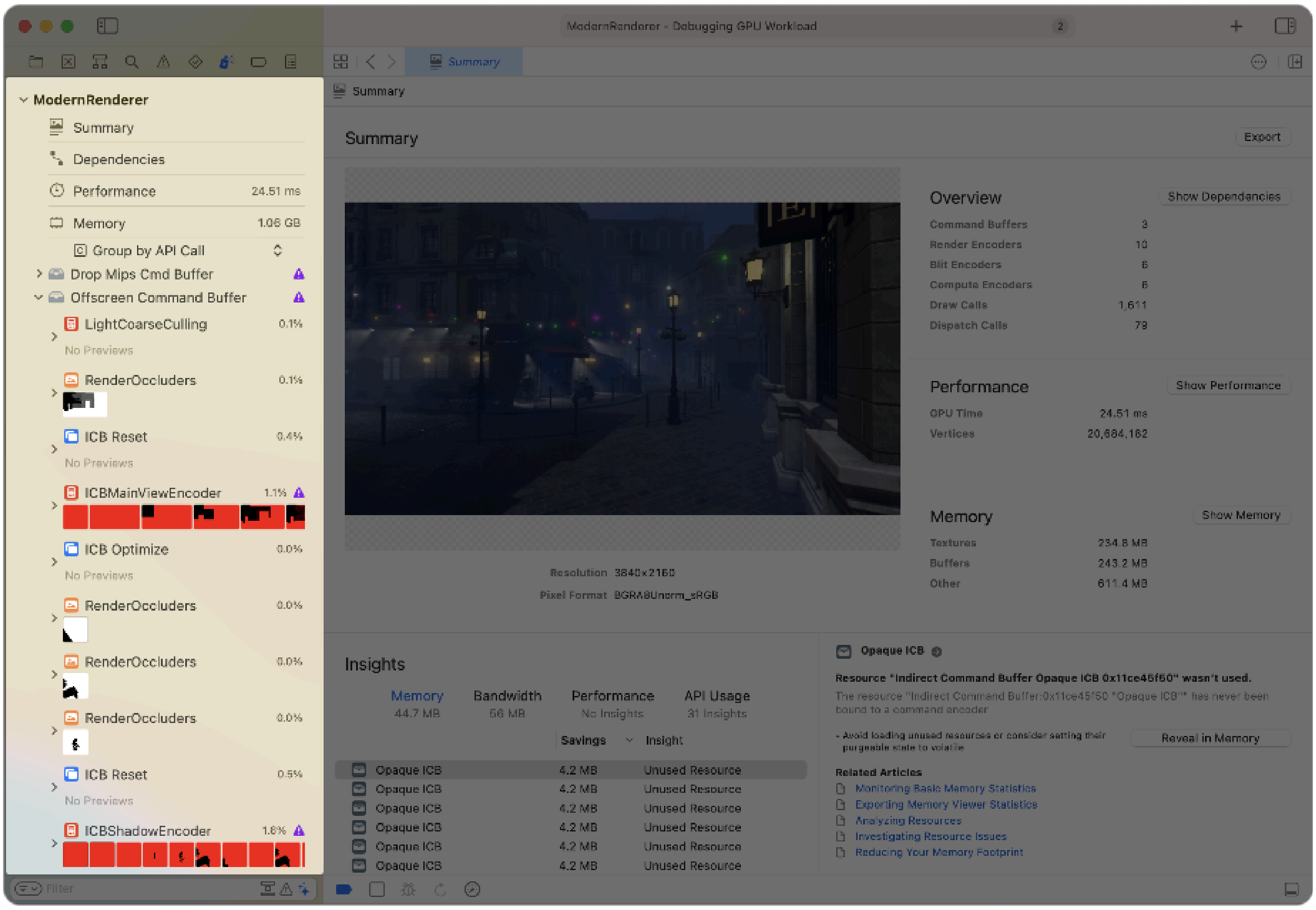
Memory

Textures	122.3 MB
Buffers	1.4 MB
Other	Zero KB

Resolution 1600×1200
Pixel Format BGRA8Unorm_sRGB

Browse API calls

By default, the Metal debugger shows an outline of captured Metal commands — grouped by command buffers, passes, and debug groups — in the Debug navigator. In addition, for render passes, the outline includes a list of render attachment thumbnail images.


ModernRenderer - Debugging GPU Workload

ModernRenderer
- Summary
- Dependencies
- Performance 24.51 ms
- Memory 1.06 GB
- Group by API Call
- Drop Mips Cmd Buffer
- Offscreen Command Buffer
  - LightCoarseCulling 0.1% — No Previews
  - RenderOccluders 0.1%
  - ICB Reset 0.4% — No Previews
  - ICBMainViewEncoder 1.1%
  - ICB Optimize 0.0% — No Previews
  - RenderOccluders 0.0%
  - RenderOccluders 0.0%
  - RenderOccluders 0.0%
  - ICB Reset 0.5% — No Previews
  - ICBShadowEncoder 1.6%

Summary

Resolution 3840x2160
Pixel Format BGRA8Unorm_sRGB

Overview

Command Buffers	3
Render Encoders	10
Blit Encoders	6
Compute Encoders	6
Draw Calls	1,611
Dispatch Calls	79

Performance

GPU Time	24.51 ms
Vertices	20,684,162

Memory

Textures	234.8 MB
Buffers	243.2 MB
Other	611.4 MB

Insights

Memory 44.7 MB · Bandwidth 56 MB · Performance No Insights · API Usage 31 Insights

	Savings	Insight
Opaque ICB	4.2 MB	Unused Resource
Opaque ICB	4.2 MB	Unused Resource
Opaque ICB	4.2 MB	Unused Resource
Opaque ICB	4.2 MB	Unused Resource
Opaque ICB	4.2 MB	Unused Resource
Opaque ICB	4.2 MB	Unused Resource

Opaque ICB

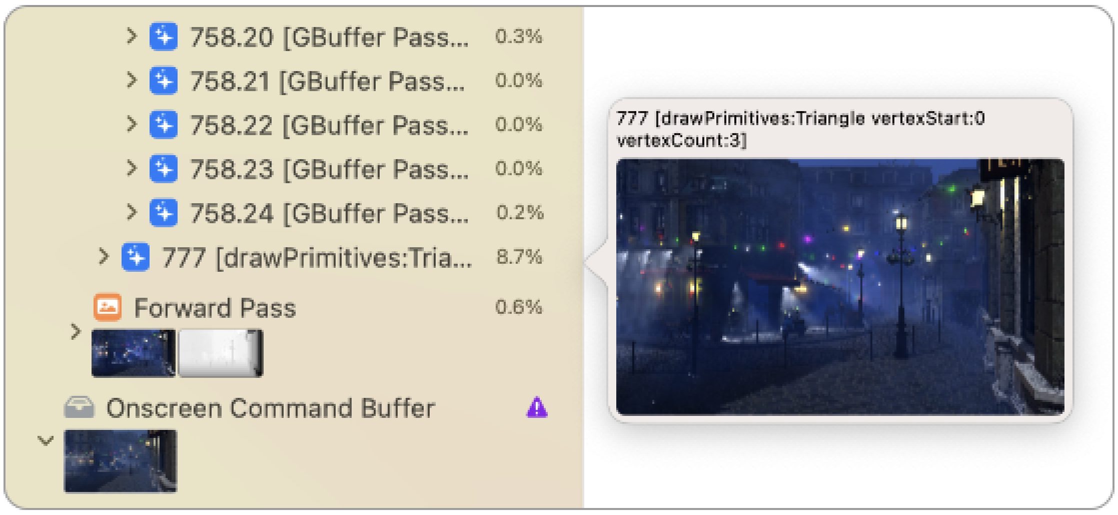
Resource "Indirect Command Buffer Opaque ICB 0x11ce45f50" wasn't used.

The resource "Indirect Command Buffer:0x11ce45f50 "Opaque ICB"" has never been bound to a command encoder

- Avoid loading unused resources or consider setting their purgeable state to volatile

Related Articles
- Monitoring Basic Memory Statistics
- Exporting Memory Viewer Statistics
- Analyzing Resources
- Investigating Resource Issues
- Reducing Your Memory Footprint

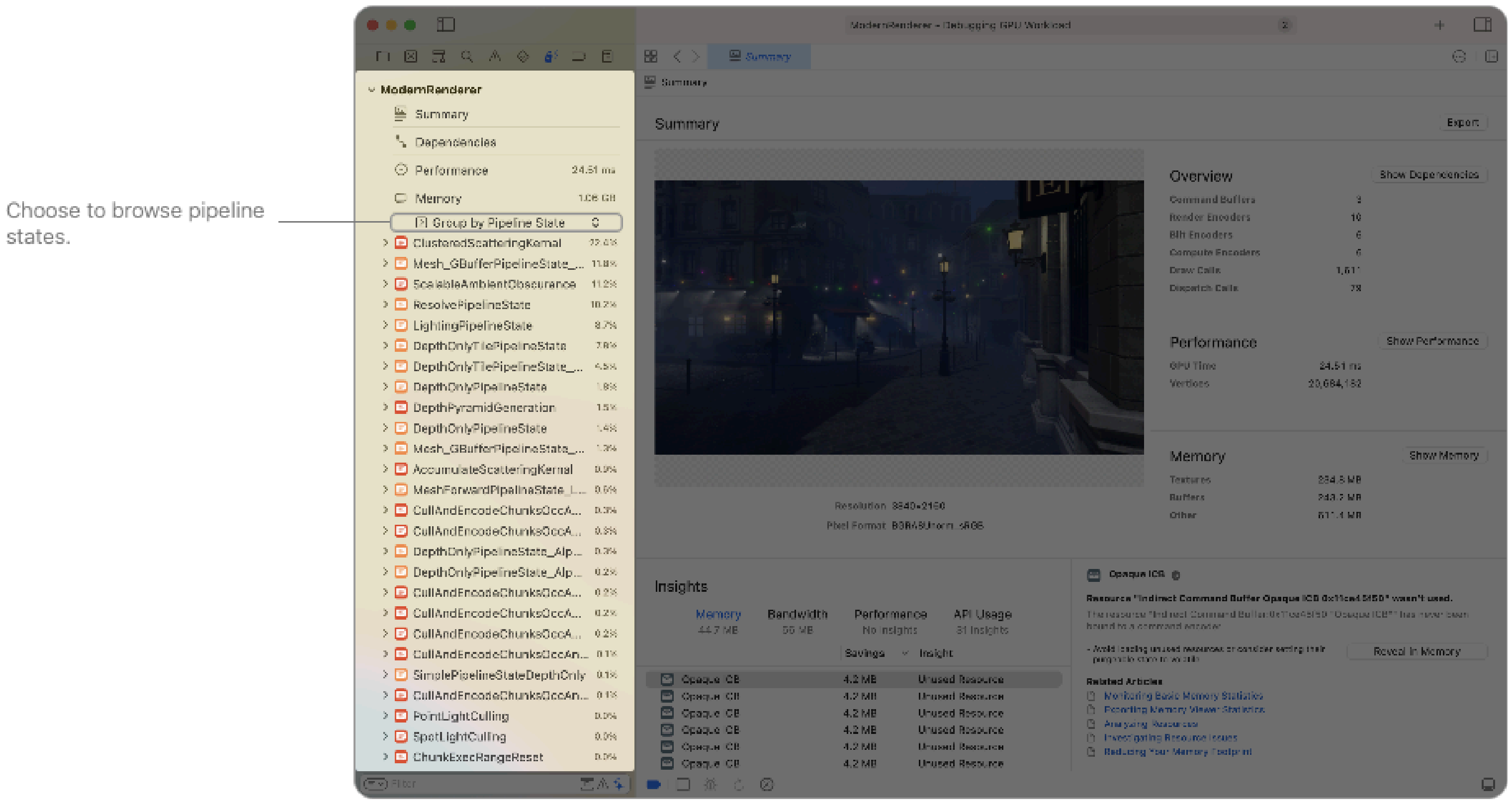
It's common for a render pass to issue a massive number of draw commands. You can skim through the draw commands by moving your pointer over the rows to quickly locate a draw command of interest. The Metal debugger shows a preview of the first attachment in the popover.


758.20 [GBuffer Pass... 0.3%
758.21 [GBuffer Pass... 0.0%
758.22 [GBuffer Pass... 0.0%
758.23 [GBuffer Pass... 0.0%
758.24 [GBuffer Pass... 0.2%
777 [drawPrimitives:Tria... 8.7%
Forward Pass 0.6%
Onscreen Command Buffer

777 [drawPrimitives:Triangle vertexStart:0 vertexCount:3]

Browse pipeline states

Instead of viewing the hierarchy of Metal commands, you can begin exploring your captured Metal workload from pipeline states. Click the Outline popup menu and select Group by Pipeline State.



The Metal debugger shows a list of pipeline states. Expanding a pipeline state allows you to see its shaders. Further expansion shows all the draw commands and compute dispatches that use that pipeline state.

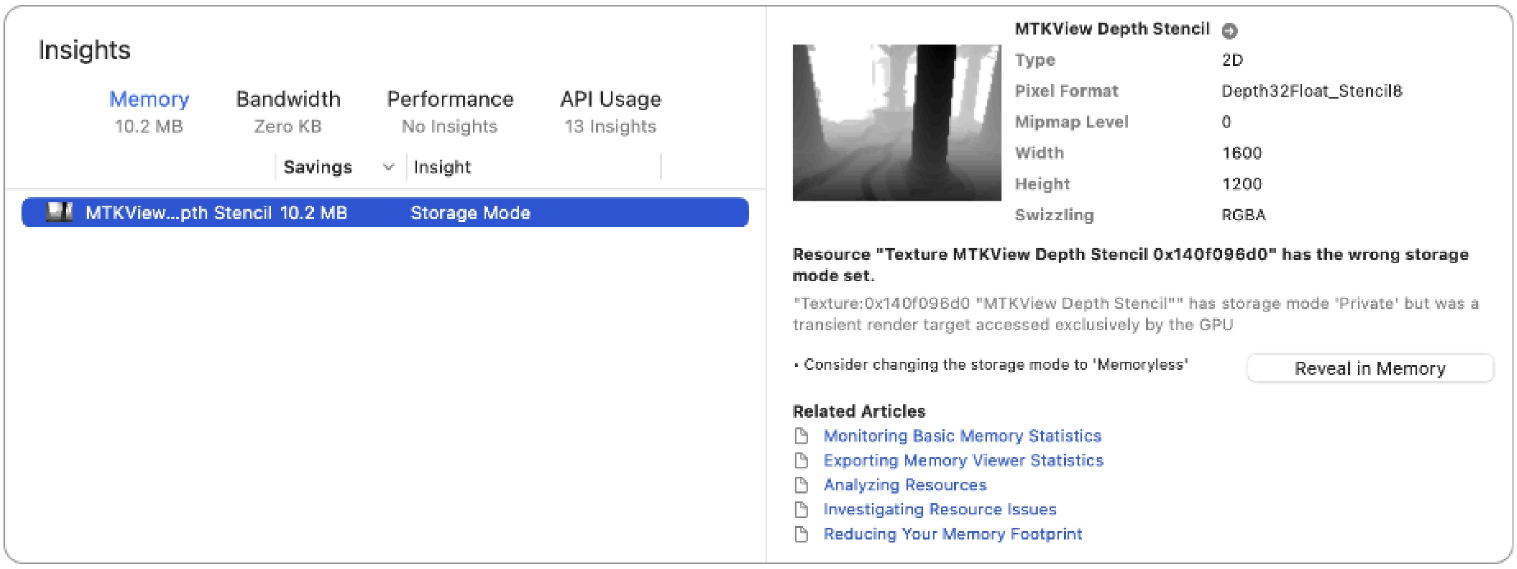
With profiling data, the Debug navigator displays the percentage of samples from the shaders of each pipeline state when running the workload with overlap. In a shader-bound workload, this sorted list of pipeline states is helpful in identifying the most expensive pipeline state.

Improve your Metal workload with Insights

The Metal debugger automatically generates a number of recommendations, called Insights, and includes them at the bottom of the Summary viewer to help you improve your Metal workload. The Insights information includes the following categories:

Memory

Memory insights provide recommendations to reduce the amount of GPU memory that your workload uses. By default, Xcode sorts the recommendations by the amount of memory saved, so you can focus on the biggest potential wins first. In the following example, setting the wrong storage mode results in an additional 10.2 MB of GPU memory usage. The developer of this app can save that memory by following the advice and switching to memoryless.


Insights

Memory	Bandwidth	Performance	API Usage
10.2 MB	Zero KB	No Insights	13 Insights

Savings | Insight

MTKView...pth Stencil 10.2 MB | Storage Mode

MTKView Depth Stencil

Type	2D
Pixel Format	Depth32Float_Stencil8
Mipmap Level	0
Width	1600
Height	1200
Swizzling	RGBA

Resource "Texture MTKView Depth Stencil 0x140f096d0" has the wrong storage mode set.

"Texture:0x140f096d0 "MTKView Depth Stencil"" has storage mode 'Private' but was a transient render target accessed exclusively by the GPU

• Consider changing the storage mode to 'Memoryless'

Related Articles

- Monitoring Basic Memory Statistics
- Exporting Memory Viewer Statistics
- Analyzing Resources
- Investigating Resource Issues
- Reducing Your Memory Footprint

Bandwidth

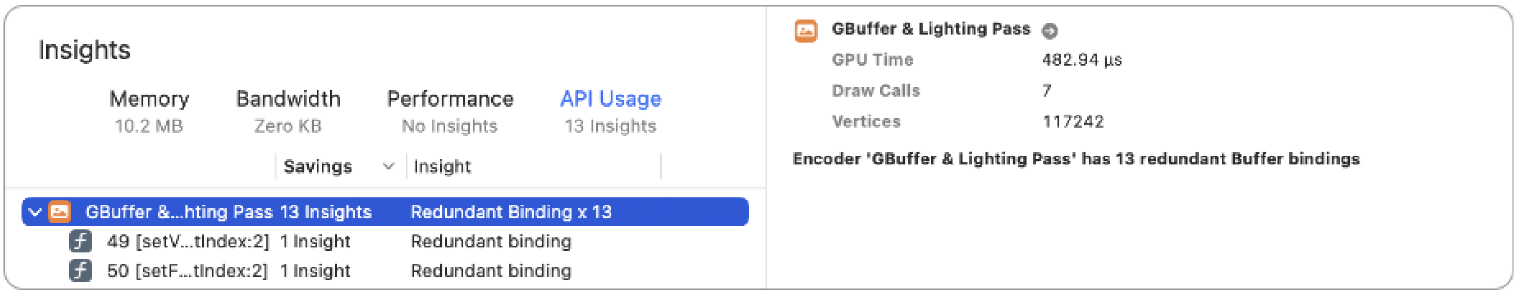
Bandwidth insights provide recommendations to reduce the amount of memory bandwidth, which is particularly important on Apple GPUs. Transfers to and from external memory often result in decreased power efficiency and slower performance. To learn more, see Tailor your apps for Apple GPUs and tile-based deferred rendering.

Performance

Performance insights provide recommendations to increase your app's performance by avoiding expensive and redundant operations in the rendering pipeline.

API Usage

API Usage insights provide recommendations to improve your Metal API usage. Fewer API calls can lead to less CPU time. In the following example, you can see that the vertex and fragment buffers are redundantly bound for each draw command:


Insights

Memory	Bandwidth	Performance	API Usage
10.2 MB	Zero KB	No Insights	13 Insights

Savings | Insight

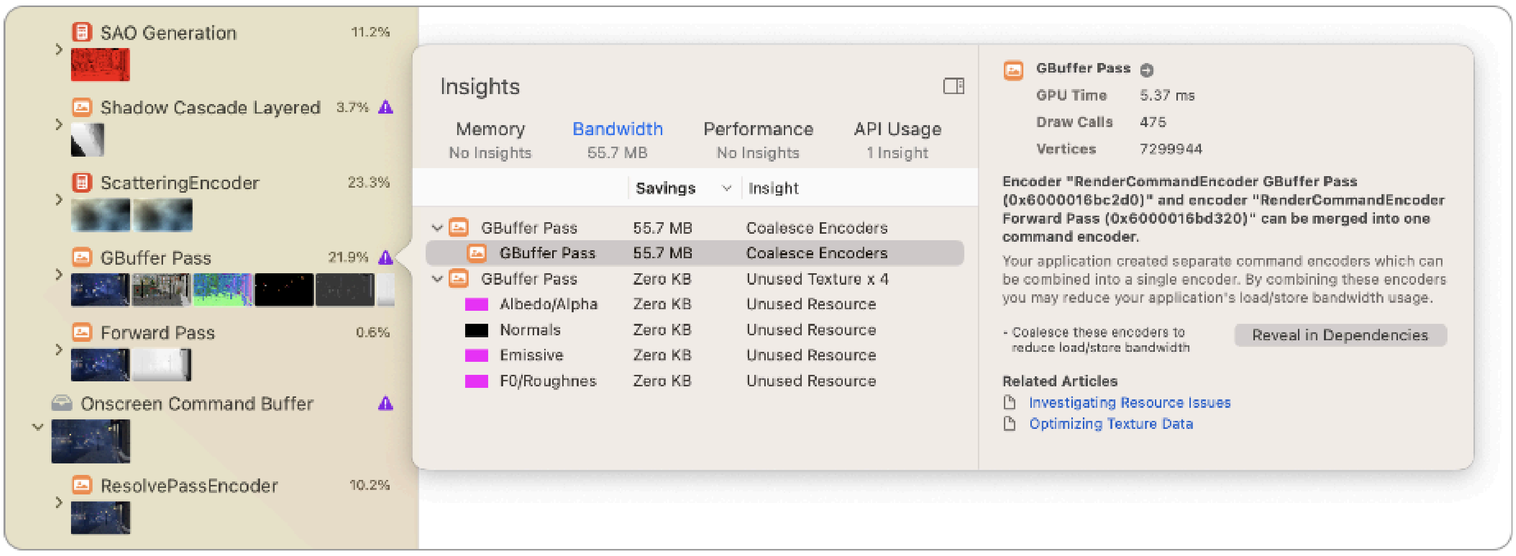
GBuffer &...hting Pass 13 Insights	Redundant Binding x 13
49 [setV...tIndex:2] 1 Insight	Redundant binding
50 [setF...tIndex:2] 1 Insight	Redundant binding

GBuffer & Lighting Pass

GPU Time	482.94 µs
Draw Calls	7
Vertices	117242

Encoder 'GBuffer & Lighting Pass' has 13 redundant Buffer bindings

Click the Insights button next to the command in the Debug navigator to see in-depth details and recommendations that may improve your command.



Insights

Memory	Bandwidth	Performance	API Usage
No Insights	55.7 MB	No Insights	1 Insight

	Savings	Insight
GBuffer Pass	55.7 MB	Coalesce Encoders
GBuffer Pass	55.7 MB	Coalesce Encoders
GBuffer Pass	Zero KB	Unused Texture x 4
Albedo/Alpha	Zero KB	Unused Resource
Normals	Zero KB	Unused Resource
Emissive	Zero KB	Unused Resource
F0/Roughnes	Zero KB	Unused Resource

GBuffer Pass

GPU Time	5.37 ms
Draw Calls	475
Vertices	7299944

Encoder "RenderCommandEncoder GBuffer Pass (0x6000016bc2d0)" and encoder "RenderCommandEncoder Forward Pass (0x6000016bd320)" can be merged into one command encoder.

Your application created separate command encoders which can be combined into a single encoder. By combining these encoders you may reduce your application's load/store bandwidth usage.

- Coalesce these encoders to reduce load/store bandwidth

Reveal in Dependencies

Related Articles

Investigating Resource Issues

Optimizing Texture Data

Limit your scope with filters

Use the filter bar at the bottom of the Debug navigator to adjust filtering criteria. You can type filter terms into the filter bar's text field, and the outline in the Debug navigator shows only rows with labels that match those filter terms.

When there are two or more filter terms, you can click the filter button to choose whether to match any or all of the terms. For any filter term, you can click it to choose to include or exclude resources that match that term.

The following additional filter tools appear to the right:

Show only related stack frames
: After capturing a fresh GPU trace, you can inspect the call stack for each Metal command. This allows filtering for the stack frames in your project.

Show only calls with issues
: This allows filtering for just the Metal commands with insights from the Metal debugger.

Show only markers and commands
: This allows filtering for Metal commands like draw commands, compute dispatches, and blit operations.

See Also

Metal workload analysis

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.